**Evo Manuscript MINI Version August 21, 1997**

***Cover Page: US Letter size version. Permission to copy and spread given. Tom Gilb***

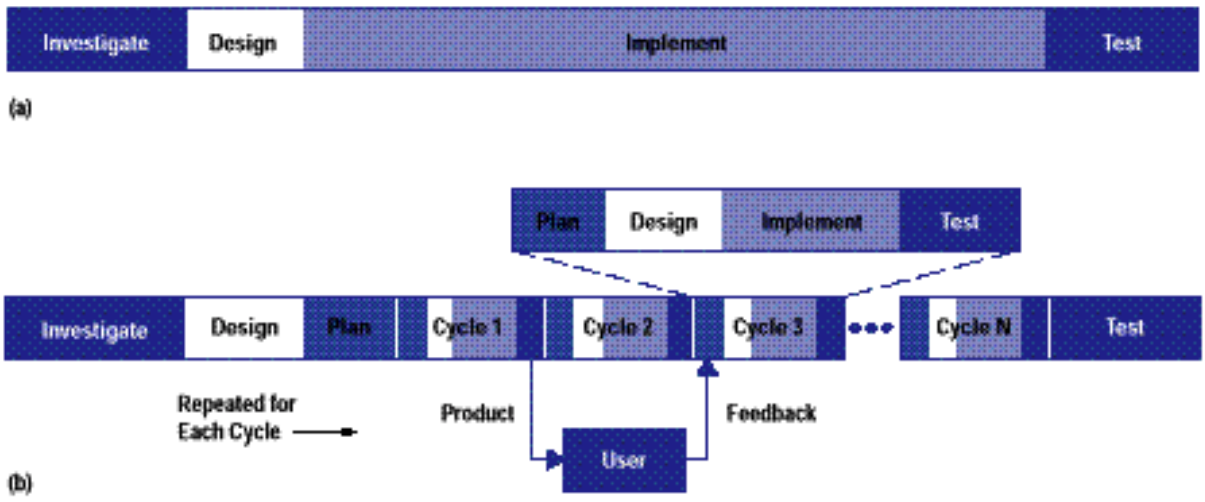
# Evo:
# The Evolutionary Project Managers Handbook


## By
## Tom Gilb

## Gilb@acm.org


**Rules: = Glossary Rules**
**DQC: none, author check.**
**Comments and advice on this manuscript are always welcome! Send to Gilb@acm.org**

Project life cycles. (a) Conventional project model. (b) Evolutionary (Evo) project model. [MAY96]



## *Table of Contents*

## *Introduction Page*

Evolutionary Project Management ("**Evo**") is a significant step forward in managing complex projects of all kinds. It promises and delivers earlier delivery of critical results and on-time delivery of deadlined results. It can be used for getting better control over quality, performance and costs than conventional project management methods.
It is particularly suited to complex technology, fast-moving environments, large scale projects. But, it certainly works well on a small scale too.
The key idea of Evo is "learning" and consequent adaptation. It is about learning about realities as early as possible and taking the consequences of any project reality, external or internal, and making the most of that information.
Evo is a 21st Century project management method, it promises both rapid results, and rapid response and adaptation to unforeseen circumstances: both good and bad.
Evo is simple and natural, but we still need to learn and master it.

This book will provide an in depth handbook for the project manager and for the training and reference of anybody who needs expertise in Evolutionary project management methods.

*"Evolutionary Development has been positioned here'* [in cited HP Journal article] *'as a life cycle for software development, but it really has much broader application to any complex system." [COTTON96].*

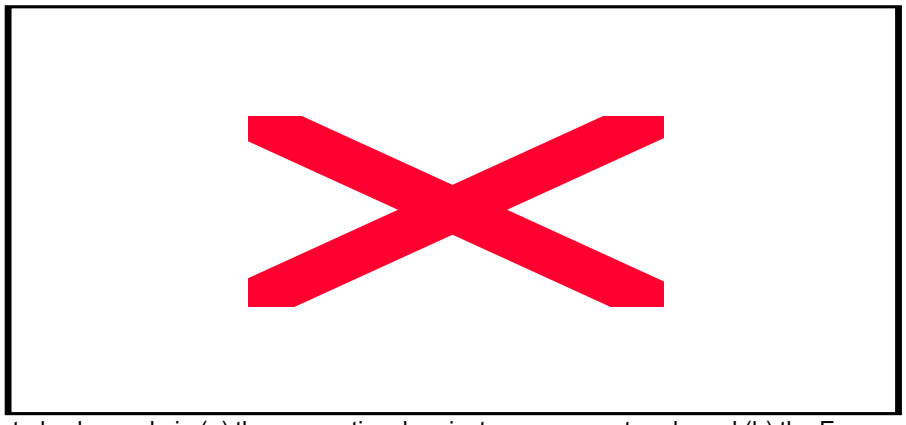


Fig. An accelerated sales cycle in (a) the conventional project management cycle and (b) the Evo cycle.[MAY96]. HP cites Evo as opportunity experienced to start the product sales cycle early and generate income earlier than usual.

*"Companies that have adopted similar incremental development methods include computer hardware vendors (such as Hewlett Packard and Digital Equipment Corporation), computer systems integrators (EDS and SAIC), aerospace and (TRW and Hughes) and electronic equipment manufacturers (Motorola and Xerox). Microsoft has been extremeley effective, however, in creating a strategy for product and process definition that supports its competitive strategy." MS. 188*

## *Chapters*

# 0: Overview. The essential character of Evo.

## *Evo Scope: What can you use it for*

Evo can be applied to almost any project management task. It has been applied to large and small scale software engineering tasks, to aircraft engineering, to telecommunications engineering, to military weapons projects, to organizational development projects, to environmental projects, to aid projects, to peace process planning, to electronics system projects, to Information System projects, to air traffic control projects.

It seems suited for almost any type of project, and seems to give better results than conventional planning approaches.

One could ask what is *not* suited for Evo planning?
I don't know the answer to that. I have not seen failure of Evo projects, which clearly had connection to the Evo method itself. But far more extensive use of the method might give us some answers.

## *Evo Focus: What are we going to focus on in this book?*

We are going to concentrate on helping *project managers* set up and run Evo projects.

## *How does it work? Fundamentals.*

Evo is identical in *concept* to the "Plan Do Study Act" Cycle which W. Edwards Deming and Walter Shewhart taught the manufacturing community and many others [DEMING86], notably in Japan and the United States. A frequent series of statistically significant measurements lays the basis for understanding how things are going, compared to expectations. Negative deviation from expectations allows you to 'act' to correct the situation.

From a *project manager* point of view:

1. Long term **goals**[1] for project success are determined. These can be improved anytime.
2. Small (2% or so of project resources) partial delivery **steps** towards the long term goals are carried out.
3. As long as a step is successful, new small steps are taken until the goals are reached.
4. If a step gives unexpected results, plans are re-evaluated (causal analysis, why?) and future step design is improved. A new step is tried.

> *Evo "involves a series of incremental deliveries. Each delivery contributes an operable, functionally valuable, partial system. The overall system is developed and delivered to its users (and thereby contracturally delivered to its sponsor) in small evolutionary increments. The users employ the evolving system in the daily conduct of their mission."*
> *[SPUCK93], Jet Propulsion Labs, JPL, on Rapid Development Method RDM hereafter called 'Evo'.*

The process resembles maintenance and enhancement of existing systems. The major difference being that there is a long term set of objectives for change, and a long term architecture to support them.

### Structure Models

**Conventional project model**

| | | | |
|---|---|---|---|
| Frozen | Frozen | Build to design | =Requirements? |
| Requirements Analysis | Design Engineering | Construction/Acquisition | Test (system, acceptance) |

[1] Terms in **bold type** are defined in the glossary.

In the conventional model the construction is one long event, based on the design, which is based on the requirements specification.

**Incremental Development Model**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Complete Detailed Frozen | Complete Detailed Frozen | Build/test | Build/test | Build/test | Build/test | Build/test | |
| **Requirements Analysis & specification** | **Design Specific-ation** | **Step 1** → | **Step 2** → | **Step 3** → | **Step 4** → | **Step n** → | **Accep-tance Test** |

In the 'Incremental Development' model multiple build-and-test steps are based on detailed requirements and design specifications. These are more or less 'frozen'. The point is gradual delivery of partial results. Incremental project management models might be *forced* to revise their requirements and design, but they do not *intend* to, and it is not a regular part of the process.

> *"We assert that, in the case of an important class of systems, namely those that automate human functions, it is unreasonable if not impossible to expect system users or operators to be able to state Final Operating Capability requirements up front. An evolutionary approach is essential. This is true because staff functions change, user insight into operations increases, and concepts of operation are modified by the introduction of automation. Further, needs that are rejected as impossible, beyond the existing technology base, or simply heretofore inconceivable under Conventional Development Methods often are perceived as achievable at some point under [Evo]." [SPUCK93]*

**Evolutionary Development Model**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Best guess. Updated stepwise. | Best Guess. Updated stepwise. | Require-ments Design Build, Test | Require-ments Design Build, Test | Require-ments Design, Build, Test | Require-ments Design Build, Test | Require-ments Design Build, Test | |
| **Requirements Analysis & specification** | **Design specs** | **Step 1** → | **Step 2** → | **Step 3** → | **Step 4** → | **Step '50'** → | **Contract Acceptance Test** |

In the Evolutionary Development model, less effort is put into the initial overall system level requirements and design specification, initially. These specifications are, however continually updated as step experience dictates. At each step, *detailed* requirements and design for the step are specified. These will be a function of experience to date, of new user needs, new technology and economics, and new market insights. The emphasis is on learning rapidly, and applying the lessons for better satisfaction of the customer, as well as better ability of developer to manage the project.

> *"Progressive Formality.*
> *Finally, the fourth defining tenet of Evo [at JPL] is progressive formality. Under Evo, the first delivery will be executed quite quickly and with very little formality, much like a rapid prototype. As succeeding deliveries are undertaken, implementation procedures become more formal and comprehensive. Under conventional project management procedures and products must be done perfectly before the next step in the implementation cycle can begin; for Evo they are implemented under a planned progression of thoroughness, so that at the final delivery the two methods converge to the same degree of formality." [SPUCK93]*

**Head-and-Body Evo Model**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Head | Head | Body | Body | Body | Body | Body | Head |
| | | ←**experience** | ← | ← | ← | ← | |
| **Requirements Analysis & specification** | **Design specs** | **Step 1** → | **Step 2** → | **Step 3** → | **Step 4** → | **Step n** → | **Acceptance Test** |
| **Needs** → | **Ideas** → | **Micro-project** | **Micro-project** | **Micro-project** | **Micro-project** | **Micro-project** | **Product ship** |

The Evo process is characterized by two major components. The 'head' which is the 'brains' behind the project. This operates at the level of project manager and systems architect.

The 'body' is the 'muscles' of the project where the action is, where the project confronts the real world (not necessarily real customers) and creates change. The steps in the body are such complete small projects, that I sometimes call them 'microprojects'.

> *"We have labeled Microsoft's style of product development the 'synch-and-stabilize' approach. The essence is simple: continually synchronize what people are doing as individuals and as members of different teams, and periodically stabilize the product in increments – in other words, as the project proceeds, rather than once at the end." MS, 14*

The head continuously receives feedback from the body, and integrates these data with long term overall project plans, external new information, draws conclusions about which new Evo steps should be next and what their requirements and design should be.

> *Evo "expects active feedback from the experience gained from one incremental delivery to the requirements from the next. As Evo periodically delivers to the users an increment of capability, the users are able to provide understanding of how effectively that delivery is meeting their needs. As the users assess the impact of a delivery on their operations, the system developer is able to work with them to adjust the system requirements to better satisfy their operational needs. Evo lets that adjusted set of requirements be the basis for all subsequent incremental deliveries. This feedback process is formal and proactive. It is a key element in making Evo effective from a user's perspective." [SPUCK93]*

## *Variations in the 'Evo process' structure.*

### Step Size Variation

The delivery step *size* may vary. It is usually in the range of 2% to 5% of total project cost budget.

> *"Mike Conte, a senior program manager for Office "We actually break our development into three separate milestones. They might be six week milestones, [or] they might be ten-week milestones ... At the end of the milestone our goal is to get all the features for that milestone that have been implemented ... for that milestone at zero bugs…. And then, when we get to the point where we get to 'ship quality', we can move on to the next milestone. The point of this is that we never get so totally out of control that we're at the end of a project and we have so many thousands of bugs that we can't ever tell when we're going to finish it." MS, page 200*

The *time* between delivery steps is usually in the range of a weekly cycle to a monthly cycle. The step size is mainly a function of the risk the project is willing to take, the risk of wasting a whole step, before learning that it is a loss.

> *The general rule of thumb is to keep the cycle length as short as possible. Within Hewlett-Packard, projects have used a cycle length as short as one week and as long as four weeks. The typical cycle time is two weeks. The primary factor in determining the cycle length is how often management wants insight into the project's progress and how often they want the opportunity to adjust the project plan, product, and process. Since it is more likely that a team will lengthen their cycle time than shorten it, it is best to start with as short a cycle as possible. [COTTON96]*

### Existing Base Exploitation

The Evo project will usually *start from the base of the existing system* or product of existing users or customers. Only rarely will it start from scratch. This applies even if the final architecture is radically different from the existing architecture.

This has a variety of reasons. The existing users provide a realistic *field trial* of the delivery steps. They provide early *relief* for existing users of known problems. This may provide *early* concrete *product* for new users and markets. They may provide *income* as a result of the improvements. They may co-operate in providing *insight* as to what they really want.

### Parallel Evo Project Paths.

Two or more parallel evolutionary projects or sub-projects can be synchronized towards reaching some common objectives.

These parallel Evo projects can for example be necessitated by needing to deal with different markets, customers, development sites, different product characteristics, different internal organization components (marketing, sales, development, top management, support and service). But, they would

ultimately be synchronized towards the achievement of some goals which only the combination of efforts can attain.

*"Evo allows the marketing department access to early deliveries, facilitating development of documentation and demonstrations. Although this access must be given judiciously, in some markets it is absolutely necessary to start the sales cycle well before product release. The ability of developers to respond to market changes is increased in Evo because the software is continuously evolving and the development team is thus better positioned to change a feature set or release it earlier." [MAY96]*

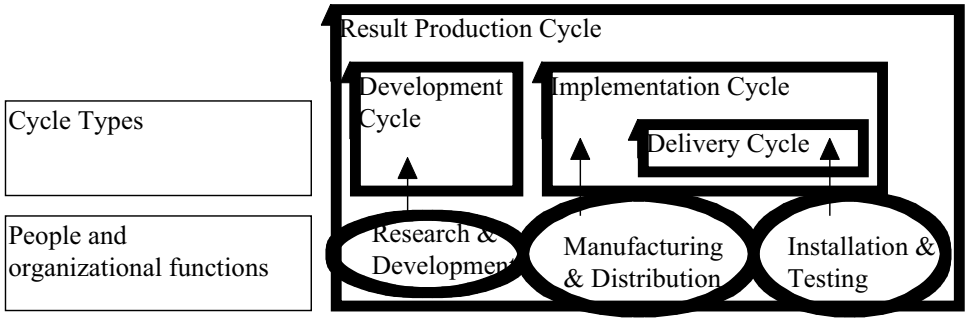
#### Backroom Development, Frontroom Delivery.

The Evo deliveries are a *'cyclical change'* from the user/customer/potential market ("**recipient**") point of view. Some of the components which make up a delivery step may take much longer time to purchase or build, than the **delivery cycle** timing. These are prepared in the background ('**backroom**'), early enough to deliver at the appropriate step, from the '**frontroom**'. The time needed to get backroom step components ready for delivery is invisible to the step recipient. Until the step components are actually ready, they cannot be considered for scheduling as a frontroom step delivery.

## *Advanced Points*

#### Delivery Step is 'change', not necessarily 'construction'.

An Evo delivery step, the *output* of a delivery cycle, is the **frontroom** interface to the **recipient**. There are many other processes which may lead up to the delivery cycle. An **implementation cycle** would cover manufacturing and distribution, but not creation of the measurable end results, as required in the goals. A **development cycle** would take care of research and development of a product or product components, which could be 'soft' as well as 'hard'. The **result production cycle** contains all processes needed to get the goal defined results.




*The Result Production Cycle, its components and related people functions.*

*"Major Milestone Releases. A project organizes the development phases around three or four major internal releases, or 'milestone subprojects'. Microsoft intends these releases to be very stable. Theoretically, projects could ship them to customers, as Chris Peters observed "... What you do is you essentially divide a project into three [or so] pieces ... and you pretend to ship the product after each piece." MS, page 200*

#### Open architecture.

The Evo process *thinks* it knows *something* about ultimate goals and needed **architecture**. But Evo also knows that it does not know everything, perhaps nothing at all, for *certain*. Any goal can be changed for a number of reasons at any time. This 'goal change' alone might require change in any **design** or any **plan**.

The consequence of this natural uncertainty, is that necessary changes might be painfully difficult and costly. Yet, by definition, these changes cannot be foreseen and the design cannot be tailored to accept them. A multinational telecommunications client reported to me in 1997 that they had found that 70% of their *approved* requirements never 'survived' to the final product!

So, we need to give some priority to design ideas which are generally good at accepting changes. I call this open architecture. The architecture is **open-ended** for many types of change, such as volume

increases, logic changes, extension, reduction, porting and any other types of change for which the specific open-ended design ideas are applicable.

Open-ended design is a specific *investment* in structure, interfaces, languages, contractual arrangements and any other device which can promise to reduce the costs of specifically unforeseen changes to goals and designs. See Chapter 2, Design, for more information.

*"When requirements uncertainty indicates an Evolutionary Acquisition approach, the program may involve little or no advanced development. In contrast, when technological uncertainty indicates an evolutionary approach, significant amounts of advanced development are ordinarily involved. Indeed, the evolutionary strategy has been derived as a means of dealing with just such uncertainties because development periods involved in making very large or "revolutionary" jumps at the limits of a state-of-the-art take so long and are so risky that U.S. readiness is being threatened.*

*"While it is highly desirable that users be constantly knowledgeable about programs with technological uncertainty _ indeed play a continuous, if reactive role in the acquisition of any DoD system - the approach for these programs does not require user acceptance of any significant responsibility at any stage of the acquisition cycle. In contrast, for programs with requirements uncertainty, succeeding blocks of work after the first cannot be adequately specified until feedback from some user is received on the usefulness and needed modifications to prior blocks." [DODEVO95] quoting from other sources*

### The Evo measure of project 'effectiveness' is targeted *results*.

Most conventional planning methods seem to acknowledge activities, such as document production, construction, testing, design, requirements specification, quality control, analysis, meetings, approvals of ideas, and other such intermediary tasks, as a 'measure' of project progress. Unfortunately they are indirect and give no guarantee that the ultimate step recipient, or the ultimate project **funder**, will be pleased.

*"It appears that this incremental approach takes longer, but it almost never does, because it keeps you in close touch with where things really are" Brad Silverberg, Sr. VP for Personal Systems Microsoft in MS, page 202*

Evo acknowledges only one measure of **effectiveness**. It is 'measured progress towards defined **target** goals at the **recipient** level'. User improvement. Customer improvement. As viewed by *them*, and *their* formal agenda.

By this criteria, most projects make no progress whatsoever until long after initial deadlines, if ever.

*"Costs of Evo:*
*Adopting Evolutionary Development is not without cost. It presents a new paradigm for the project manager to follow when decomposing and planning the project, and it requires more explicit, organized decision making than many managers and teams are accustomed to. In traditional projects, subsystems or code modules are identified and then parceled out for implementation. As a result, planning and staffing of large projects were driven by the structure of the system and not by its intended use. In contrast, Evolutionary Development focuses on the intended use of the system. The functionality to be delivered in a given cycle is determined first. It is common practice to implement only those portions of subsystems or modules that support that*
*functionality during that cycle. This approach to building a work breakdown structure presents a new paradigm to the project manager and the development team. Subsystem and module completion cannot be used for intermediate milestone definition because their full functionality is not in place until the end of the project. The time needed to adopt this new paradigm and create an initial plan can be a major barrier for some project teams." [COTTON96] HP*

### The Evo measure of project 'efficiency' is targeted *results in relation to resources needed to deliver them.*

The secondary measure of a project is **efficiency**. This is the effectiveness (delivered recipient-stipulated results) in relation to all 'resources consumed' to deliver the results. This is a measure of project profitability, or of return on investment.

### Universality of application.

Evo is a project management tool which can be used for any project management task. It is not limited to any technology or any application area. It has been shown suited to multiple discipline projects,

software projects, organizational improvement projects, environmental improvement projects, peace planning projects and personal life planning 'projects'.

I am not making a statement that Evo is 'best suited' for any given project task, merely that Evo cannot be excluded as a candidate to manage the projects. The 'best-suited project management method' selection will depend on many other factors, including convention, culture, law, stipulation, and availability of other methods.

## *What is the rest of the book about?*

The rest of this book will:

- Show you how to define Evo targets, that is 'requirements'
- Show you how to find and evaluate ways to deliver your target levels, that is 'design'
- Show you how to convert 'design' into Evo result delivery steps.
- Show you how to manage Evo project progress.
- Explain how to make Evo a part of your company and project culture.

Evo is simply about gradual delivery of desired results.

## 1: Requirements at Project Level: The Evo direction.

The main point of difference between Evo and other project management methods is that 'requirements achievement' dominates, rather than formal processes (such as 'approve design', or 'conduct field trials'). The 'Evolutionist' (one who does Evo) can use *any* form of requirements specification they wish to. But, my experience is that most cultures have terrible specification habits. Peter Morris in his excellent book 'The Management of Projects' [MORRIS94] named *requirements* as top of the list of variables which have influence on project failure or relative success. So, this chapter will give some advice about what I consider good habits.

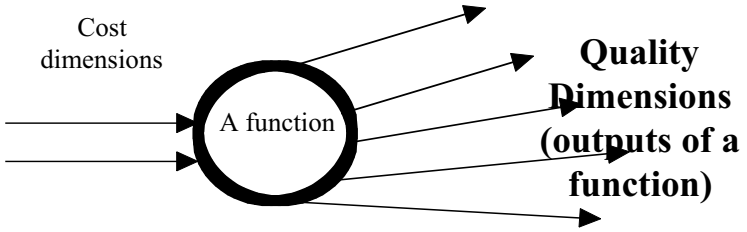
The problems with requirements?

- Lack of clarity
- Lack of known quality-controlled defect level
- Lack of testability
- Ambiguity
- Inconsistency
- Incomplete, missing
- Specification of perceived means, rather than real ends.
- Lack of measurability for variables
- Lack of information about priorities
- And much more …

> *"We are continually amazed at how many managers fail to achieve results simply because their employees don't understand the desired goal state." [Kaplan94, p.203] IBM STL*

The method below is based on my earlier writings [GILB], some of which are more detailed. But, this should be sufficient for our current purpose. The Glossary reflects the full requirements and design language [Planguage] and so it can hint as to some of the specification ideas not covered here directly.

### *The Primary Drivers: Quality.*




*'quality' is any variable result from a system.*

The fundamental reason why projects are created and funded is to *achieve some improvement*. So, the fundamental *measure of progress* of a project is to see if the *desired improvement* is being reached in practice. I use the term ***quality*** *attributes* to describe all *variable* outputs of a defined function. A **function** is *what* a system does, and change in function is one possible type of requirement, treated in this chapter later. Right now we are going to concentrate on *how well* a function performs: *quality*.

All qualities can, by my definition above, be specified measurably. That is we can speak about degrees of a quality using numbers.

Many people do not believe or understand this fundamental principle of qualities, because of lack of training or experience. They speak about 'qualitative' ideas, meaning they do not recognize that they can be articulated with numbers.

How do you feel about a requirement to make a 'highly adaptable' system? 'Highly' speaks to us of degrees, but few are accustomed to using a scale of measure for 'adaptability'. Yet adaptability is no

more complex than a notion of 'a degree of *time, human effort or money* to adapt some system to another environment'. For example to 'adapt' application software to another hardware or operating system software environment.

### Quality Specification Templates.

To specify a quality requirement we need :

- To 'name' the requirement, so it can be referenced
- To define the units of measure we will apply
- To specify the performance level we will require on that scale of measure

For example:

| |
|---|
| Adaptability:<br>Scale: Engineering Hours needed to modify product for a new market<br>Plan [Product XX, Market YY] 1,000 Eng. H. |

That was a minimal simple example, here is a 'whistles and bells' example:
(all the parameters 'Gist, Scale, … Must, Plan' are defined in your Glossary)

| |
|---|
| Usability:<br>Gist: the relative ease of learning and using a defined product compared to previously used products.<br>Scale: average minutes per [defined User type] to learn to use [defined Tasks to use the product].<br>Meter: at least 30 users of representative defined User type will be monitored doing at least 10 defined Tasks of defined function type.<br>Past [Old Product PP, Home Buyer, Adult, Task: Build telephone number list] 30 minutes.<br>Record [MM, Adult, Task: Dial Out] 10 seconds ← Consumer Reports, January<br>Trend [US Market, Children, Mix] 20 minutes ← Market Analysis Feb.<br>Wish [Our Customers, Mix] 5 minutes ← Chairman's Dream in Speech<br>MinLevel: Must [Our Customers, Mix, New Product] 10 minutes ← marketing minimum<br>Plan [Our customers, Mix, New Product, First Field Release] 50% of MinLevel? ← Guess by Project Mgr., [within 2 years of First Field Release] 30% of MinLevel ←Guess.<br>Local Definitions of Terms.<br>Mix: Defined: representative mix of common frequent user tasks.<br>User: Defined: person who intends to use the product in the long term, not a test person.<br>First Field Release: Defined: First sold releases to any public market after Field Trials. |

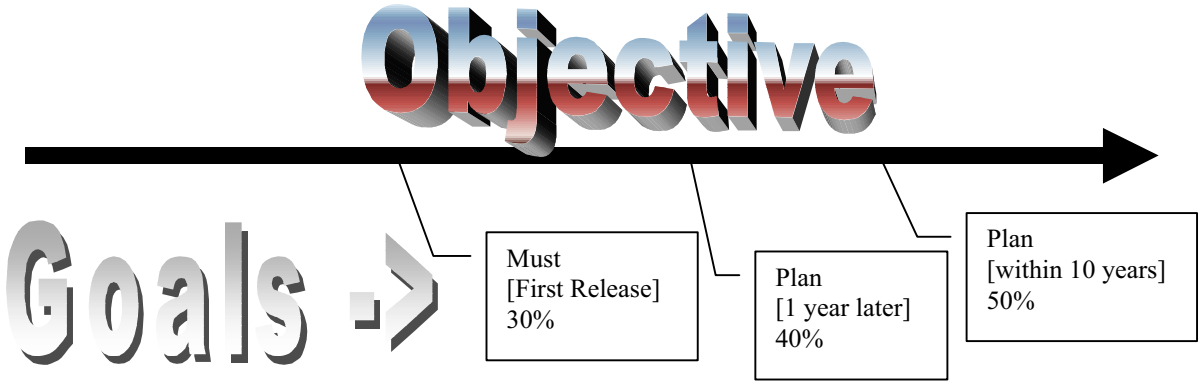
By adjusting the **qualifiers** (stuff in [square brackets] which define *where*, *when* and *under which conditions*) *and* the **Scale** definition, the **Meter** Definition and the up to six types of levels (**Benchmarks: Past, Record, Trend, and Targets: Wish, Must, Plan**), you can specify practically any quality in clear detail.

You can do this for the top ten to twenty most-critical quality requirements. You will then have the major criteria defined.

You must then control your project Evo steps, so as to move in the direction the *Must* and finally the *Plan* levels. For project success, you must at least *meet* the Plan levels. To avoid failure you must at least *meet* the Must levels.

### Goals and Objectives.

**Objectives** such as 'Adaptability' and 'Usability' above can contain any useful number of **targets** for the future. Each of these targets, in a [qualified] Must or Plan, possibly Wish, statement is a separate '**goal**'.
An Objective can be viewed as a collection of related destinations which the evolutionary project must visit on its way to the ultimate goal statement in it.




### Priority Statements.

If we define a '**priority**' as 'something which causes us to apply scarce resources to attain', then we can see that objectives *contain* priority statements in the form of their 'goal levels'.

The *existence* of a goal causes us to prioritize action to attain it. The *level* of a goal will tend to require more resources, the higher it is.

Project managers must *identify* requirements, so they can *prioritize* project resources. Initially the project priorities, in the form of requirements, drive project management to supply resources for adequate *design* to meet the requirements. Finally, the priority levels force project management to prioritize adequate resources to *implement* the design at a step of delivery.

> *"Development Phase: Feature development in 3 or 4 sequential subprojects that each results in a milestone release.*
> *---*
> *Program managers coordinate evolution of specification.*
> *Developers design code and debug.*
> *Testers pair up with developers for continuous testing.*
> *---*
> - *Subproject I First 1/3 of features. Most critical features and shared components.*
> - *Subproject II Second 1/3 of features.*
> - *Subproject III Final 1/3 of features. Least critical features."*
>
> *MS, page 194. The point I want to bring out here is the priority sequencing rule at Microsoft. A milestone is a 6-10 week segment.*

Notice that priority is a *variable*. It is a function of goal levels set. Then it is a function of the degree of *satisfaction* of those goals. Priority amongst objectives is relative to the *degree* of satisfaction given to their Must, and then Plan levels.

From the project manager point of view, the requirements are a primary tool for allocation of all resources at all times. The requirements are the 'voice of the customer' and should help to keep the project manager on that track.

> *"Change to functional requirements (especially additions to current requirements) can be controlled by accepting only very important changes. The philosophy of permitting only crucial requirement changes is essential because:*
>
> *Feedback on effectiveness and suitability from actual operations and maintenance is almost always required to determine the value of proposed changes with any degree of certainty. For programs with short times between development increments, deferring requirements changes until the next program increment might be a better course of action because it preserves schedule and does not place delivery and fielding plans at risk. However, preserving schedule is of little value if feedback indicates an inability to meet or sustain specified performance thresholds or a lack of logistics supportability." [DODEVO95]*

### *Conflicting Requirements.*

In any real project there will be a large number of quality requirements. Approximately 10 to 20 quality objectives seem to cover the most critical factors in any project. Each objective can define many goals.

Each one of the goals demands some resources (people, time, money) for attainment. Because we always have finite resources, and infinite ambitions (direction: perfect quality) there will come a point where 'we cannot have it all'. There will not be enough of some resource to satisfy a stated goal.

This conflict of goals for finite resources is natural and inevitable. You cannot drop a requirement merely because it is in conflict with another conflicting requirement.

Some intelligent resolution of the problem is called for. *Early* resolution senses the problem at the *design* stage. *Implementation* stage resolution of conflict can be *costly*, although with Evo methods the problem is not as serious as it would be with conventional (at end of project) conflict resolution.

We will show the use of Impact Tables, to detect conflicts both at design stages, and between Evo steps; in later chapters. The important point is that clear measurable requirements specifications lay the groundwork for such conflict resolution.

*"Microsoft managers also try to 'fix' project resources - limiting the number of people and. amount of time on any one project. The fixed project resources thus become the key defining elements in a product development schedule; in particular; the intended ship date causes the whole development team to bound its creativity and effort. The team must define intermediate steps and milestones that work backward from the target ship date, and co-ordinate product delivery with other Microsoft projects, product distributors, and third-party system integrators. Projects accomplish these goals even though the intended ship date almost always changes."*
*MS, 188-9*

### *The 'Costs' of Evolving: 'Resources' and 'Inputs'..*

Requirements can also include specification of the 'fuel' needed to both build and operate a system at the levels of quality required. In the simplest form, any cost element can be stated as a generic **cost constraint**. For example, 'The development budget must not exceed last year's budget'. But more articulate requirements specification is necessary and desirable in Evo planning. It is needed because of the many Evo steps which each have *individual* resource consumption. It is also needed to articulate *operational* costs.

The format shown for quality requirements can be used to articulate specific resource requirements.

A simple example:

Development Budget Spend:
Scale: Money committed (even if not paid out yet).
Plan [by Successful Project End, Meeting top 5 critical Plan Levels] 1,000,000.

Notice that even this simple example contains richer specification detail and control than the usual notion of 'The budget is one million'.

Here is a more complex example:

Engineering Hours Planned:
Gist: the load in qualified engineering hours.
Scale: Engineering Hours allocated to [defined Tasks] under [defined Conditions].
Meter [Project Manager Level] Weekly worksheets reports.
Must [Entire Project, IF Successful] 100,000 E-hours, [Phase 1] 10,000 E-hours.
Plan [To achieve Plan Levels for all Quality Requirements] 80,000 E-hours,
[To achieve Plan Level for Performance [Phase One] and Availability [Initial Delivery] alone] 40,000 E-hours.

It should be obvious that you can use the Planguage to allocate resources for different phases of a project and to allocate resources for defined levels of quality results. "Performance [Phase One]" means, for example the Plan level of "Performance" (as defined by it's Scale) and relating to the Plan for "Phase One" only. The use of Capital Letters in a term implies that the term is formally defined.

### *The 'Generic Constraints' as Requirements.*

Generic Constraints (for short, just 'constraints') are 'framework' requirements. They put a fence around our project, but we are simultaneously free to additionally set more-specific requirements, than the generic constraints, for specific times and conditions.

There are many additional ways of categorizing constraints. For example:

- Quality
- Cost
- Design
- Function
- Legal
- Ethical
- Local Community
- Social

No attempt to be comprehensive is made here.

There are a variety of ways to specify constraints. For example the 'Must' level of a requirement is one sort of constraint

| |
|---|
| Usability<br>Scale: minutes to learn [a defined task] by [defined users]<br>Must [All users, All Tasks] less than 1 hour.<br>Plan [Novice Users, Basic Communication, First Release] 1 minute. |

The advantage of this format is that it is integrated with other specification. But it does not apply to all types of constraint specification.

Another format is the straightforward "Constraint" specification. In my practice organized into groups. For example:

| |
|---|
| Generic Constraints.<br><br>Design Constraints.<br>Spruce Goose: Constraint: The design must not use metal as far as possible. ←DoD<br><br>Cost Constraints.<br>Budget Maximum: Constraint: The total expenditure for all aspects of the project cannot exceed the official budget amount under any circumstances.←CFO<br><br>Legal Constraints.<br>Safety Regulations: Constraint: The design must adhere to all known and projected safety regulations in all projected markets as delivered from factory.<br><br>Child Seats: Constraint [First Release, USA, CA] No front seat child seats must be possible in our design. <-- California Statute 702 paragraph 16, applies only next year. |

Constraints are not targets for the Evo project to aim at reaching, but are more like 'walls' and 'fences' to keep within, as the Evo project plunges ahead.

Specifically the Evo project needs to use quality control to assure that no plan or design inadvertently steps outside these fences. The test process also needs to systematically test that the constraints are respected in practice.

## *Function: 'What' a system does.*

The final requirements category is *functionality*. I define this as the raw 'what the system does', separated from all other attributes, such as 'how well it does it's function' (quality), or 'how much it costs to do so well' (cost), or 'how it manages to do so well as that cost' (design).

Many people get function and design mixed up. The way to determine the difference is the question, 'why must it do that function?". If you get answers like
"Because that is an immutable part of being what it is", it is a function.

If you get answers like "so it can be good at doing something"
It is probably 'design' in order to achieve some quality or cost aspect of the system.

For example:
Is an arm of a human function or design? Design! To help us lift and move etc. heavy things.

Is a brain of a human, function or design? Function, an essential element. No brain, no human.

In human made systems, this distinction is sometimes arbitrary: is the fifth (spare) tire an inherent part of the car, a design to improve maintainability (of flat tires) and to improve availability of the car? Or, is it a legal constraint?

### Levels of Perception

Sometimes a 'design' at one level of human thought, must be viewed at an inherent function at the next level of thought. For example keyboards and screens seem part of the inherent function of a computer these days, but one has only to play with the handwriting pen-based input of an Apple Newton, or voice input to realize that the keyboard is only a convenient design, as long as technology will not allow us more natural ways of communicating with computers.

### Step Content

From an evolutionary point of view, a system may evolve by increments of *both* function and design. Increments of design are expected to improve the quality and costs of the system. Functional increments or changes should normally affect the area of *how* the system can be used with these qualities and costs.

> *"The delivery schedule is generally quite arbitrary. It is determined a priori that deliveries will occur .. at whatever… interval .. the project and sponsor select. … This approach might be called 'design to schedule' because only those functions that can be implemented in the allocated time are candidates for the next delivery." [SPUCK93]*

### Simplifying the Functionality Definition

Sometimes, I find it useful to ignore formal specification of function at all. I will identify the system in question, and declare, or quietly acknowledge, that 'it has the functionality which it has', but that I will concentrate on making it better (qualities and costs). This point of view is a useful simplification.

From a practical point of view, functionality seems well understood for a given system, and everything else about it, worth changing, is really a matter of improving its other attributes. Functionality only *seems* like a useful topic, if you define it as being the '*design* you do', to achieve qualities and costs. I find this point of view very limiting, and only held by people who lack ability to articulate quality quantitatively, and who lack ability to design towards multiple objectives simultaneously.

For example:

| Functional Requirements:<br><br>Mobile Telephone Base Station: <whatever is fundamentally in all such base stations>. |
|---|

## 2: Design: The Evo 'Means' to the Target 'ends'.

'Design' I define as the means by which a function gets its quality and cost attribute levels. Design ideas are the fuel of the evolutionary engine. Design ideas drive the Evo project in the direction of target requirements, within defined constraints.

### The Multiple Quality and Cost Attributes of Design Ideas

All design ideas contain multiple quality impacts, and multiple cost impacts. No design idea is so simple as having a single-dimensional impact. No more than a chess move impacts one things.

The impacts of most design ideas are nowhere scientifically charted, nor are they put into engineering tables to be looked up. The result is that we have little knowledge of how the ideas will work in practice, until we can try them out. The problem is not only that we do not know some set of properties that the design ideas possess. The problem is more that we are going to add these ingredients to a stew of many other ingredients, and it is the *combination* of the ingredients which determine the taste and toxicity.

### The Tricky Business of Knowing the Effects of a Design Idea

Worse, the effects of a design idea are also *dependent* on changes made to the system *after* they are first measured in place in the system. In addition, the effects of a design idea are also dependent on 'design ideas which are undetermined, and unknown', at the moment we select that particular idea! This 'scary' scenario has no calculation, or guaranteed known answer. But that is *not* exactly a *new* scientific or engineering or management, or culinary problem! The cook must taste the concoction, every time, and many times, in its preparation.

### Evo process is analytical and serves the true taste of the food.

Here is where 'evolutionary delivery' rides to the rescue. We can form a hypothesis that a certain design idea will do us 'a lot of good', with 'tolerable costs' and 'tolerable negative side effects'. We can either guess the results, based on previous experience (high risk), or use a formal 'impact estimation process' (next chapter) to keep score of the possibilities, in a more systematic analysis than mere 'belief' (safer!). Then, we can evolutionarily, cautiously, try out ideas one by one, keeping those that do us good, rejecting those that surprise us with negative effects.

> *"Because some design issues are cheaper to resolve through experimentation than through analysis, Evo can reduce costs by providing a structured, disciplined avenue for experimentation.." [MAY96]*

Better, we can *measure* the total resulting *reality* of the *cumulation* of *all* our designs, at any Evo step. The truth is what we measure, not what we hoped or expected. From this standpoint of reality we can see the remaining **gaps** to our desired target quality goals, and the remaining budgeted available resources which can be used to close the gaps.

> *"In contrast to conventional project management, the overall budget for an Evo project is taken as a given, and the Evo budget process is closely analogous to the scheduling process; it is 'build to cost' " [SPUCK93]*

And more good news, we can do this 'small step by small step', committing no more than maybe 2% of project budget, until we have some concrete evidence of success or failure. We can see the 'cause and effect' in clear relationships. So, if we are in trouble, we know *why*; and the error is reversible. Go back to the previous step, and try another hypothesis.

So, the entire process of design under an Evo process, is less high-risk guesswork than otherwise. The *need* for design is spelled out by *concrete gaps* from 'reality' to our 'targets'. The evolutionary capability of design confirms or denies our hypothesis, and if necessary gives us another shot at the target, with losses from our experiment as 'tolerable wounds', but never fatal.

### Reality is our New Approval Committee

And, positive news! Design ideas with theoretically high potential, but many unknowns and high risks, can tolerably be tried out. There is little need to get approval from conservative Masters of the past. If ideas fail, we are quickly rid of them, and suffer no disaster. But, if they succeed, then we are big

winners, in our design lottery, and we know we can use the design, and the know-how acquired, in future steps of our project, multiplying the winnings. Approval is given by the wisest Master of them all. Reality.

This amounts to delegation of authority, faster competitive decision-making, and a 'learning organization' all rolled into one package. And, having avoided the caution necessary in non Evo cultures, we have competitive know-how for other projects, which otherwise might have been lost, or never gained, by us, at least. Who needs 'research' when a *research possibility* is built into 'development'?

> *Many development teams lack a well-defined, efficient decision-making process. Often they make decisions implicitly within a limited context, risking the compromise of the broader project goals and slowing progress dramatically. Evolutionary Development forces many decisions to be made explicitly in an organized way, because feedback on the product is received regularly and schedules must be updated for each implementation cycle.*
> *The continual stream of information that the project team receives must be translated into three categories of decisions: changes to the product as it is currently implemented, changes to the plan that will further the product implementation, and changes to the development process used to develop the product. Fortunately, because of Evo's short cycle time, teams have many opportunities to assess the results of decisions and adjust accordingly.[COTTON96]*

### The Choice of Risk and Benefit Levels at Each Step

It is also worth noting that the designer, has a choice at each evolutionary step. They can choose high potential benefit and low cost over other designs. They can choose high risk designs, with high potential, early in the Evolutionary project process. They can, like the chess player, choose any legal move, and will seek some 'best move' in their eyes, to meet their goals. The point being that if you choose what appears to be very high benefit and low cost Evo step contents, and there is some degree of disappointment in the result, then the reduced results might still be of a credible, useful or even 'impressive' nature.

Every Evo step is declared to be an experiment, with no certain results promised in advance. Disappointing results can be declared a success in proving what the project should not invest more in. Encouraging results can be bragged about after they are a fact. There is no need to brag in advance!

### The New Guides for the Evo Designer.

Designers do not need the same degree of *design idea experience and knowledge* as before. They need to understand how to *exploit the evolutionary design process* for maximum benefit at minimum risk. They need to be taught to *know what they do not know*; what they *need to k*now; and how to *safely find out*.

The evolutionary designer is constantly focussed on the Everest of reaching the customer targets. They have the best Sherpa guide in the world, in the Evo process, which will safely get them to the top, or warn them in good time of the necessity of safe retreat.

Evolutionary design is done in the interactive rhythm of the Plan Do Study Act (PDSA) feedback learning cycle of Deming. Design is constantly being tested against the *reality* of the design idea itself, and the recipients appreciation of the resulting effects. Design is not 'approved' by 'sterile irresponsible premature unrepresentative, and unknowing-of-the-unknowable undocumented facts, office meeting committee' (whew!), but by the *reality of results*.

> *"a key benefit ... is its ability to progressively refine requirements and to respond easily to the refinements. Refinement is done on the basis of developmental test, training, and operational experience. Requirements feedback facilitates working in an environment of change." [SPUCK93]*

## *The Process of Design.*

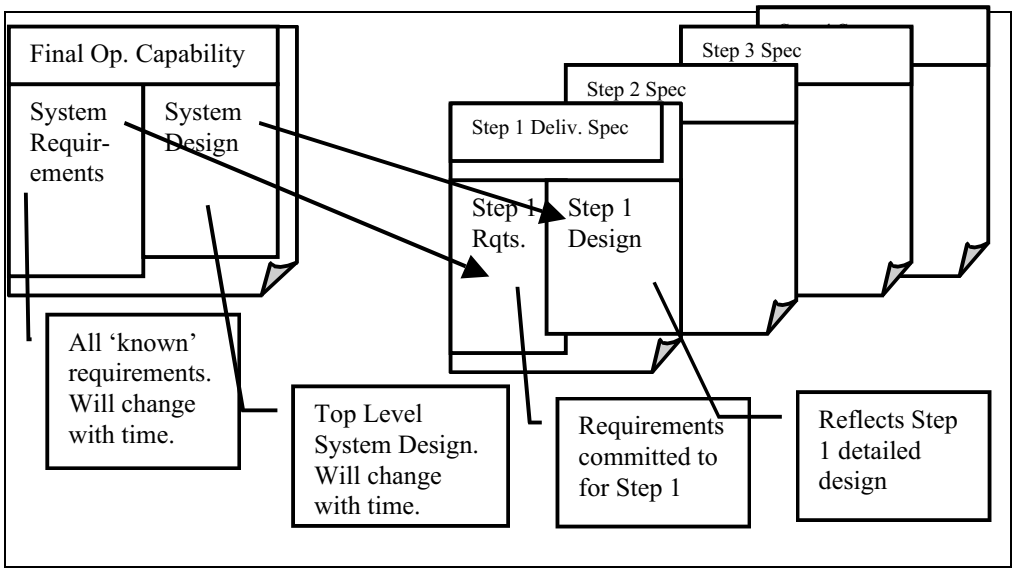
The process of finding a design idea starts with a clear picture of the total requirements to be filled. We should be looking for a design idea which will have maximum effect on fulfilling all requirements, within the constraints.

### The Dream of the Automatic Design Engine.

Where do we look for design ideas? In some fantasy world we would deliver the requirements to some web search engine and a list of ideas which at least partly seemed to fill the bill would pop up, after a search of the Web. Years ago (1979) Lech Krzanik and I made such a search engine work ('Aspect Engine' on an Apple II file!), but we discovered we were severely limited by the lack of data about the qualities and costs of any and all potential design ideas.

### Where do we get ideas from?

Design ideas come from a variety of sources. Our experience, our studies, colleagues ideas, product offerings, finally perhaps inspired by the surprising results of an evolutionary steps delivery..
So, we do get ideas, somehow, even if the systematization and depth of knowledge leaves much to be desired.




Specification Evolution. [based on SPUCK93]

> *"The [step] delivery specification is required to be much more complete and specific than the Final Operational Capability (FOC) specification, although good [Evo] practice suggests that any requirements or design information that is known at any point in time be captured in the FOC specification regardless of the delivery in which it will be implemented. Also the FOC specification is kept current with the [step] delivery specification so that the FOC specification evolves with time to become the complete 'as built' specification." [SPUCK93]*

### How do we know if ideas are any good?

There are two ways to figure out if a design idea is useful, *estimations*, and *experience*. Estimations, save you the trouble of trying out unpromising ideas. But they may require data about quality and cost attributes which you really do *not* have. Sometimes there is only the option of 'trying out' an idea, however bad or unknown, and seeing what it does for your requirements.

The advantage of evolutionary project management is that we don't *have* to know for certain, in advance, what all the design idea attributes are. We can, with low risk, inject them experimentally into a step, when we believe they are the best option we have. Should surprisingly-negative attributes surface, we are always prepared to learn the lesson, remove the offending step, and try alternatives.

We can reduce the risk of things going wrong by seeking as much knowledge about the idea *before* trying it out. The 'impact estimation method', next chapter, is one method for analyzing design ideas before we experimentally try them on an evolutionary step.

### Stages of Evo design.

The process of design, in evolutionary projects, has two essential stages:

1. Finding at least a rough architecture (set of design ideas) which probably will be able to satisfy our requirements (also called 'design objectives').

2. Extracting, from this architecture, specific sub-ideas which are suitable for injection into an evolutionary delivery step. Trying them out. Noting the results. Noting the remaining gaps to fulfilling requirements. Then continuing the search for other ideas to fill the gap in the next steps.

### Evo seeks to meet requirements by any means

It is important to note that since the emphasis is always on *meeting defined requirements*, the design is always open for *any class* of technical or organization design ideas which do the required job. This is *systems* engineering, and is not limited, even by the major discipline which the project team assumes they are using to build a system.

For example a *software* project, will also have to be concerned by agreements, contracts, marketing, training, help desks, service, web sites for updates, hardware compatibility of potential recipients, motivation to use their steps, and an endless stream of practical considerations which might prove to be necessary to reach the *project* requirements, as opposed to narrower *product* requirements.

An Evo product is constantly being thrown out into the cruel real world and we have to deal with these annoying details which make product work in the customer environment. Naturally, this is a healthy dose of reality for the young inexperienced or narrowly focussed engineer. And, it is a competitive advantage for the company that exposes their people and products to the customers, so that the engineers have to deal with the real source of their income.

> *"In parallel with the development activities of the team, selected users or customers of the system are working with and providing feedback on the release from the previous cycle. This feedback is used to adjust the plan for the following cycles." [COTTON96]*

### When is the design process done?

The process of design is continuous and stops only when no more requirements remain to be satisfied, or at least until we run out of resources to satisfy our desires. It certainly continues long after initial evolutionary delivery steps are in the hands of initial field trial customers. It most certainly is *not* one big architecture process before the building start is approved.

> *"a key benefit ... is its ability to progressively refine requirements and to respond easily to the refinements. Refinement is done on the basis of developmental test, training, and operational experience. Requirements feedback facilitates working in an environment of change." [SPUCK93]*

### Open Architecture

This does not mean that we dive in with no overall idea of some of the architecture fundamentals. But, since the detailed solution ideas are unknown and uncertain, not least because the real requirements of the market and customer can change due to external forces or internal insights, the most important architectural ideas are simple ones which make it easy to change, alter, extend, improve, port. We call these flexible design ideas 'open architecture' and will deal with them later in this chapter.

## *The Design Specification.*

Basic design specification is simple: give the idea a name tag, describe the idea.

Example:

DOORS: Use the 'DOORS' Requirements Specification tool made by QSS.

The entire 'Planguage' specification language can be used to enhance the detail and intelligibility of the specification. See Glossary for more information, but hopefully the examples give you the idea.

For example

- ✓ Hierarchical tags: Productivity. Tools.DOORS
- ✓ Intended Impact: DOORS IMPACTS Lead Time. DOORS → {Lead Time, Defect Level}.
- ✓ Qualifiers: DOORS [USA, Europe], SLATE [Out the year, UK Civil Service].
- ✓ Comments: DOORS: Use 'DOORS', "until better alternatives established".

- ✓ Sources: Motive: Pay project team 10% completion bonus ← Project Manager.
- ✓ Defined Term: Dual: Use Distinct software.
  Distinct: DEFINED: same function but different faults.
- ✓ Fuzzy Twin Antenna: Use <multiple> antennas in <noisy environments>.

## *Open Architecture.*

As mentioned above, the most fundamental design ideas in an evolutionary project will be ideas which make it relatively easy to later introduce totally new and unforeseen ideas, without disturbing unexpected cost and system quality degradation.

### Open Architecture is both 'soft' and 'hard' technology.

Open architecture consists of absolutely anything that make future changes easy to introduce, both during the project development period and afterwards when the system is enhanced, ported, extended or changed in any way for any reason. In particular open architecture is not merely 'technical' design {interfaces, structures, languages} , but can include any contracting, agreement, measurement, motivation, insurance, organizational aspect which, in fact, has the effect of making the Evo step changes less costly, less time or effort consuming, and less likely to disturb other valued qualities of the system such as performance, battery life, range, availability, for example.

### Open Architecture is not additional stuff necessarily, but a choice of paths.

Open architecture need not cost anything extra to have or to install. It is often merely the choice of the right standard, the right interface, the right insurance, the right technology. It does require conscious planning at the 'architecture' level. Not all open architecture devices will be understood initially. But, hopefully the practical experience of change and system extension at each new Evo step, will cause your project to realize that there are decisions they could make which will make things easier for themselves in future steps. It is sufficient that the open architecture design components are installed early, so that maximum benefit is gained from them on the project. But even if they are discovered late in the initial project, hopefully they will turn out to be value as the system or product changes after the project transitions to the operational field stages of life.

### Open Architecture is an Engineering thing, not an intuitive thing.

Choice of open architecture design ideas should not be a dogmatic or intuitive thing. It should largely be an architectural decision-making phase in response to specific engineering requirements for adaptability, extendibility, portability, serviceability, connectivity, maintainability and other qualities of system change ease. All of these can be characterized quantitatively in terms of the time or cost of carrying out such changes. It is in response to specific requirements that we should engineer the system, from the beginning of our evolutionary project, to have such properties. Evolutionary projects are a great opportunity to test and measure the ability of a system to withstand constant frequent unpredicted and unplanned change, just like it is going to have to do after our development project is 'over'.

Example of a quality requirement demanding open architecture:

Adaptability:
Gist: the ability of our system to easily tolerate unexpected changes. The set of all other ease-of-change qualities below { Extendibility, Portability, Serviceability }.

Extendibility:
Scale: the engineering effort needed to add [defined capacity] to the product.
Plan [memory by factor 10] less than 10% of cost of memory itself.

Portability:
Scale: the engineering effort needed to move [defined system elements] to [defined target environments] using [defined tools or skilled people or processes].

Plan [software logic and data, East Asian Markets, Average Programmers] 1 hour per 100 lines of code.

Serviceability:
Scale: The ease of giving [defined service types] in [defined service locations] by [defined levels of service people].
Plan [Shop Counter, Major Chains, Certified Trained Specialists] 90% Service Cases within 30 minutes "in shop wait".

Examples of Open Architecture:

JAVA: Use Java programming Language → Portability.

IEEE675: Use the IEEE 675 Interface Specification → Extendibility.

Self Test: Build all components hardware and software with thorough self test and defect reporting capability in fully automatic mode →Testability & Adaptability.

Accessories: Use the Corporate Product Line Interface for all accessories, or at least include necessary plugs, cables and software with each product to enable interface to it →Adaptability.

Display: All design of displays will assume that future displays can be of any size and dimension both smaller and larger than initial releases →Adaptability.

# 3: Impact Tables: The Evo Accounting and Planning Mechanism

## *Impact Tables as an Evo Step Selection Mechanism.*

Impact Estimation Tables can be used for any purpose related to understanding the relationship between means and ends, between solutions and problems, between designs and goals.

### Evo step analysis using Impact Tables.

They were initially evolved by me to help analyze any two sets of design ideas against multiple quality and cost objectives. But, it turns out that they are well suited for modeling the evolutionary process. They can help give the following questions some analytical answers.

- ✓ How much will a particular design idea impact the step goals?
- ✓ Which ideas should be *selected* for this *particular* step?
- ✓ What is the expected outcome of implementing this step in terms of quality improvements and costs?
- ✓ What is the *uncertainty* of the step implementation, and what is worst case results?
- ✓ What is the expected impact of a *planned series* of steps on our objectives?
- ✓ What was the cumulative impact of the *past series* of steps on our objectives
- ✓ What was the impact of the *last* step on our objectives?
- ✓ What were the things we understood least on the past step?

### Straightforward Evo step comparison, disregarding risk.

For example, here are two Evo step candidates rated on an Impact Table (100% =Plan level)

| | Step Candidate **A**: {Design-X, Function-Y} | Step Candidate **B:** {Design Z, Design F} |
|---|---|---|
| Reliability 99%-99.9% | 50% | 100% |
| Performance 11sec.-1 sec. | 80% | 30% |
| Usability 30 min.-30 sec. | -10% | 20% |
| Capital Cost 1 mill. | 20% | 5% |
| Engineering Hours 10,000 | 2% | 10% |
| *Performance/Capital Cost Ratio* | 80/20= *4.0* | 30/5= *6.0* |
| **Quality/Cost Ratio** | 120/22=**5.46** | 150/15=**10.00** |

At first the pattern of impacts is confusing, and it may be difficult to pick a best 'next step' alternative. But if we assume that 'meeting the planned levels of goals' is equally important for success; that *is* the definition of a planned level, the *success* level, then we can calculate a general figure of merit, the 'Quality/Cost Ratio'. This Q/C ratio is a useful general indicator of the efficiency (benefits to cost ratio) of a step.

However, it could happen that for some political or practical reason you do not want to decide on the steps based on the general overall impacts, but on some narrower criteria such as Performance and capital cost. As shown that ratio can be calculated and used to select a step.

### Evaluating Evo steps with regard to real world spread of experience.

If the uncertainty, or risk of deviation from main estimate, were to be evaluated then, this example shows us the possibilities:

| | Step Candidate **A**: {Design-X, Function-Y} | Step Candidate **B:** {Design Z, Design F} |
|---|---|---|
| Reliability 99%-99.9% | 50%±50% | 100%±20% |
| Performance 11sec.-1 sec. | 80%±40% | 30%±50% |
| Usability 30 min.-30 sec. | -10%±20% | 20%±15% |
| Capital Cost 1 mill. | 20%±1% | 5%±2% |
| Engineering Hours 10,000 | 2%±1% | 10%±2.5% |
| *Worst Case B/C ratio* | (0+40-10)/(21+3) *=1.25* | (80-20+5)/(7+12.5) *=3.33* |
| **Best Case B/C ratio** | (100+120+10)/(19+1) = **11.5** | (120+80+35)/(3+7.5)= **22.38** |

We can see that there is an order of magnitude difference at the extremes of optimism and pessimism. We can also see, if these are our only choices, that 'B' is our best bet, even considering risk.

The risk here is usually the spread of available experience data. '-10±20' above, is taken to mean that the available data stretches from minus 30 to plus 10, and that minus 10 is the most frequent impact.

There is no pretence that these calculations give desired accuracy. For example we have not yet considered the credibility of the estimates. But it does give us a tool for somehow summarizing the effects of multiple benefits and costs. If we want accurate data, then thankfully it is only an Evo step away from being a reality. So, if it takes a week of project time to *do* that step, how long time would you find it worth spending gathering more-credible data? Not two weeks we assume.

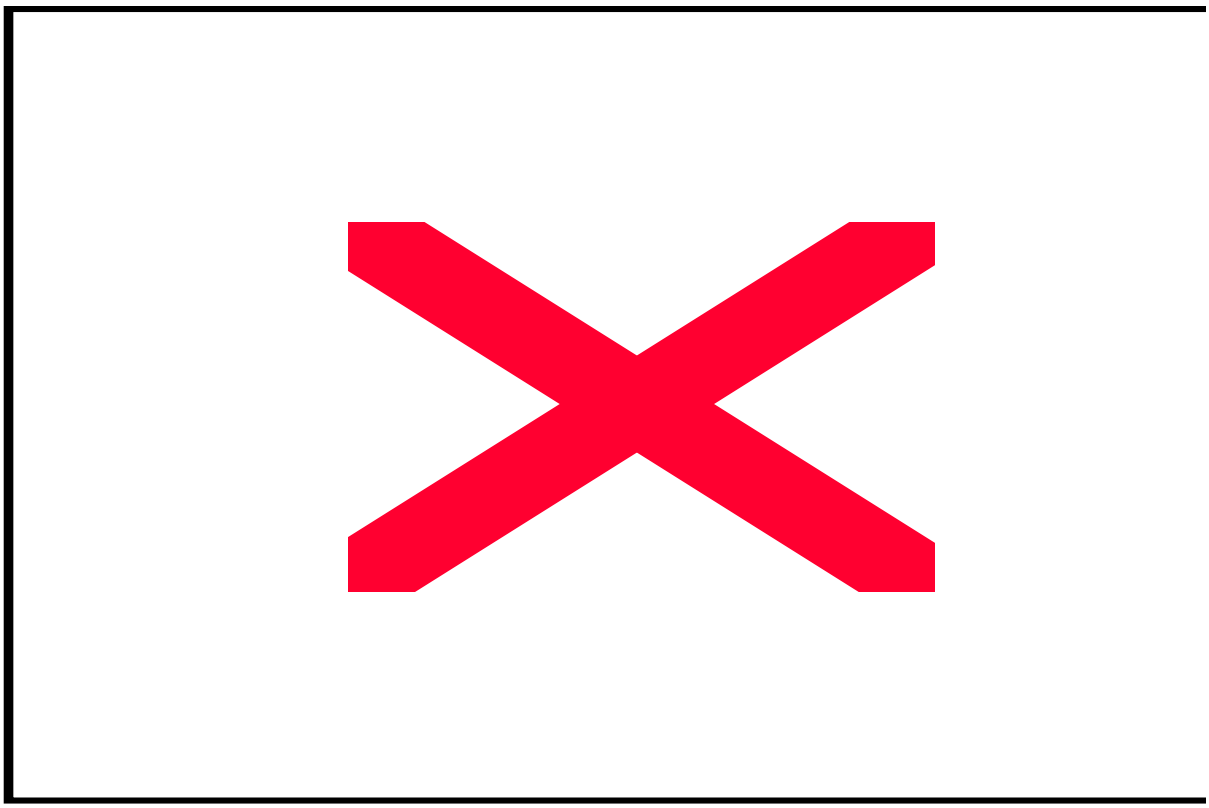
### Evaluating Evo steps with regard to credibility of estimation data.

Having raised the subject of the credibility of the estimates, treated earlier, let us try an example. Let us say that Step A had credibility of 0.8 (1.0 is perfect) and Step B a credibility of 0.2 (0.0 is a random number estimate).

| | Step Candidate **A**: {Design-X, Function-Y} | Step Candidate **B:** {Design Z, Design F} |
|---|---|---|
| Reliability 99%-99.9% | 50%±50% | 100%±20% |
| Performance 11sec.-1 sec. | 80%±40% | 30%±50% |
| Usability 30 min.-30 sec. | -10%±20% | 20%±15% |
| Capital Cost 1 mill. | 20%±1% | 5%±2% |
| Engineering Hours 10,000 | 2%±1% | 10%±2.5% |
| *Worst Case B/C ratio* | (0+40-10)/(21+3) *=1.25* | (80-20+5)/(7+12.5) *=3.33* |
| **'Worst Worst' case considering estimate credibility factor** | 0.8 x 1.25= ***1.00*** | 0.2 x 3.33= ***0.67*** |

Now, Step A looks like the best bet for the conservative project manager.

My son and colleague, Kai Thomas Gilb, has long ago built a spreadsheet tool for making all these calculations, and displaying them as graphs. Make one yourself.



This is an artificial example from Kai's tool. Step CC looks bad initially, but not so bad when the "Worst Worst' case is considered.

## *Impact Tables as an Evo Step Construction Mechanism.*

Evo steps are constructed according to a policy of being small enough that you can afford to be wrong, and can afford to lose the time and money spent, if the step fails.
If the Project Policy dictates maximum 2% of budget, and maximum one week to delivery of a cycle, then we can use Impact tables to assemble design ideas into a suitably large step.

**Building Up an Evo step to a 'biggest allowable bet' threshold.**

| | Design:<br>Dual [USA] | Design<br>Dual [Europe] | Next Step<br>**Maximum** at the 2% level | Next Step *Totals*<br>Before looking at possible additional Design ideas |
|---|---|---|---|---|
| Capital Investment | 150,000±100,000 | 10,000 ±5,000 | **200,000** | *180,000* |
| Project Calendar Time | 0.1 to 0.5 weeks | 0.05 week | **1 week** | *0.9 week* |
| Engineering Hours | 20 ±10 hours | 10±5 hours | **150** | *130 hours* |

You can see, from this constructed example, that implementing the design idea 'Dual' in 'Europe' will work if added to the current step 'bucket'. But, that the same design idea in 'USA' would be consistently 'over the top' of the allowable risk. That one would be fine to think about for the next step after this.

| | **Development Team** | **Users** |
|---|---|---|
| Monday | ✓ System Test and Release Version N<br>✓ Decide What to Do for Version N+1 | |

| | | |
|---|---|---|
| | ✓ Design Version N+1 | |
| Tuesday | ✓ Develop Code | ✓ Use Version N and Give Feedback |
| Wednesday | ✓ Develop Code<br>✓ Meet with users to Discuss Action Taken Regarding Feedback From Version N–1 | ✓ Meet with developers to Discuss Action Taken Regarding Feedback From Version N–1 |
| Thursday | ✓ Complete Code | |
| Friday | ✓ Test and Build Version N+1<br>✓ Analyze Feedback From Version N and Decide What to Do Next | |

Fig. An example of a typical one-week Evo cycle at the HP Manufacturing Test Division during a project. [MAY96]

**Reducing overweight design ideas to fit the policy step size.**

Similarly, if a projected design alone exceeds the 'betting limits' then it must be decomposed so that it does not blow the budgets. For example assume the basic design idea 'Dual' (above) was 10% of all cost ideas of the project. Then we need to split it up into partial implementations, such as the ones above (Dual [Europe] and Dual [USA]) and we can use the Impact table to keep accounts and make decisions as to good decompositions, and good Evo step sets of design ideas.

*"[Evo] adapts easily to changing programmatic environments of which ... budget changes ...are typical. .... Each succeeding delivery can be scaled to funds available…… [Evo] has an excellent immune system to normal funding variations and finds little challenge in providing meaningful increments of capability under most real-life funding scenarios." [SPUCK93, 7.2]*

## *Impact Tables as an Evo planning mechanism.*

Assuming we have settled on suitable sized Evo steps, we can sequence them, and we can make longer range provisional plans. *All* Evo plans are *provisional* on the outcome of steps and new insights from any relevant source.

The Impact table can help us to do intelligent sequencing. The general sequencing paradigm is:

- ✓ 'Do the steps which can contribute the most towards meeting quality requirements in relation to cost'

But, many other options are possible, including

- ✓ Do the step that the *customer* selects, from the list of possible steps.

*"Microsoft's general philosophy has been to ..... focus on evolving features and whole products incrementally, with direct input from customers during the development process." MS, 13*

- ✓ Do a step which emphasizes progress towards a *selected few* quality goals.
- ✓ Do a step which makes *some* visible progress at the least possible cost.
- ✓ Do a step which gives best insight into areas of greatest risk of failure.
- ✓ Do a step which is necessary for co-ordination with other teams or dependencies

*"Although it is difficult and time-consuming, the work breakdown structure and dependency information must be done and done correctly." [MAY96]*

We can use estimated values on Impact Tables to establish any such interesting set of selection criteria for a step. We can compose a step from single design ideas using such criteria. We can even factor in the risk and credibility factors as illustrated above.

*"Some of the criteria commonly used in setting priorities during this initial planning activity are ...:*
***Features with greatest risk.***
*The most common criterion used for prioritizing the development phase implementation cycles is risk.*
*When adopting [new] technology, many teams are concerned that the system performance will not be adequate.*
*Ease-of-use is another common risk for a project.*
*The **[step content]** that will provide the best insight into areas of greatest risk should be scheduled for implementation as early as possible. " [COTTON96]*

## *Impact Tables as A Feedback and adjustment mechanism.*

Impact tables can be used to plan and budget future Evo steps. But they can also capture corresponding measures of actual progress and costs for each step and for the cumulation of each step. The analysis of the differences between you plan and the reality gives a basis for action (yes, Plan Do Study Act) to either keep you on target for critical requirements, or to consider intelligent adjustment of requirements to realistic or possible levels.

| | Step #1 **A**: {Design-X, Function-Y} | Actual | Difference. - is bad + is good | Total | Step #2 **B:** {Design Z, Design F} | Actual | Difference | Total | Step #3 Next step plan |
|---|---|---|---|---|---|---|---|---|---|
| Reliability 99%-99.9% | 50% ±50% | 40% | -10% | 40% | 100% ±20% | 80% | -20% | 120% | 0% |
| Performance 11sec.-1 sec. | 80% ±40% | 40% | -40 | 40 | 30% ±50% | 30% | 0 | 70% | 30% |
| Usability 30 min.-30 sec. | 10% ±20% | 12% | +2% | 12% | 20% ±15% | 5% | -15% | 17% | 83% |
| *Capital Cost 1 mill.* | 20% ±1% | 10% | +10% | 10% | 5% ±2% | 10% | -5% | 20% | 5% |
| *Engineering Hours 10,000* | 2% ±1% | 4% | -2% | 4% | 10% ±2.5% | 3% | +7% | 7% | 5% |
| *Calendar Time* | 1 week | 2 weeks | -1week | 2 weeks | 1 week | 0.5 weeks | +0.5 wk | 2.5 weeks | 1 week |

This Impact table for project management gives the project manager constant realistic feedback based on actual measures of progress towards goals, alongside the cost information.

# 4: Evo Planning: How to specify an Evo Project plan.

## *What all Evo planning specification methods have in common.*

### Basic Features

All Evo plan specifications need:

- ✓ A definition of the 'next step' to be delivered.
- ✓ An overview of all other steps which are contemplated.
- ✓ Some notion of benefit and cost attached to the steps.

### Advanced Optional Features

- ✓ Feedback about how well steps have measured up to expectations
- ✓ Multiple quantified quality and cost estimation per step
- ✓ Graphic progress reporting and planning
- ✓ Forward estimation of consequences of experiences thus far
- ✓ Groupware sharing and commenting: including Web access
- ✓ Quality control certification through Inspection for exit ('Max. 0.2 Majors/Page remain')

Here are some practical details.

## *A simple list.*

An Evo plan can be specified on a page as a simple list of tasks. Each task will give some recipient visible result. Many tasks are logical pre-requisites for the others. All tasks after the next committed task are subject to re-planning. New ideas can be inserted at any time. The envisaged sequence can be altered for any good reason.

This method is quite suitable for a 'project' done by a single individual, of a few weeks duration. I used this method for building data processing systems up from 1960, before I was aware that there was an 'Evo' method.

Simple Evo Plan.

1. Build customer files and produce published customer lists, replacing old style ones.
2. Build product files and pricing information, replace old price catalogues.
3. Take orders and produce a warehouse article picking list.
4. Produce invoices for simple frequent cases.
5. Produce more complex invoices with special discounts.
6. Produce invoice reminders.
7. Accept payment information and produce differential invoices.
8. Do accounting for all invoices
9. All other steps symbolized as included in this rough last one.

In a sense this is not much more than a 'to do' list. It builds only functionality, not quality improvement. The steps are logical from a business point of view. But, it is evolutionary in that each step delivers something useful to an end user. And, the end user can give feedback and make new choices.

## *A word processor outliner.*

For larger projects, a hierarchical word processor outliner is a natural list maker. I normally use one when as I consultant I am drafting industrial scale plane, several years ahead.

Project Evo Plan

Rest of This Year
- November
  - Week 45
    - Deliver Change of Priority for Top Brass
  - Week 46
    - Increase Performance by at least 10%
- December
  - All remaining performance improvements

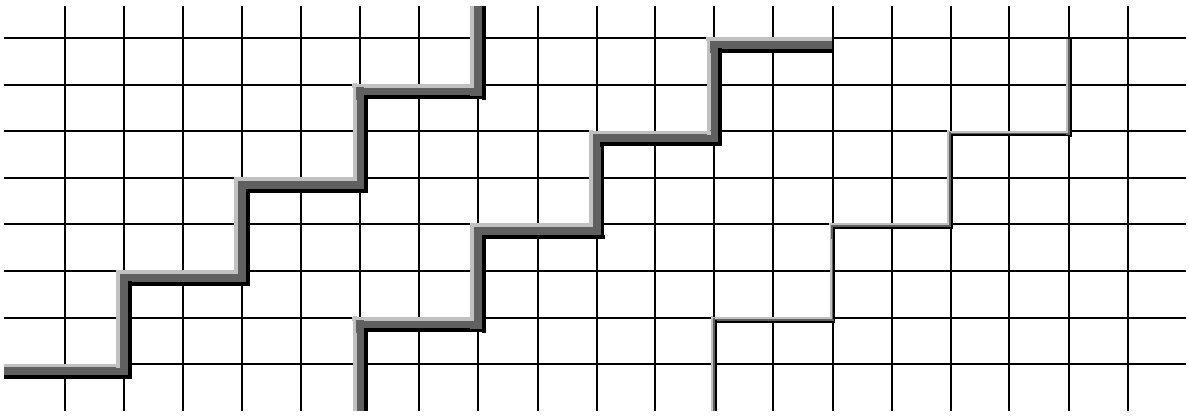
Next Year
- First Half
  - Focus on Quality Improvement targets
- Second Half
  - Focus on Customer Service Improvement

All remaining Years
- Content and sequence dependent on what has been learned and what has happened until this time.

This is just a skeleton. It shows that we get very concrete about the nearest times, and sketch more roughly as time goes into the future. The hierarchy and a word processor or other outline mode aid is a natural way to keep track of a plan which is very detailed at the front end, and sketched at the back end. You can of course hide and show detail as appropriate.

### *The Step Diagram.*

When working in meetings, with flip charts or boards, or even when thinking by myself, I have consistently found a 'step diagram' a convenient way to express an Evo plan.



This is the basic pattern I use. Two or three sets of steps, the left-most for the long range and the right-most for the shorter range. On larger projects I have needed to go 7 levels down in depth (down to step 1.1.1.1.1.1.1, that very first practical step). Step zero I reserve symbolically for pure planning steps which do not intend to produce a recipient result, except plans.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Mega-** | **steps** | | | 2003 Step 6 | ? | **Mega -** | **Steps** | Rest of year 1.9 | ? rest tune | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 2002 Step 5 | ? | | | May 1.5 | Bottle necks | | |
| | | | 2001 Step 4 | Speed | | | April 1.4 | Priority | | 1.1.4 | Wk. 4 |
| | | 2000 Step 3. | Capa-city | | | March 1.3 | Nodes | | 1.1.3 | Wk. 3 | Clea n out |
| | 1999 Step 2. | Usab-ility | | | Feb. 1.2 | Com munic | | 1.1.2 | Wk. 2 | Auto tune | ← |
| 1998 1. | **Perfor - mance** | → | → | Jan. 1.1 | **Tune Data base** | → | → | Wk. 1 1.1.1 | **Reorg-anize** | ← | ***Deli-very steps*** |

Using a word processor table, here is a sketch of a three level Evo plan. The years are sketched first, then months, finally weeks or whatever is the Evo cycle length. I started using this format 1970-1980 as my main format for Evo thinking.

'**Mega-steps'** are the two left side step sets in the example above; mega-steps are a super-set of delivery steps. **Delivery steps** are a package which intend to deliver some impact and results to a recipient of some kind.

I began to feel that this type of chart, which I have used for sketching large projects with groups of people was obsolete, when impact table specification was developed, but I am surprised to find that it is the way I draft or present an Evo plan to a group, especially when we want to solicit ideas. It is not the best way to keep track of the plan during the project. But, that is why we have other formats.

## *Ladder Format*

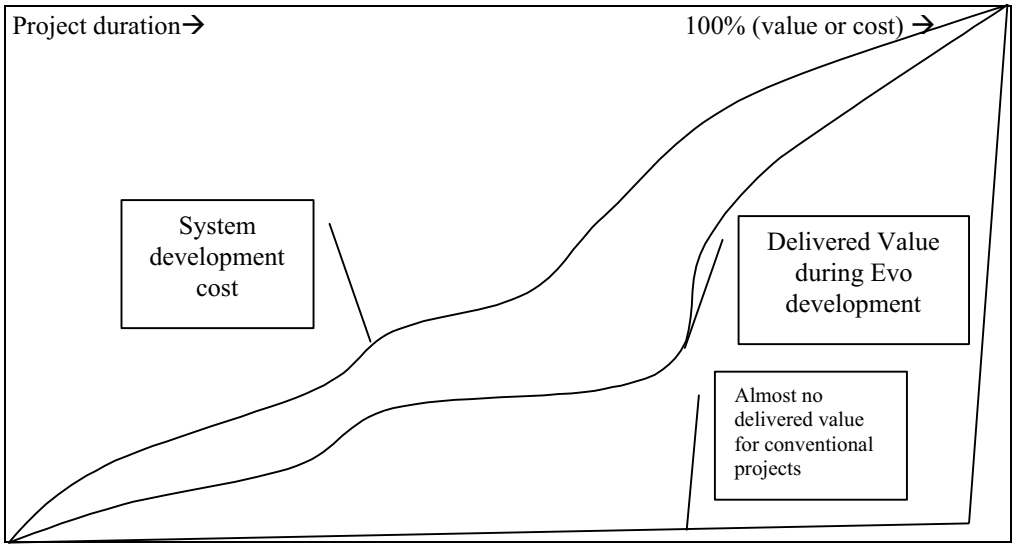
The ladder format was developed by me about 1985 in UK. It is not quite as convenient to sketch freehand on a chart or board, but it fits into computer screens better!

| *Step Name tag* | *Description (Design Tag)* | *Other related information* |
|---|---|---|
| **User Enhance** | {Interact, Intelligent} | 9/1 (Value/Cost ratio) |
| **Port** | {Convert Code [UNIX], Convert Data [Database, Tables]} | 8/3 |
| **Miniaturize** | {Mini Phone, Field Trial} | *Depends on Port 7/3* |
| **Retail** | {Market, Distribute, Discount} | 5/1 |
| **Europe** | {European Community, EEconC} | 2/1 Do by Country Size, Small up |

This format can use the formally assigned names of steps and of designs. It is suitable for keying in a word processor. It 'organizes' the step information. It can be done hierarchically. The 'descriptions' are the set of design ideas which will be implemented at this step. These are described in detail elsewhere. The design idea tags are a cross reference to the detailed descriptions.

The value to cost ratio is a simple subjective opinion device using a scale of 0 (low) to 9 (high) for value and costs of the step. I have used it with groups to get consensus about the value and cost of a step. The value to cost ratio is a rough subjective, often negotiated, determination of the step impact on multiple qualities and multiple costs which we are trying to manage. It does not replace thorough determination of those many attributes later, but it simplifies a group meeting and communication process at early stages of an Evo process.

*"For Conventional Development Methods (CDM), money is invested steadily in order to derive value only at the final delivery, Final Operating Capability(FOC). That is, under CDM there is no value to users until FOC. In contrast, [Evo] makes the same investment and derives value to users incrementally, shortly after the investment. It is this characteristic of CDM that has prompted sponsors to impose performance measurement systems (PMSs) that aggregate artfully composed measures of 'earned value' for accomplishment of the various processes of the CDM life cycle. [Evo] substitutes operational capability or delivered value to earned value." [SPUCK93]*




Evo projects deliver value *during* the project. [after SPUCK93]

## *Impact Table Planning*

As shown in the previous chapter, impact tables can be used to specify evolutionary steps, and the mega steps which summarize a set of delivery steps.

| | Host System base | Mega-step **Next Year** | Mega-step **Jan Next** | Delivery step **Convert** | **Convert** Impact % | **Jan Next** Impact % | | | |
|---|---|---|---|---|---|---|---|---|---|
| Q1 99% to 99.9% | 99.00% | 99.5% | 99.05% | 99.05% | 6% | 6% | | | |
| Q2 10 min-1 min. | 10 min. | 5 min. | 9 min. | 10 min. | 0% | 11% | | | |
| Ca 0-$2mill. | 0 | $1,000,000 | $100,000 | $20,000 | 1% | 10% | | | |
| Cb 0-20,000 Eng., hours | 0 | 10,000 hours | 5,000 hours | 1,000 hours | 5% | 25% | | | |
| Q/C | | | | | 6/6=1 | 17/35=0.49 | | | |

In this example we have shown two levels of mega-steps (Next Year, Jan Next), and one delivery step (Convert). We have shown the real values and the percent-of-target values.

The real values communicate directly, in 'accomplishment levels'. The 'real values' correspond to the 'defined scales of measure, in target requirements planned levels'.

The '% of target' values 'normalize' all these real numbers, so that we can more easily see 'how well we plan to do' at this step, in relation to the target requirements. We can also perform calculations on sets of estimates from different scales of measure, using the % of target 'common currency'. One was discussed in the previous chapter, 'Benefit/Cost'.

*"For Conventional Development Methods (CDM), money is invested steadily in order to derive value only at the final delivery, Final Operating Capability(FOC). That is, under CDM there is no value to users until FOC. In contrast, [Evo] makes the same investment and derives value to users incrementally, shortly after the investment. It is this characteristic of CDM that has prompted sponsors to impose performance measurement systems (PMSs) that aggregate artfully composed measures of 'earned value' for accomplishment of the various processes of the CDM life cycle. [Evo] substitutes operational capability or delivered value to earned value." [SPUCK93]*

**Horizontal summing of percentage impact to evaluate next step total impact.**

Another useful calculation with impact % is horizontal summing of impacts to get a very rough idea of the impact of a concurrent set of design ideas (example of A,B and C in example below).

This is admittedly very rough, but it can be used to get some estimate of expectation for a set of design ideas which are being considered for a single step.

We call it 'order-of-magnitude guesstimation'. It makes us aware of extremes, like far too little impact, or overwhelmingly much.

Rather than attempt to get data (almost impossible for the most part) to do a more reliable 'expected total impact' of a set of design idea, it is easier, when doing Evo, to simply put the design ideas into a step, do the step, and measure the result. This is more accurate, and faster. Early steps can be used to measure the expected impact of a set of technologies when they are later spread to other customers or recipient areas.

The 'actual impact observed' is immediately put into the 'project management impact table'. The observed impact used as the basis for any extrapolation of the use of this set of designs. The 'observed impacts' can also be used as part of the platform of 'known impacts thus far'.

Evo means you don't have to deal with a *large* number of uncertainties. You can focus on *one* set of uncertainties, your *next* step. This is much like the car driver who can focus on the immediate road ahead, in spite of the that that every road further on contains risks and uncertainties.

*For example:*

| *Design idea →* | *A* | *B* | *C* | *Total* |
|---|---|---|---|---|
| *Quality-X 1 min. – 10 seconds.* | *0%* | *100%* | *-20%* | *80%* |

**The 100% concept, the target.**

The target requirement we are using (for the 100% concept) is normally 'summarized' ("1 min. – 10 seconds" in example above, "10 seconds"=100%) under the quality or cost tag ("Quality-X" above) in the left most column. The sure interpretation of this number would normally require you to look at the detailed requirements specification, especially the scale, the qualifiers, and the type of target (Must, Plan)

## *The 'single step extraction' and 'minimal planning' option*

There is no strict necessity, every project, to plan ahead in terms of step, and step sequences. There is an argument that it is likely to be wasted in the face of the gradually unfurling truth of the project feedback at each step. In fact, the forward planning of Evo steps and Mega steps is rarely very detailed, and serves, more than anything, to give a 'comforting feeling' of overview and completeness.

So, one simplification of Evo planning is to be like the driver who knows his ultimate destination, and rough direction, but who focuses one street at a time, on the task of getting there.

We assume the primary driver is in place, the long term requirements; even if it is a single page quantifying the top ten critical quality improvements for a system, organization or product.

The next question, one that could be answered in a sentence, or no more than a page, of planning for each Evo step is: 'what shall we do for the next step?

The general answer is 'whatever best serves your recipients interest'.

This can be a simple as extracting a *set* of possibilities, and giving the recipient a *choice*.

In lieu of a reasonable direct 'recipient person' to ask, they could try to find some step that gives reasonably good progress towards the longer term requirements. Maybe the recipient has merely given a direction of which objective to work on for the time being. "Increase service response above all". So the project management will concentrate on designs and steps which will estimably give the most progress in that direction.

As long as frequent (like weekly) Evo delivery steps are making on-target visible progress towards recipient objectives, the lack of detailed planning will be forgiven.

*"Some [Evo] projects have very successfully implemented the notions with only a Final Operating Capability specification and a tabular appendix to the specification, listing which specification provisions are to be (or were) implemented at each delivery" [SPUCK93] JPL, CA, USA*

The main reason organizations develop 'standards for detailed planning' is that they have 'big bang projects' which promise to deliver something after a long while, and which invariably disappoint. Most planning systems are a reaction to the traditional lack of requirements clarity, design clarity, and measurable progress. Evo assumes a better climate. It may take traditional companies a while to realize that they do not need the project methods they have built up in 'self defense' for so many years.

I should mention as a matter of experience that we regularly used this variant of Evo for a wide variety of projects at a well known aircraft factory.

There is a famous chess analogy, Jose Raul Capablanca Y Granperra (1888-1942) , 1921-27 World Champion, a Cuban Grand Master pointed out that thinking about concrete moves far ahead was largely a waste of energy. He recommended focus on the 'next move', making it as powerful as possible against a Grand Master, no matter what the answering move was.

I think there is some wisdom in that, for many other projects.

So, the specification for a next step, may be as simple as

"Do Step 23" where Step 23 is defined as:

Step23:
{Priority [NL, GB, USA], Service [Product XX] }

where 'Priority' and 'Service' are defined design ideas. Example of definition below.

### *Ways to specify Parts of a defined design idea*

As illustrated above, there are ways in 'Planguage' to specify partial and conditional application of a single design idea. This is particularly useful in Evo planning since small increments of partial implementations is the 'name of the game'. This is part of the trick of breaking seemingly large ideas into smaller 'weekly' steps.

So let us take an example:

Design ideas:
Service:
Our shops or mail order distributors will undertake 24 hour turnaround service, or the service will be free.

Priority:
Large and frequent customers will be given priority in service, and in getting temporary replacement products while their product is being serviced.

Now let us look at possible ways of partially implementing the design ideas.

### Qualifiers for partial implementation in 'Evo steps'.

A 'qualifier' is a condition which must be fulfilled in order for its parameter to be valid.
For example in requirements 'Plan [Next Year]' means that the plan is valid for 'Next Year' only. Correspondingly, 'Service [USA & CAN]' or more explicitly 'Service [Country = {USA, CAN}]', means that the design ideas application is limited to application in two countries. Obviously, using this device alone we can regulate the spread of a design idea 'geographically', and that includes special categories within a country.

So, for example:

| Service [USA, Oil Industry] |
|---|

limits the application of the "Service' design idea to the country USA and the customer category 'Oil Industry'.

Time elements can be added to this picture:

| Service [GB, Electrical Shops, Weekdays]. |
|---|

And even event-related conditions:

| Service [NL, Toy Shops, Weekends, IF Shop Is Open]<br>Shop Is Open: CONDITION: True = Days when the shop is open for any kind of sales or inventory activity, for any period of time in the midnight to midnight range of a day, where employees are on the shop premise, even if not open to the public |
|---|

This use of the qualifier can be stretched to include any set of things useful.

### Qualifiers at Step Level.

Any useful set of design ideas can be grouped into sets:

| Good Ideas: {A, B, C OR D}. *"example of defining a mega-idea, notice the 'OR' "*<br>*Examples of steps derived from the mega-idea.*<br>Step 23: Good Ideas [USA, Weekends]<br>Step 24: Good Ideas [GB, Weekdays, IF Shop Is Open].<br>Step 25: Good Ideas [Rest of World, IF Shop Is Open]. |
|---|

Note that the ideas a so precisely defined that they are testable. All terms have a definition somewhere, even if we don't always show it here. The Capital letters imply a defined term.

## *More Using Planguage to specify Evo plans: Relationships.*

### Step Relationships, sequence and dependencies.

Planguage contains defined terms which can be used to formally specify step relationships. The terms are defined in the Glossary. Here are some examples.

Example:

| |
|---|
| X [X **STARTS BEFORE** A **ENDS**] meaning X must be started before A ends or completes successfully.<br>STEP23: XX [ XX AFTER A ENDS] meaning XX will only start when A is completed. |

In addition to defined words, there are corresponding defined '**keyed icons**'.
'ENDS' = '•' and 'BEFORE' = '=>'

| |
|---|
| X [X **=>** A• ] meaning X must be started before A ends or completes successfully.<br><br>STEP23: XX [ A• => XX] "meaning XX will only start when A is completed." A ENDS **BEFORE** XX **STARTS** |

*Example:*

| |
|---|
| X [ A BEFORE B STARTS], Y [ B AFTER C STARTS]. |

Example:

| |
|---|
| X [A =>I B] |

*meaning X is conditional upon A being* true (or completed) *before B starts, while . The 'I' means that A must be brought to completion before B can start.*

| |
|---|
| X [A => B] |

*meaning X is conditional upon A being* started *before B starts , A does not have to be completed or true before B starts.*

| |
|---|
| **THINGS BEFORE => THINGS AFTER.** |

| |
|---|
| **Stage-Liftoff**:{**Ignition-On** BEFORE **Check Thrust OK => Release Tie-down** STARTS}. |

All other components of 'Planguage' are available to express Evo plans. It is not the purpose of this book to explain **Planguage** in detail, but the Glossary is a relatively complete set of Planguage concepts. Planguage is not fundamental to Evo (with the possible exception of Impact Estimation Tables), merely a possible convenient notation.

## *The DoD Evolutionary Architecture Recommendation*

"The EA [Evolutionary Architecture] strategy should include the following elements thought necessary to ensure program success:

1. An Evolution Plan: An outline of the projected incremental allocation of capabilities and a time frame for their implementation. Included should be a time phased description of system interfaces, a guide for operational test planning and a basis for negotiating shared development and support responsibilities.

2. An Architectural Plan: A description of the principles on which the system architecture is based and the kinds of change that architecture can facilitate. It should include a set of guiding principles for management, development and maintenance; and an outline of how the architecture is expected to be improved in the future.

3. A Technology Road Map: A schedule for the availability of technology developments which relate to the system under development. This should include a survey of COTS products and a projected schedule for maturing emerging technologies.

4. A Funding Profile and Contract Strategy: A summary of the funding requirements for each incremental development, at least for the first increment. A contract strategy should be selected which tailors existing contract practice to the needs and structure of the evolving program. Early planning will also provide maximum opportunity to insure effective competition practice.

It may be useful to include a two phased approach in the acquisition plan to facilitate competitive benefits.

- The first phase would involve multiple awards with resulting contracts addressing the initial (or core) capability. Potential teaming arrangements would be indicated. Conceptual segments and approaches to incremental system performance improvement would be prepared and system specifications prepared at this time. In some cases, the plan might even provide for deliverable demonstration models.

- The second phase would involve selection of a contractor for system integration. Competition would be preserved at the second tier for each individual system increment development.

**5. A Product Assurance Plan**: Solid product assurance planning must link all aspects and phases of the system and be visible at decision milestones. Such planning should recognize specifically that in an evolutionary program, the developer's responsibility must extend through user fielded verification which might entail special maintenance or warranty provisions.

> *"the milestone approach is a major practice for us" Bill Gates in MS, 18*

**6. Integrated Logistic Support (ILS) Planning**: Support planning and analysis serves three purposes:

First, it determines the minimum investment in logistics support assets for the COTS core capability.

Second, it ensures that the evolving design concurrently pursues, and meets, both technical and support performance requirements.

Third, it tailors an optimal support program to sustain and measure performance over the expected service life." [DODEVO95]

## 5: Evo Step Objectives: Cycle Requirements

The purpose of this chapter is to explore the setting of detailed objectives for a single step. These objectives are necessarily a subset of the longer range objectives.

### *Determining the Quality Objectives for the next step.*

There would be no point in a step if we could not move the quality attributes some distance towards our long term objectives, or at least improve functionality. Contrary to traditional planning bureaucracies we do not view anything less than *real progress* for *real recipients* as interesting. Of course there are some overheads that will have to be suffered *within* the delivery cycle, but the cycle *will* aim to produce a useful improvement for real recipients and customers. It is a matter of pride, and a most useful discipline.

> *"A second concern is that it will be too difficult to make so many releases. If it is difficult to make one release every 9 to 18 months, how much more difficult will it be to release every two weeks? The answer is that when you make frequent releases, you get better at it (if this is not the case, EVO becomes too inefficient). Further, the small chunks in each cycle keep things to a manageable size." [MAY96]*

So the first question is which quality objective, or which functional change are we going to go for?

Part of the answer, obviously is it must be *practical*. It must not require something which is not 'in place' or ready. Common sense. The essential part of the answer is simply that it is something which will give the *greatest value* and *least cost* to the recipient, of all available alternatives. This is sort of like choosing the best chess move.

> *"When we reach milestones, we have not just functionality but size, performance requirements and quality requirements. ... So that milestone isn't met until [then]... Brad Silverberg, Sr. VP for Personal Systems Microsoft in MS, page 202*

The 'value' (benefit, quality, function) is primary, and other things being equal, we would choose the alternative which steals least of our limited resources. If in any doubt, ask a real recipient what they want. They cannot be unhappy if you constantly give them their first choices! Of course it is essential that such trial recipients be representative of any larger group of potential customers you are aiming for.

> *"The first maxim is the '80/20 rule'. The 80/20 rule observes that 80 % of results can be achieved with 20 % of an effort; it takes the remaining 80 % of effort to accomplish the remaining 20 percent of results. Under [Evo] the general guideline is to apply the 20 % of effort to accomplish the 80% of results and to be satisfied with that until the next delivery..... The second maxim is 'just-in-time engineering' ..... Requirements are not completed until design work must be initiated. Designs are not complete until shortly before they are to be implemented. In particular, requirements and designs for a future delivery are postponed until work on that future delivery begins. In general, engineering is completed just in time for the need. Again, this maxim must not be pursued to absurdity. In particular, inasmuch as the products of [Evo] engineering processes evolve throughout the successive deliveries, engineering data should be permanently deposited in these products as they become available." [SPUCK93, 6.3]*

There are many routes to the answer. And we are focussed on quick decisions rather than perfect decisions. An '80%-of-potential decision' delivered end of this week (and same next week), is worth a lot more than a '99%-of-potential' decision which takes three weeks to arrive at. We are simply going to arbitrarily cut off the decision making process, after a reasonably short time (like one day), so that we can focus on delivering something useful, rather than talking about it.

> *"Because many more project management decisions need to be made in Evo, handling decisions can also become a problem. If the decisions are not timely or cause dissension, progress can be delayed. Participatory decision-making techniques have been one solution at HP." [MAY96]*

If we make a bad decision, we are going to learn about it faster, by delivering and measuring, than agonizing for weeks about the truth. Scientists prioritize knowing the truth, at any cost. Engineers

(that's us) are more focussed on producing real working systems, even if they are not quite sure what the truth is about how they do it.

So you can use this recipe for starters:

1. Decide on one required quality objective which the recipient would most value improvement in (example 'Reliability').
2. Decide on an interesting minimal increment for the recipient. Use this as Must level.
3. Decide what might be accomplished by the best technology you can insert in the next cycle. Use this as a Plan level. *This technology will normally be extracted from the system architecture specification.*
4. Specify the technology you believe will get you the increase (in the Step specification). More on *choosing* the technology in next chapter.
5. Estimate that resources needed to implement *that* technology. Put this in the step plan.
6. If the resources exceed those available or permitted by step planning policy, go to step 2 and 'adjust'.
7. Estimate the impact of the step technology on all *other* qualities. Document your estimates. Example: in an Impact Table.

| | Thus Far | Next Step est. | Worst Case est. | Risks | Long term target |
|---|---|---|---|---|---|
| Reliability | 99.10% | 99.20% | 99.00% | NewTech =??? | 99.99% |
| Usability | 5 minutes | 5 minutes | = | none | 1 minute |
| Budget | 50,000 | 10,000 | 20,000 | Weekend work | 100,000 |
| Calendar Time | 300 days | 5 days | 10 days | Weather | 300 days left |

For example, above we decided to go for 'Reliability'. We estimate an increase to 99.20% is possible, but worst case, we could get worse. This reason is the 'NewTech' is dubious. However if we are right, we will make reasonable progress towards our longer term target of 99.99%.

Somewhat worrying is that we are quickly running out of money, and 'Usability' must be tackled. It is tempting to ask if there is not some much cheaper way to raise reliability. What I really need is a technology which will get me more Reliability at less cost. Now is the time to search for it. But we cannot wait weeks to find it, either. Some real progress has its own value for the recipient now.

## *Determining the Cost Budgets for the next step*

Cost budgets refer to *all the resources* we care to track. They typically include money, people hours, calendar time, and space; both as *initial investments* and *recurrent operational costs* or *future commitments.*

In the example above 'Budget' and 'Calendar Time' are examples of the concepts of cost budgets.

It is *logically impossible* to estimate a step cost budget until you know the following about the particular step:

- ✓ The design ideas to be used, and their costs.
- ✓ The functionality to be changed, and their costs.
- ✓ The process for *acquiring* the step content, and its cost.
- ✓ The process for *integrating and testing* the step content, and its cost.
- ✓ The process for *deploying* the step content to the recipient, and its cost.

But, it *is* possible to 'stipulate' an arbitrary cost budget, and then trim the step's cost-causing elements to correspond to it. This is known as 'design to cost', and is traditional engineering.

| Step Costs | Design idea costs | Functionality costs | **Acquisition** Costs | **Integration** and testing | **Deployment** costs |
|---|---|---|---|---|---|

| | | | | costs | |
|---|---|---|---|---|---|
| Budget | $20,000 | $1,000 | $2,000 | $3,000 | $4,000 |
| Calendar Time | | | 1 day | 6 days | 3 days |
| Engineering hours (Eh) | | | 20 Eh | 24 Eh | 12 Eh |
| Post deployment costs/year oper. | $1,000/year | 0 | 0 | $1,000 | |

The above table is an example of estimating the cost types for each cost requirement specification area. These costs represent a 'budget' idea. After deployment of the step, some of these costs are measurable. The measured cost information can be used to:

- ✓ Update real costs-to-date.
- ✓ Learn, from differences, about better cost estimation (estimation process improvement).
- ✓ Improve the step activities so as to reduce costs (step task process improvement).

## *Invoking other global constraints, or consciously not doing so.*

At each step there is the opportunity to audit and see whether any specified constraints have been violated. One way of doing this is in document quality control of the plans for the step. Another is to do periodic audits against defined constraints. A third is to integrate testing of constraint violation into the step tests and system tests.

*"For example, comments received from document reviews can be incorporated in the update of the document emanating from the next delivery: thus, many typical document iterations are eliminated. The chance to use a system while it evolves and to fix deficiencies as they arise is simply not available under Conventional Development Methods" [SPUCK93]*

# 6: Detailed Evo Step Design: Extracting function and design to make a step.

The purpose of this chapter is to show you how to extract a single step's **technology** content from the larger system **architecture**. It will *not* include a discussion of the step tasks (next chapter!) needed to manage the delivery of these design ideas. This is about *choosing the raw materials*, 'the cooking ingredients, for the meal' which are needed to satisfy your clients appetite and tastes.

This chapter is about the *general process of design*. We assume that a *rough* process of design, the architecture stage, has been done initially (Chapters 2 and 3). We assume that we will basically draw on those earlier, rougher, architectural ideas for our *detailed* step design ideas. But, we also assume that if we get any better ideas, we *will* revise the architecture, when the ideas are available, known, or proven through the Evo steps. We assume that the detailed practical step-level work will give us insights which were not available to the earlier architectural stages.

As a consequence, the architecture *specification* itself is probably *evolving* as the Evo steps progress. And, the design is getting *more detailed* as we are forced to confront the detail of each step. Each step can *teach* us something new, which can be used to *update* the architecture specification, which then is *available* to any *succeeding* steps as know-how about *what* works, how *well* it works and what it *costs*.

The key word is '*learning'*. Evolutionary management is a process for learning about *technology*, *economics* and *culture*. Learning what is not already *known*, or what is not already *certain* in this **host system.**

One major difference between Evo and conventional design or architecture methods, is that we will use less effort to *analyze* technology *theoretically*. We can more quickly, easily, and above all, more reliably try it out in *practice*. If it fails, we have lost little; lost less than the time needed to analyze the technology theoretically. If it succeeds, we can charge forward, secure in the knowledge of how it works. More timid competitors might not dare take a chance, or may delay implementation out of fear of the unknown. Meanwhile we have a competitive edge!

> *"Microsoft is one of those rare firms that has sustained and extended its market power. We think two principles we cite in this chapter explain how. First, Microsoft frequently makes incremental improvements and occasionally introduces major advances in its products. While often doing little more than packaging many incremental innovations, these major changes make older product versions obsolete. With a continual cycle of incremental and occasionally more radical innovations, competitors have little opportunity to challenge the market leader. Microsoft has accumulated enormous financial and technical resources that enable it to sustain this level of R&D. It has followed a somewhat unusual strategy: Dominant firms generally hesitate to introduce new products that steal sales from their existing product lines." MS,129*

We need specify less detail, than if we were building a system that must all work together correctly at once, upon final delivery and 'test flight'. We should be as ingenious as we can, in specification of **open architecture** (discussed above). This is the main architecture need for both the short and the long term. But, I have experienced that early Evo steps, are great teachers of the need and power of good open architecture, when we must observe the pain of incrementing new steps onto the host system, *without* such ideas. For once, that architect must witness the pain, experienced by post-delivery system maintainers and developers, and thankfully the architect is still fully empowered to correct unnecessarily closed architecture.

We will place less emphasis on 'making sure' all system components fit well together, in *advance* of any building or acquisition. We place more emphasis on ensuring that almost any credible technology changes can be easily accommodated during the initial evolution project, and in any later stages of life or 're-incarnation' of the system.

In earlier Chapters we hinted that technology is somehow extracted from the larger architecture, on the basis of perceived benefits and costs. We also hinted that if it were too large for our preferred step size, we were to somehow break it up so that it *was* a suitable size. How do we do that?

### How we extract ideas: which ones have priority?

We can approach the subject of which design ideas to use from several perspectives. A good general approach is to seek *value* (as defined by your *quality* requirements) for '*money'* (as defined by *cost* objectives).

*".. the synch-and-stabilize process ... provides several benefits that serve Microsoft well ... It facilitates competition based on customer feedback, product features, and short development times by providing a mechanism to incorporate customer inputs, set priorities, complete the most important parts first, and change or cut less important features." MS, 17*

An approach to this is to identify the largest current gap between the current levels of quality achievement, and the 'most threatening' requirements.

Threatening requirements are:
- ✓ Any Must levels not yet fulfilled.
- ✓ Then, any Plan levels not fulfilled

Threatening resources are:
- ✓ Any cost requirements which have exceeded Must levels, then
- ✓ Any cost requirements which soon threaten to exceed Must levels, then
- ✓ Any cost requirements which have exceeded Plan levels, then
- ✓ Any cost requirements which soon threaten to exceed Plan levels.

These requirements are 'threatening' in the sense that they threaten the project ability to achieve success (Plan) or avoid failure (Must).

Requirements threat analysis, as outlined above is both a form of 'risk analysis', and a way of determining project manager operational priorities. This is identical to what Ansoff calls 'Gap Analysis' in the quotation below.

*GAP ANALYSIS by Igor H. Ansoff*

*'The procedure within each step of the cascade is similar.*

*(1 ) A set of objectives is established.*
*(2) The difference (the 'gap') between the current position of the firm and the objectives is estimated.*
*(3) One or more courses of action (strategy) is proposed.*
*(4) These are tested for their 'gap-reducing properties.'*
*A course is accepted if it substantially closes the gaps; if it does not, new alternatives are tried."*
*... ← Igor H. Ansoff, "Corporate Strategy", 1965(25-26) Quoted in MINTZBERG94:44*

Here is a pictorial explanation of the residual gap idea.

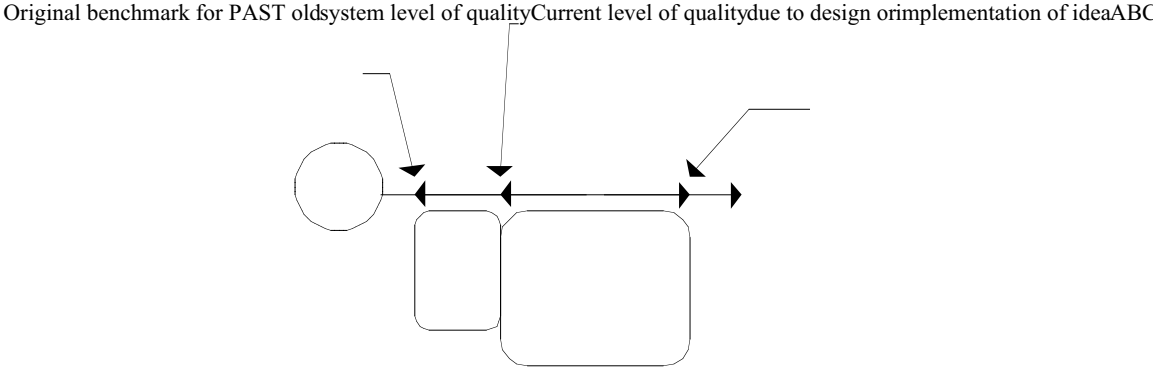



*Illustration of the residual requirement concept.*

So, 'gap analysis' gives us some idea of which requirements we must satisfy on the next step ('prioritize'). From this information we can search for appropriate design ideas, normally those in our

system level architecture, but if necessary, outside of it. The primary thing is to effectively close the gap, even if we were unable to recognize the need for it during our initial architecture definition.

*"It is important to observe that under [Evo] low-priority requirements often will never be implemented. They will 'fall off the end' of the development cycle. They are not implemented by the project because, as it proceeds from delivery to delivery, they are never selected for implementation. The development team consistently selects the most important things to do. What is left undone are those tasks that someone thought necessary but were never prioritized high enough to be included in the budgeted and scheduled deliveries. …… Nevertheless user priority must dominate delivery capability selection" [SPUCK93]*

A proposed design idea can be evaluated by the impact estimation method. If the idea can contribute to satisfaction of *other* priority requirements; good, that should be taken into account. If the idea uses little of constrained resources, *also* worth taking into account. If the idea has no or little *negative* effect on other requirements, good too. Notice that this evaluation is very much like the chess player's evaluation of a possible move. A chess move is very much an evolutionary step.

### A policy for evolutionary planning.

One way of guiding Evo planners is by means of a 'policy'. A general policy looks like this:

Evo Planning Policy (example)
1. Steps will be sequenced on the basis of their overall benefit to cost efficiency.
2. No step may normally exceed 2% of total project financial budget.
3. No step may normally exceed 2% of initial total project length.

**Profitability Control**
The *first* policy item tries to ensure that we maximize benefits on the official requirements values of the customer, while minimizing associated costs. If this is not in place, technologists consistently choose steps which are technologically interesting, and are not guided by the 'voice of the customer'. Obviously an Impact Table can be used to calculate which potential steps are to be done before others.

*"Benefits of Evo*
*The teams within Hewlett-Packard that have adopted Evolutionary Development as a project life cycle have done so with explicit benefits in mind. In addition to better meeting customer needs or hitting market windows, there have been a number of unexpected benefits, such as increased productivity and reduced risk, even the risks associated with changing the development process." [COTTON96]*

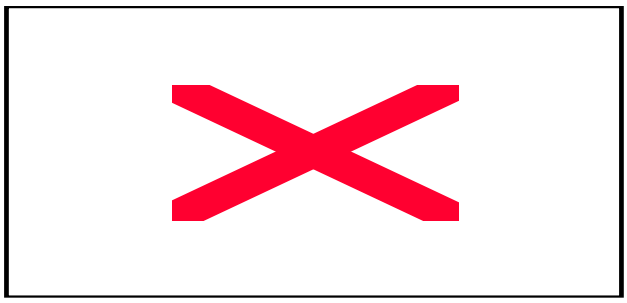


Fig. . Hitting market windows with a waterfall life cycle. [COTTON96]

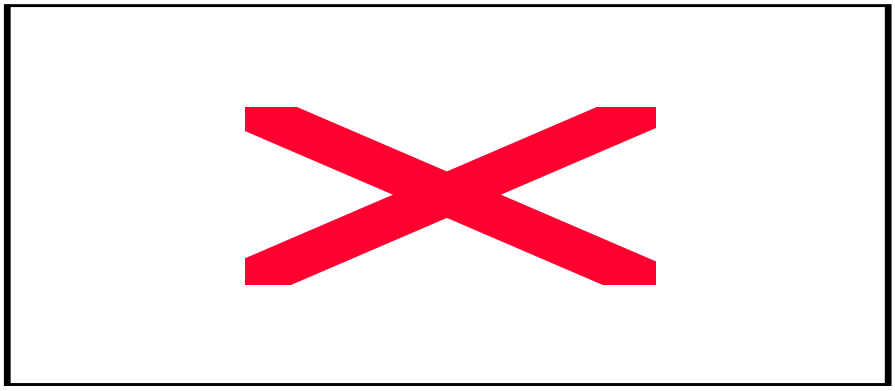


Fig. Hitting market windows with an evolutionary life cycle. [COTTON96]
*(edit note the checkmark from the original has become a bar, and should be fixed ).*

**Financial Control**
The *second* policy item is designed to limit our exposure to the financial risk of losing more than 2% of total project budget, if a step should fail totally. If this policy is not in place, technologists will insist on taking much larger financial risks, *with someone else's money*. They will wrongly, but sincerely, claim it *has* to be done, and there are 'no other alternatives'. This is usually due to lack of motivation, lack of strong management guidance, and lack of knowledge of how to divide steps into financially smaller expenditures. We will discuss how to handle the problem of decomposition below, shortly, in this Chapter.

This is really a 'gambling strategy'. How much can your project lose, and still survive, to complete the project as initially expected?

The 2% number is based on observation of common practice, but it is not holy and intelligent deviation is acceptable. It is dependent on how sure you are that a step may succeed or fail. The more conservative the step, the larger the financial risk you may be willing to consciously take.

**Deadline Control**
The *third* policy item is similar in principle to the second, except that the critical resource being controlled is calendar time. Again, the 2% is not a magic holy number. Intelligent deviation may be made. But, it is important to have a firm general guideline, or sincere people will always argue, wrongly, that they need more time before any delivery is possible. So, the 2% forces the issue. People have to argue their case well for deviation. And if they constantly deviate, they are to be suspected of incompetence. Give them help!

*"[Microsoft] Projects reserve about one-third of the total development period for unplanned contingencies or 'buffer time' allocated to each of the milestone subprojects. MS, page 200.*

The 2% is based on observation of common evolutionary increment sizes. For example IBM Federal Systems Division reported (Mills, 1980, IBM SJ 4-80, see Gilb88, page 104) that the 'LAMPS' project had 45 incremental deliveries in a 4 year period. About 2%. And, cycles were about monthly for 4 years.

It is critical to understand that this 2% cycle is *not necessarily* the **acquisition**, or **implementation cycle**. These two cycle tasks can be done independently, in advance, in the '**backroom'** (See Chapter 8). This '2%' cycle is above all the 'deployment' component where step content is delivered to the **recipient** and the **host system**. It is the frequency which the project interfaces with the . It is the frequency with which we expect to get feedback on the correctness of both our own evaluations, and the users evaluation. If we allow much larger step increments, the essence of evolutionary delivery management is lost. Worst; irretrievable time can go by totally wasted, without us being aware of the loss, until it is too late to recover.

*"The third tenet is extensive user interaction during development. This interaction refers not only to the feedback of requirements from one delivery to future deliveries, but also to the intimate involvement of the users during the implementation life cycle of each delivery. [Evo] embraces the premise that the more the real users are involved, the more effectively the system will meet their needs. Thus, the [Evo] process includes a role for the users in virtually every step of the delivery life cycle and involves them, at a minimum, in the key decision processes leading to each delivery." [SPUCK93, 2.3]*

Microsoft [CUSUMANO95] stressed, in connection with their 24-hour evolutionary cycles ( the 5 PM Daily Build of software) that it was vital to their interests to get feedback daily, so as not to lose a single day of progress without being aware of the problem.

*"The fifth key event [in Microsofts organizational evolution] was a 1989 retreat where top managers and developers grappled with how to reduce defects and proposed ... the idea of breaking a project into subprojects and milestones, which Publisher 1.0 did successfully in 1988. Another was to do daily builds of products, which several groups had done but without enforcing the goal of zero defects. These critical ideas would become the essence of the synch-and-stabilize process. Excel 3.0 (developed in 1989 and 1990) was the first Microsoft project that was large in size and a major revenue generator to use the new techniques, and it shipped only eleven days late" MS, 36*

While on the subject, note that Microsoft has a variety of cycle lengths, up to two years for a variety of purposes. Note also that evolutionary delivery is a 'way of life' for this very successful company.

> *"Many companies also put pieces of their products together frequently (usually not daily, but often biweekly or monthly). This is useful to determine what works and what does not, without waiting until the end of the project – which may be several years in duration". MS, 15*

Hewlett Packard [COTTON96] noted that their average step size was two weeks.

> *"There are two other variations to Tom Gilb's guidelines that we have found useful within Hewlett-Packard. First, the guideline that each cycle represent less than 5% of the overall implementation effort has translated into cycle lengths of one to four weeks, with two weeks being the most common. Second, ordering the content of the cycles is used within Hewlett-Packard as a key risk-management opportunity. Instead of implementing the most useful and easiest features first, many development teams choose to implement in an order that gives the earliest insight into key areas of risk for the project, such as performance, ease of use, or managing dependencies with other teams." [COTTON96]*

### *How we reduce step size.*

Almost everyone has various problems reducing their initial concepts of a step to a suitable '2%' size. It is, however, almost invariably possible to do so. Everyone can be taught how to do it. But many highly educated, intelligent, experienced 'engineering directors' cannot quite believe this until they experience it themselves. I believe this is a *cultural* problem, not a technological problem.

Here is a summary of what I do, and how I teach, about the art of decomposing larger step ideas into smaller sizes.

*How to decompose systems into small evolutionary steps.*

*1•* Believe *there is a way to do it, you just have not* found *it yet!*

*2•* Identify *obstacles, but don't use them as excuses: use your imagination to get* rid *of them!*

*3• Focus on* some usefulness *for the user or customer, however small.*

*4• Do <u>not</u> focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!*

*5• Think; one customer, tomorrow, one interesting improvement.*

*6• Focus on the* results *(which you should have defined in your goals, moving toward PLAN levels).*

*7• Don't be afraid to use temporary-scaffolding designs. Their cost must be seen in the light of the value of making some progress, and getting practical experience.*

*8• Don't be worried that your design is inelegant; it is results that count, not style.*

*9• Don't be afraid that the customer won't like it.* If *you are focusing on results* they want*, then by definition,* they *should like it. If you are not, then* do*!*

*10• Don't get so worried about "what might happen afterwards" that you can make no practical progress.*

*11• You cannot foresee everything. Don't even* think *about it!*

*12• If you focus on helping your customer in practice,* now*, where they* really *need it, you will be forgiven a lot of 'sins'!*

*13• You can understand things much better, by getting* some *practical experience (and removing* some *of your fears).*

*14• Do* early *cycles, on willing local mature parts of your user community.*

*15• When some cycles, like a purchase-order cycle, take a long time, initiate them early, and do other useful cycles while you wait.*

*16• If something seems to need to wait for 'the big new system', ask if you cannot usefully do it with the 'awful old system', so as to pilot it realistically, and perhaps alleviate some 'pain' in the old system.*

*17• If something seems too costly to buy, for limited initial use, see if you can negotiate some kind of 'pay as you really use' contract. Most suppliers would like to do this to get your patronage, and to avoid competitors making the same deal.*

*18• If* you *can't think of some useful small cycles, then talk directly with the real 'customer' or end user. They probably have dozens of suggestions.*

*19• Talk with end users in* any *case, they have insights you need.*

*20• Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.*

This list is compiled by mental review of my practices and experiences. One or more of these hints should enable you to find a practical solution. Finally, when you have enough successful experience to realize that there always seems to be a way to decompose into small evolutionary steps, you will give up the *'impossible!' mentality*, and concentrate on the constructive work of finding a reasonable solution. The *'can't do it' mentality* is built on a number of misconceptions so let us deal with them.

**Misconceptions about Evo step decomposition.**

By decomposition, we do not mean that '2% of a car' is going to delivered. We will probably be dealing with 'whole cars'. But we may start with our old car and put better tires on it, to improve braking ability on ice.

The problem of getting 'critical mass', meaning enough system to do any useful job with real users, is usually dealt with by one of two methods:

- ✓ Make use of existing systems
- ✓ Build the critical mass in the 'backroom', invisible to the user.

The conceptual problem of 'dividing up the indivisible', which people wrongly perceive as a problem, is explained by pointing out that:

- ✓ We do *not* ask you to Solomon-like 'split the baby in half',
- ✓ We are *not* talking about system 'construction'
- ✓ We *are* talking about delivering 'results', not technical components, in small increments.

So, the systems are *organically whole*. No unreasonable chopping them into non-viable components is being asked. But we are still not going to overwhelm the user with all the capability of the system.

For example, you probably learned PC software, like your word processor or email capability gradually. It might have been delivered to you as five million software instructions from Microsoft, with 20,000 features, most of which *you* do not need. But you could ignore most of them, and focus on getting your current job done. You probably picked up additional tricks gradually. You found them by experimentation with the menus. A friend showed you some tricks. You might be one of those weirdoes who actually read a manual. Maybe you looked new capability up using the Help menu. Maybe you got help from a help desk. You probably did not go on a two-week course, with an examination, to learn to be proficient at a high level. Nevertheless, the entire system was 'delivered' to you all at once. Then a new version became available, a million lines of code at once, but again you probably used it same way as last week, picking up new features gradually. Microsoft is not worried about building these features as each new user needs them. They build what they must for their market in their 'backroom', and we deliver to ourselves, in our 'frontroom', at an evolutionary learning pace, based on need and ability.

The key idea is the incremental *use* of a system, which increases the *user* capability. Building and construction are semi-independent topics. You *can* build a system in the same step cycle time as it takes to deploy with a user, but you do not *have* to!

### System Function is stable, it is quality that improves incrementally.

Much evolutionary delivery has relatively constant raw *basic function*. Word processors help you write, telephones help you talk remotely, the TV gets sound and picture to you, the car moves people from place to place. Yet there is a constant stream of new products which we buy and replace our old products with. What are we buying? The answer is 'quality'. Even that which we choose to perceive as 'functionality' is probably, when you ask 'why is it there?', really *'design'* in order to achieve some quality level such as usability, security, portability or the like.

So, one view of Evo is that it is the stream of gradual improvement to the quality and performance aspects of the system. Basic functionality really does not change in a class of systems (telephone, car, ATM Cash Dispenser, computer, person, organization).

The art of extracting small Evo steps is to focus on immediate but continuous improvement of the targeted quality requirements. Steps deliver increments of quality, by means of implementation of 'design ideas'.

This is often done by gradually 'milking' some larger system, a large capacity machine, an organization, a database, a computer program.

### Using a Large Scale Design Idea at variable and low initial costs.

We are not obliged to use the *entire* larger system, to get what we want. We are not obliged to *pay* for the entire larger system, just because we use part of it. We can usually negotiate some sort of 'pay by use degree' agreement, so that we do not get unprofitable costs at early stages of use. This may not be the 'normal way' to pay for that system, but it *can* be done.

It is in the interest of the supplier of that system that you begin to use *their* system, not a competitor. It is in their interest that you *become* a large scale long-term customer. So, I have found that, especially if you talk to people high enough in their organization to be empowered to make a deal like this, you can *use* large costly systems without paying the full bill. The salesman may say 'no', but the Sales Director will find a way to say 'yes'.

This is critical for *smooth* evolutionary growth. You can grow *into* a larger system, without worrying about swapping to a new larger system. Economically, the supplier is only paying some kind of 'interest cost' on their production outlay. In the case of 'software', of course that cost is negligible.

People who cannot see this possibility, or who will not take the trouble to negotiate a special deal for themselves, will not be able to understand the evolutionary possibilities. But, people like that can therefore not act in the best interests of their project and their organization.

### No Cure, No Pay.

If a design idea, a product, a royalty charged method, or anything else, is not giving the results you expect, then you should not have to pay. You must make agreements and contracts so that payment is contingent upon satisfactory benefit, and consequent continued use. Not merely initial use. A supplier who will not somehow make that deal, has little faith in their product.

We are not merely obliged to minimize our losses using small-risk steps. Our steps need to be designed as the experiments they are. We should arrange to avoid 'getting stuck with the bill', when design ideas do not work as advertised or as expected.

### Spreading Design Ideas by Using Qualifiers

Another frequently used way to divide larger design idea concepts into smaller implementable steps is to implement them different places. For example:

- ✓ Geographic territory
- ✓ By type of user or customer
- ✓ By your office sites
- ✓ By priority (most value to customer or to you)
- ✓ In low risk areas first.

The specification will use the '[qualifier]'. For example: ('High Capacity' is the name of a defined design idea). Four varied examples of the use of the qualifier to specify steps are given below:

- ✓ Step 1: High Capacity [Norway "trial market"], Step 2: High Capacity [ {Euro, NATO}], Step 3: High Capacity [All other European Continent].

- ✓ Step 1: High Capacity [ Offices], Step 2: High Capacity [ Factories], Step 3: High Capacity [ Chains].

- ✓ Step 1: High Capacity [ GB, Office], Step 2: High Capacity [ Euro, Chains], Step 3: High Capacity [ {Warehouses, Small Shops}, {USA, Central America, South America, Canada}].

- ✓ Step 1: High Capacity [ Volunteers, Small Sites], Step 2: High Capacity [ Critical Prospects], Step 3: High Capacity [ Large Customers].

This allows us to take any architecture idea, however seemingly large, and 'distribute' it into an interesting series of implementations. What we learn in previous implementations may be very useful in the later ones.

**What can we do next week, to please a customer?**

I have found the question of what we can do 'next week' to somehow make progress, to be a surprisingly powerful question. It tends to have interesting answers!

Forget everything else. Forget the total integrated long-term dream. Forget the complex new system. Don't worry so much! Just focus on what you can do 'next week'.

Without exception in about 25 aircraft design or maintenance teams, on five real project workshop training courses, we found something we could do (CA, 1988). Without exception, the managers of the team gave approval to go ahead and do it. We manage to get that approval almost every time, anywhere, any project.

Why not. Nothing to lose. Everything to gain. Worst case: no progress. Fine, we learn something about our optimism, about thinking things are simpler than they turn out to be. The managers found it predictably approvable ( we told project teams on the Monday they would seek and get approval on the Friday).

It is worth noting that we tried to involve the manager-in-question every day, at least half an hour, with their team. That way the manager gave necessary correction, understood the process, and bought into the idea. Quick visible results are very attractive. And they need not be short term sub-optimization, since we make sure they are measurably on the path to the longer term measurable requirements. We make sure that they exploit the new longer term architecture.

My worst 'disappointment', in getting action next week on a step, was with a U.S. Army Brigadier General in the Pentagon. He did sort of approve it. And he had been 'out playing golf' earlier in the week. So, he had not been briefed all week long, as we *try* to practice with the 'approval manager'. But, he wanted to spend the next two weeks refining the *planning documentation*, so he could brag about this great new planning method to his buddies, and not embarrass himself with the quick-and-dirty detail of our first week with his managers. At least there was no war on then! (He did give me a nice diploma, and pin, I thought of it as a 'medal', on the Friday! That is some kind of result, I suppose). The planning documentation did indeed look very impressive a few weeks later. I like to think that his managers went ahead and started improving the system, in spite of their General. They told him *they* believed in the next week's step, and had the power to put it in action that next

week. This was a concept they themselves scoffed at when I told them on the Monday that they would be doing it. The next week's implementation step was 'an improvement in priority settings for the big brass', so they got better service from the system. (Pentagon, August 1991)

**Success, no matter what the outcome!**

So what, if it *really* takes a month! We *learn*, and we have accomplished *something*. You do not have to announce to the whole world you *intended* to improve things next week!

If you fail, *declare success* in having performed a limited 'experiment' to show how well or poorly something *risky* worked, which you will *not* now make the mistake of threatening the larger project with. If it succeeds, declare success in making *real* progress, while others suffer 'analysis paralysis'.

## *Chapter Summary: 6. Detailed Evo Step Design: Extracting function and design to make a step.*

You need to have a detailed design process at the step level which can extract suitably profitable, but suitably low risk steps. The residual requirements are your priority guide as to what needs to be done at a particular step.

# 7: Planning the Evo Step: The delivery cycle in detail.

This chapter will detail the process of 'cooking and serving' the design ideas raw ingredients, so as to give the impacts expected.
So, you have decided you want Paella for dinner tonight (requirements). You have decided what ingredients it is going to have (design). Now you have to get the ingredients, prepare them, taste them and serve them. Maybe you even have to wash up afterwards? Maybe even get the bill paid.

This is the chapter about all the stuff an evolutionary cycle has to manage, before the step result can become a reality.

> "Despite the uncertainties that characterize a program utilizing an alternative [Evo] approach, it is of paramount importance that each design increment be preceded by a complete, and unambiguous articulation of system requirements for that particular increment. Additionally, an adequate understanding of the 'final' system's capability must be known in order to provide for incremental designs that will allow future enhancements without negating previous design efforts. If either of these components are lacking, then acquisition design initiation should not proceed." [DODEVO95]

## *Choosing design components and estimating their impacts. 'Plan'.*

This step has been discussed in detail above. It is shown here for completeness of understanding the delivery cycle detail. I will use the process control *Plan Do Study Act* paradigm of Shewhart/Deming, to describe the activity of the delivery cycle (**PDSA Cycle**).

Step requirements are selected, candidate design ideas are evaluated (Impact Estimation) and when a satisfactory set of design components is established, we are ready to estimate their costs.

## *Estimating the step resources. 'Plan'.*

We covered cost estimation for the step earlier. But it needs to be included here as a reminder that it is a part of the delivery cycle process. This step is only meaningful if the design components chosen give adequate benefits as estimated against the requirements.

If the costs exceed Evo planning policy threshold limits (like '2%' of budgeted time or money discussed above) then it may be necessary to go back to the previous step, adjust the design, or to accept the risk of higher than normal cost levels.

The outcome of this sub-step task is that we hypothetically have identified design components which will give us results we find useful and acceptable, at costs we can afford.

## *Acquisition: of design components for the step level. 'Do'.*

Acquisition brings design components into the delivery cycle. They may have been acquired to some degree by a process outside the step acquisition cycle, or they may be something we are going to have to deal with right now.

Here are some ways we may have made design ideas 'ready' to pull into our step:

- ✓ We have bought or other wise arranged for it. It is waiting for us to use it as a result of a '*backroom* ' process.
- ✓ We have previously developed it, inside our *own* research and development. Again this is a *backroom* process. It is now ready, and awaiting potential use in *this* delivery cycle.

But, maybe it is only an item on our list of design ideas, and we must acquire it, for example:

- ✓ By immediate purchase/rental/lease/contract/agreement,
- ✓ By accessing it from a web site (data bases, processes, programs, other information)
- ✓ By building it ourselves during this cycle (programming, hardware construction, gathering data, writing a handbook, or training materials)
- ✓ By re-deploying machines or devices used by us for other purposes previously (example loading a new system onto our existing computers, communications, office facilities).

In all these cases we need to move all necessary design element into our *frontroom*. We need to get the cooking ingredients on our kitchen table, whether they come from our cupboards, or must be shopped for.

### *Quality Control of step level components. 'Do'.*

We need to check that the ingredients are 'fresh and tasty'. So, various forms of quality control of the design components, are needed to make sure our components will not spoil the system after integration.

### *Integrating step level components. 'Do'.*

If we have several design components, we may need to mix them before integration with the host.

They need to be then added to the host system, even if there is only one component.

Just because we have integrated the step components with the host system, does not imply that recipients have access to them, or that we can expect any measurable results from the user end.

### *Testing the upgraded host system. 'Do'.*

We need to thoroughly test that the new components do what they are supposed to do, and that the host system is not unexpectedly degraded as a result of this integration.

> *"With an Evo approach, the team has greater flexibility as the market window approaches. Two attributes of Evo contribute to this flexibility. First, the sequencing of functionality during the implementation phase is such that "must have" features are completed as early as possible, while the "high want" features are delayed until the later Evo cycles. Second, since each cycle of the implementation phase is expected to generate a 'complete' release, much of the integration testing has already been completed. Any of the last several Evo cycles can become release candidates after a final round of integration and system test. When an earlier-than-planned release is needed, the last one or two Evo cycles can be skipped as long as a viable product already exists. If a limited number of key features are still needed, an additional Evo cycle or two can be defined and implemented..." [COTTON96]*

### *Deploying the upgraded host system. 'Do'.*

We are now ready to announce the existence of the upgraded host system to the recipients, and trial the system at the user level.

This is where we get the users involved. They will need training, coaching, support. At some stage when things have settled down (perhaps are statistically stable) we can undertake to measure the results.

### *Measurement of the step results. 'Study'*

We need to measure the results of deployment:

- ✓ The cumulative effects (how the entire host system qualities and total project costs are)
- ✓ The differential effects (caused by the last step): these will be compared to your estimates.
- ✓ The opinions of the recipients and other involved parties (if they hate it, get ready to reverse)

### *Analysis of the step results. 'Study'.*

Your project team needs to study the information so that:

- ✓ You can plan the next step better
- ✓ You can make better long term estimates (especially costs, time to market)
- ✓ You can adjust the step quickly if there is some problem
- ✓ You can yank out the step if it is causing more grief than pleasure.
- ✓ You can update estimates of quality effects for larger-scale and longer-term use of design ideas which were measured in this step. (Feedback to architect and project management).

### *Correction of process and plans as a result of step result analysis. 'Act'.*

As indicated, the 'study' activity will normally give you insights which need to be *immediately* exploited, next step. Evo is a form of 'continuous process improvement' *during* the project. It is a form of continuous process improvement for the *project* primarily, rather than the large organization.

However, the company will wisely somehow convert lessons into action to positively impact other projects, even ones running in parallel with this one.

> *"With Evo, the software implementation cycle is dramatically reduced and repeated multiple times for each project. All parameters of the implementation process are now available for review and improvement. The impact of changes in processes and tools can be measured and refined throughout the implementation phase" [COTTON96].*

## *Possibility of total reversal.*

When a new step is deployed, the project team should be intensely and directly watching what happens, on site with users. They should be ready to 'slam into reverse' or 'pull the plug'. They need to take immediate action before any damage becomes irreversible. They need to move the users back to the status before the step deployment, if necessary.

This ability to reverse needs to be part of the step deployment plan. There should be no irreversible steps.

## *Step Tasks determination and notation*

| DESCRIPTION | SHORT NAME | Start | End | Engineering Hours est. | Responsible | Documents Produced |
|---|---|---|---|---|---|---|
| Choose design components & Estimate Benefit Impacts | Design | | | | | |
| Estimating the step resources | Cost | | | | | |
| Acquisition: of design components for the step level | Acquire | | | | | |
| Quality Control of step level components | Check | | | | | |
| Integrating step level components | Integrate | | | | | |
| Testing the upgraded host system | Test | | | | | |
| Deploying the upgraded host system | Deploy | | | | | |
| Measurement of the step results. | Measure | | | | | |
| Analysis of the step results | Study | | | | | |
| Correction of process and plans as a result of step result analysis. | Act | | | | | |

Above is a 'form' for controlling each delivery step tasks. You might like to think of it as the 'weeks' plan.

> *"Now I don't know about perfect specs and perfect estimates, but there's no way in hell we're going to have perfect knowledge about what the competition is doing [or] … which way this very exciting industry is moving. The old scheduling method is just adding up all the stuff you know about and saying that's the date – that's all we know about. Are we at all surprised that it generates a date which is too little, considering how little we know about the future? … The way you do an accurate ship date is*

*that you add buffer time, which is time that says the future is uncertain, which it clearly and obviously is. This isn't not being hard-core or being a wimp. This is just saying the future is the future. It is self-evident." Chris Peters (VP Office, Microsoft), MS, page 206*

You could embellish the plan with activities which might be natural for your particular situation such as:

- ✓ Training the user recipients
- ✓ Reporting data to a project database
- ✓ Getting customer signoff for the apparently successful deployment, first day
- ✓ Sending a step invoice to the customer, as consequence of signoff
- ✓ Getting customers final approval of the step 30 days after deployment
- ✓ Gathering data on the step after 30 days deployment

*"Schedule Estimates for Fine-Grain Tasks:*

*A second Microsoft scheduling approach tries to 'force' more realism and avoid wild underestimates (as occurred on Word for Windows in the 1980s and many other projects) by asking developers to give their estimates for implementation activities based on a very detailed consideration of tasks to complete. The level of granularity for the tasks is usually between four hours (a half day) and three days." MS, page 253-4*

## *Managing the Evo step contractually.*

Here is an example of a contract for evolutionary delivery, which I made for a client:

"Design idea for this contract modification: designed to work within the scope of present contract with minimum modification. An Evo step is considered a step on the path to delivering a phase.
You can choose to declare this paragraph has priority over conflicting statements, or to clean up other conflicting statements."

§30. Evolutionary Result Delivery Management.

30.1 Precedence. This paragraph has precedence over conflicting paragraphs.
30.2 Steps of a Phase. The Society may optionally undertake to specify, accept and pay for evolutionary usable increments of delivery, of the defined Phase, of any size. These are hereafter called "Steps".
30.3 Step Size. Step size can vary as needed and desired by the Society, but is assumed to usually be based on a regular weekly cycle duration.
30.4 Intent. The intent of this evolutionary project management method is that the Society shall gain several benefits: earlier delivery of prioritized system components, limited risk, ability to improve specification after gaining experience, incremental learning of use of the new system, better visibility of project progress, and many other benefits. This method is the best known way to control software projects (now US DoD Mil Standard 498. 1994).
30.5 Specification Improvement. All specification of requirements and design for a phase will be considered a framework for planning, not a frozen definition. The Society shall be free to improve upon such specification in any way that suits their interests, at any time. This includes any extension, change or retraction of framework specification which the Society needs.
30.6 Payment for Acceptable Results. Estimates given in proposals are based on initial requirements, and are for budgeting and planning purposes. Actual payment will be based on successful acceptable delivery to the Society in Evolutionary Step deliveries, fully under Society Control. The Society is not obliged to pay for results which do not conform to the Society-agreed Step Requirements Specification.

30.7 Payment Mechanism. Invoicing will be on a Step basis triggered by end of Step preliminary (same day) signed acceptance that the Step is apparently as defined in Step Requirements. If Society experience during the 30 day payment due period demonstrates that there is a breach of specified Step requirements, and this is not satisfactorily resolved by the Company, then a Stop Payment signal for that Step can be sent and will be respected until the problem is resolved to meet specified Step Requirements.
30.8 Invoicing Basis. The documented time and materials will be the basis for invoicing a Step. An estimate of the Step costs will be made by
the Company in advance and form a part of the Step Plan, approved by the Society.
30.9 Deviation. Deviation plus or minus of up to 100% from Step cost and times estimates will normally be acceptable (because they are small in absolute terms), as long as the Step

Here, for another client is an example of an evolutionary policy.

The Buyer's Project Policy
Nov. 21 1996 Version 0.2
Owner: The Supplier Project Leader for The Buyer
Author: TG

Objective: to create a relationship for The Buyer which

• removes problems caused by dynamically changing and evolving requirements.
• gives The Buyer rapid actual usable system improvement.
• gives The Buyer complete control of cost (no cure no pay).
• gives The Buyer complete flexibility to change requirements to suit current insights into their critical needs.
• gives The Supplier the ability to focus on delivering satisfactory real improvements to the way The Buyer does business.
• creates a sound basis for a happy long term relationship between the parties based on delivered value for money, as judged by The Buyer.

THE EVOLUTIONARY RESULT DELIVERY POLICY
1. The current project will continue by planning to deliver customer usable/evaluatable system improvements in approximately weekly intervals.
2. The precise increment requirements will be settled at the week beginning from a menu of interesting options, as selected by the Customer.
3. The increment will be intentionally scaled down to probably be doable within the scope of a week, but shorter or longer cycles may be agreed as needed.
4. The agreed incremental result delivery will be normally delivered to the client for their appraisal and use by Friday morning.
5. The Customer will preliminarily evaluate it by end of day.
6. If it meets agreed requirements the customer will formally indicate that an invoice for the incremental effort can be sent, payable within 30 days. If not accepted, reasons will be given in writing, which relate to failure to meet agreed written specifications.
7. Payment is effectively due when no hidden problems are discovered in the next 30 days in which payment is due, which invalidate acceptance. I.e. that it did not in fact meet specified requirements. Written notice giving details of failure to meet specified requirements will be given as a basis for holding up payment.

8. The Supplier is responsible for rectifying any previously unacceptable delivery increments before proceeding to do any later work on the project.

Here is a template made for this client to document each Evo step:

Evolutionary Delivery Step Plan (the Form)

Buyer Requirements
Functional Requirements

Benefit/Quality/Performance Requirements
Tag:_____
GIST: _____
SCALE:_____
METER [END STEP ACCEPTANCE TEST] ___
PAST[WHEN?, WHERE?] ___
MUST [when?, where?]_____
PLAN[when?, where?]_____
Tag:_____
GIST: _____
SCALE:_____
METER [END STEP ACCEPTANCE TEST] ___
PAST[WHEN?, WHERE?] ___
MUST [when?, where?]_____
PLAN[when?, where?]_____

Resource Constraints:
Calendar Time:
Work-Hours:
Qualified People:
Money (Specific Cost Constraints for this step):
*Other Constraints*
*Design Constraints*
*Legal Constraints*
*Generic Cost Constraints*
*Quality Constraints*

Assumptions:

Dependencies:
Design:
Technical Design (for Benefit Cost requirements)

Tag:
Description (or pointer to tags defining it):
Expected impacts:
Evidence (for expected level of impacts)
Source (of evidence)

Tag:
Description (or pointer to tags defining it):
Expected impacts:
Evidence (for expected level of impacts)

Source (of evidence)

Tag:
Description (or pointer to tags defining it):
Expected impacts:
Evidence (for expected level of impacts)
Source (of evidence)

Tag:
Description (or pointer to tags defining it):
Expected impacts:
Evidence (for expected level of impacts)
Source (of evidence)

Test Design

Supplier Test Plan:
Customer Acceptance Testing Plan:
First day trial:
30 Day trial:

Documentation Design:
Training Design:

Estimates:

Estimated Cost $
Estimated work-hours

Actual Cost $
Actual Work-hours
Reasons for differences:

Cost
Work-hours

Signoffs

Customer accepts and supports the plan (esp. requirements)
Customer Accepts that requirements are met during first trial day (Invoice can be sent): _____signature
Comments:
Changes desired (new requirements):
Customer accepts that Invoice can be paid for this increment :
_____ sign.

Here is a US Department of Defense View:

*"Among categories of factors which influence (Evolutionary Acquisition) EA are requirements uncertainties, technical uncertainties, funding availability, schedule problems, interoperability and commonality requirements, the need in some kinds of systems for continuous user involvement, and instabilities due to environment. An evolutionary process may be especially effective when change to any of these factors is likely during the time period of system development.*

*The major approach which underlies EA is encouraging early fielding of a well defined core capability in response to a validated requirement. This, while planning actions which will, within an approved architectural framework, enhance that core and ultimately provide a complete system with the required overall capabilities. Senior leadership must be actively involved in such a strategy.*

*Each incremental capability to be acquired is treated as a tailored individual acquisition. Scope and content result from both continuous feedback from the developer, independent testing agencies, the user (operating forces) and supporting organizations; and application of desirable technology within the constraints of time, requirements, cost and risk." [DODEVO95]*

## *Chapter Summary: 7: Planning the Evo Step: The delivery cycle in detail.*

An Evo step is a process, which itself is composed of a number of administrative sub-processes:

| **Design** |
|---|
| **Cost** |
| **Acquire** |
| **Check** |
| **Integrate** |
| **Test** |
| **Deploy** |
| **Measure** |
| **Study** |
| **Act** |

These sub-processes constitute the detailed planning and control of the step, along with the learning from experience which each step provides for the project future.

## 8: The Evo Backroom: Readying components for packaging and delivery.

The purpose of this chapter is to explore the role of 'backroom' preparation of the Frontroom Evo delivery steps.

| **Backroom →** | **Frontroom →** | **User :>)** |
|---|---|---|
| Design Idea being made ready | Not ready | Unaware of these activities |
| Whole steps being made ready | Not ready | Unaware of these activities |
| Steps which are ready | Not selected | Might be offered this option |
| Other steps which are ready | Not Selected | Might be offered this option |
| High Priority Step which is now ready | Pull to Frontroom, Go through Frontroom step process, Deploy | Receives this step, |

The simplest form of Evo doesn't have a 'Backroom'. After the initial architecture and planning stages ('Step Zero') a simple Evo process extracts steps from the basic architecture, and the Step Process does whatever is needed to deploy that step within the step cycle. This seems to work best when the project is in control of the resources needed to build each step.

### *Why Backroom Activity May Be Needed*

The moment the time needed to acquire a step design component is longer than the step cycle policy restriction (of for example 2%, or a week for a one year project), this is no longer realistic. The acquisition time can be longer because of for example these factors:

- ✓ Time needed to make a choice (which might have long term consequences)
- ✓ Time needed to build and test a component
- ✓ Time needed to go through bureaucratic, but required procedures
- ✓ Time needed to test or quality control a component
- ✓ Time needed to integrate design ideas with each other for the same step.

*"Daily Build Process [at Microsoft]*

1. *Check Out [copies of code from master version]*
2. *Implement Feature.*
3. *Build Private Release.*
4. *Test Private Release.*
5. *Synch Code Changes [use compare tool to make sure no changes since checkout].*
6. *Merge Code Changes.*
7. *Build Private Release [merged with other developers' changes]*
8. *Test Private Release*
9. *Execute Quick Test.*
10. *Check In.*
11. *Generate Daily Build.*
12. *Execute automated tests*
13. *Make Build Available to All Project Personnel*

*(including program managers, developers, testers, and user education staff for their use and evaluation)"*

*Headlines only, from MS, page 264-7*

*This is clearly a form of evolutionary cycle which goes to 'internal users'. Microsoft has another level (6 to 8 weeks duration) called "Milestone" , 3 or 4 of which make up a "Project", which again may well have to synchronize and integrate with other projects to form a delivery such as an Office version.*

*I have heard from first hand observers that Microsoft and others (Siemens) ship these daily builds by satellite to other continents (China, India, USA) to test during their night and return the next morning.*

In addition to preparation time to get a step ready to pull into an ordinary step cycle, a step which is ready, may be kept in the 'Backroom' waiting, for a variety of reasons:

- ✓ The step has a lower priority or interest for the user, than other currently selected steps
- ✓ Formal permissions have not been obtained
- ✓ The user is not able to absorb that particular step, in terms of training, or other host system capacity restrictions

- ✓ There is a delay in implementing steps which have already been started.

For any of these, or similar, reasons a step may be forced to languish in the Backroom, until its time has arrived.

## *Backroom Independence of Short Delivery Cycle Requirements*

The Backroom, is not subject to the regular short '2%' cycles of delivery to the user. Steps can take whatever time they need. If they are delayed in the Backroom, then they are simply not available for deployment to the users. Hopefully *something* is ready. Some companies use the concept of a 'train' ready to pull out from the station. Passengers and goods which are there, and ready, can join the train, but late ones must wait for a later train (future delivery steps).

Conceptually the Backroom solves the problem many people have about how to chop up certain things into 2% cycles. The 2% cycles are primarily a 'rhythm of delivery' to users. If 'necessary acquisition' for a step fits nicely into that cycle. Fine. If not, don't worry. But, try to keep the flow of Backroom projects 'cooking' so that you always have something to serve your customer.

## *Parallel Backroom Activity*

By its nature, Backroom activity is parallel. At the extreme, all steps could take an average of 26 weeks to prepare in the backroom. One new step could be initiated in the backroom every week. And every week a step could be delivered to the user. As many as 26 steps could be in preparation in parallel in the Backroom, in the middle of the project. This is similar to preparing chess moves. They player has a lot of ideas of possible moves (steps) they are considering. Some of them require considerable preparation. Some never get done. Some new moves occur as potential, and take high priority over those you have been considering for so long.

## *The Risk of Backroom Early Step Preparation*

This points out the problem with backroom preparation. There is the risk that events and feedback after you have initiated backroom activity, make you realize that you do not need that step at all, or you need a modified version of it. In other words, the price you might have to pay, for the advantage of parallel activity and Backroom preparation, is the *loss incurred* because what you have ordered is not what you ultimately decide you want. We risk losing one of the prime advantages of Evo delivery if we commit too much too early, before we have enough feedback to know if that is what we *really* want to do!

But we are going to take some risk. A calculated risk. As long as we are willing to face the potential losses, in order to gain the perceived advantages. There are some steps you can take, to mitigate the damage, if it turns out you want something different from what you initiated in the backroom:

## *How to Avoid or Mitigate Losses Due to Premature Backroom Activity.*

Here are some Proactive Tactics (to avoid *any* damage to yourself)

- ✓ If ordering from sub-suppliers, contract for 'payment of the quantity you actually need and use'. Point out that this quantity is uncertain, and you will *not* take responsibility for zero or later usage. It is up to them to be ready with the required quantity, if they want to be your supplier. It is up to them to sell the products elsewhere, if necessary.
- ✓ Insure yourself against the possibility of loss due to defined circumstances
- ✓ Use standard market commodities, rather than tailored ones, so that you do not have to pay for a tailoring which the supplier cannot sell to others.

Here are some Reactive Tactics (to reduce damage):

- ✓ Cancel, or modify the order, as soon as it becomes apparent you do not need it.
- ✓ Re-deploy to other projects or organizations.
- ✓ Avoid committing to heavy financial expenditure until the *last possible moment*, within a long backroom activity. Do not *assume* you are going to go through with a particular step.

✓ Make sure *early feedback* about, for example 'user and market reaction to early steps' is *fed immediately* into the Backroom management process. Make sure rapid *action* is taken to reduce commitment and exposure.

> ***Better Match to Customer Need and Market Requirements.*** *The explicit customer feedback loop of Evolutionary Development results in the delivery of products that better meet the customers' need. The waterfall life cycle provides an investigation or definition phase for eliciting customer needs through focus groups and storyboards, but it does not provide a mechanism for continual validation and refinement of customer needs throughout the long implementation phase. Many customers find it difficult to articulate the full range of what they want from a product until they have actually used the product. Their needs and expectations evolve as they gain experience with the product. Evolutionary Development addresses this by incorporating customer feedback early and often during the implementation phase. The small implementation cycles allow the development team to respond to customer feedback by modifying the plans for future implementation cycles. Existing functionality can be changed, while planned functionality can be redefined. [COTTON96]*

## *Platform Development: An advanced industrial HP backroom concept.*

In his article [JANDOUREK96] 'A Model for Platform Development', Emil Jandourek of Hewlett Packard describes a way of organizing development in an industrial setting where many products are derived from a common architecture base. This is applicable to industrial engineering and is not for immature organizations, he stresses. But it is clearly a type of 'backroom activity'. HP applies this to software and firmware.

I am seeking permission to reprint his article or a revision of it entirely as an appendix to this book, but in the meantime and in any case let me bring out some highlights.

### The Purpose of Platform Development.

The purpose of 'Platform' is primarily to reduce time to market, and secondarily to reduce product development cost. The primary means is to have a well-organized product architecture from which product variations can be derived. The point is reuse and avoiding duplication of effort.

### The Basic Principles of Platform Development.

HP has developed a flexible basic model for how Platform works in their organization. They evolve this model continuously (Evo applied to an organizational development). They spread this model to receptive parts of the company through training and consulting.

The model results in a range of "10% to nearly 90% reuse of code or development effort" (Ibid. p.59).

The essence of Platform is "to pull out those product elements, features and subsystems that are stable and well-understood, and that provide a basis for value-added, differentiating features." (Ibid. p.59)

| **Investigation** | | | | *Implementation [← Evo Step for Construction →]* | | | | | *← iterate & release* | Final Release |
|---|---|---|---|---|---|---|---|---|---|---|
| **Platform Requirements Definition** | **Feasibility Validation** | **Architecture Definition** | **Platform Development Plan** | Infra-structure Development | *Plan* | *Design* | *Implement* | *Test* | *Platform Integration Test (as needed)* | |

Figure: The sequence for Platform development, after JANDOUREK96.

The above figure is the Evo development cycle at the Platform (architecture) level. The 'Investigation' component and the 'Infrastructure Development' activity is the 'head', and the 'implementation' component is the 'body' of an Evo process.

| **Platform Investigation** | | | | *Platform Implementation [← Evo Step for Construction →]* | | | | | *← iterate & release* | Final Release |
|---|---|---|---|---|---|---|---|---|---|---|
| **Platform Requirements Definition** | **Feasibility Validation** | **Architecture Definition** | **Platform Development Plan** | Infra-structure | *Plan* | *Design* | *Implement* | *Test* | *Platform Integration Test (as needed)* | |

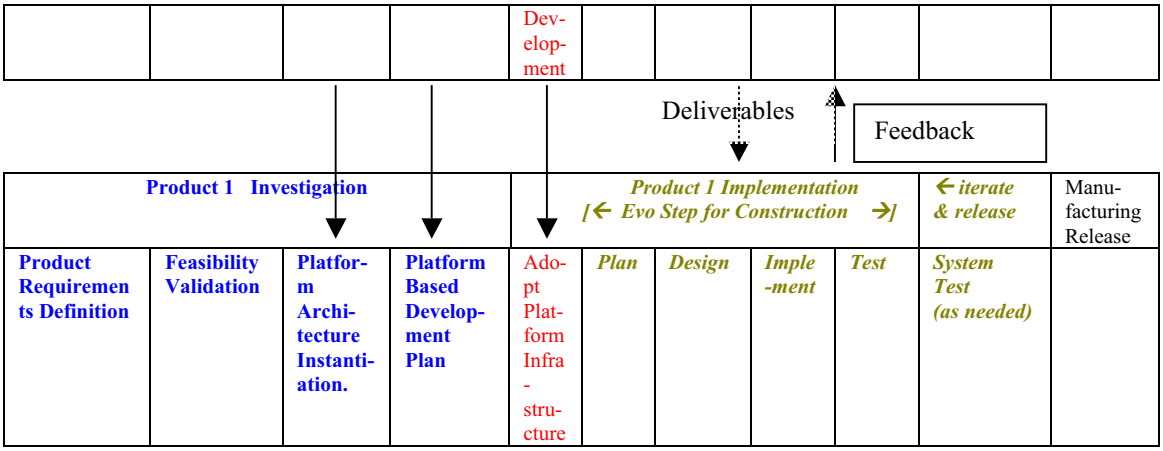



Figure: The first product derivative from the Platform Architecture, also uses an Evo cycle. After JANDOUREK96.

In the figure above we see the Platform development is the 'Backroom' for the first product development.

> ***"Infrastructure development:***
> *A significant amount of development environment infrastructure must be put in place during the first few implementation cycles.*
> *The tools that will be used…, as well as the processes that are adopted, can be developed in an evolutionary fashion in parallel with the functionality intended for the user. Some teams have found it valuable to make the infrastructure tasks an explicit category in the plan for each implementation cycle." [COTTON96]*

### The 16 Basic Elements of Platform Development.

The reported evolution of the Platform Development Model has 16 stable components:

1. Product Portfolio Planning
2. Architecture Definition and Partitioning
3. Product Feature Mapping
4. Test Architecture and Strategy
5. Organizational Structure and Work Partitioning
6. Partnership Model and Contract
7. Management Processes and Steering Teams
8. Communication and Feedback Model
9. Support Model
10. Platform and Product Life Cycles
11. Development Model and Development Process
12. Delivery Model
13. Validation and Test Processes
14. Development Tools and Infrastructure
15. Metrics and Measurement Processes
16. Values and Reward System

Only the full article can do justice to these concepts, but hopefully the list makes it clear that Platform is a widely embracing concept.

| Architectural Elements | | | Platform Management Elements | | |
|---|---|---|---|---|---|
| | Architectural Definitions and Partitioning | Product Feature Mapping | Organizational Structure and Work Partitioning | Partnership Model and Contract | |

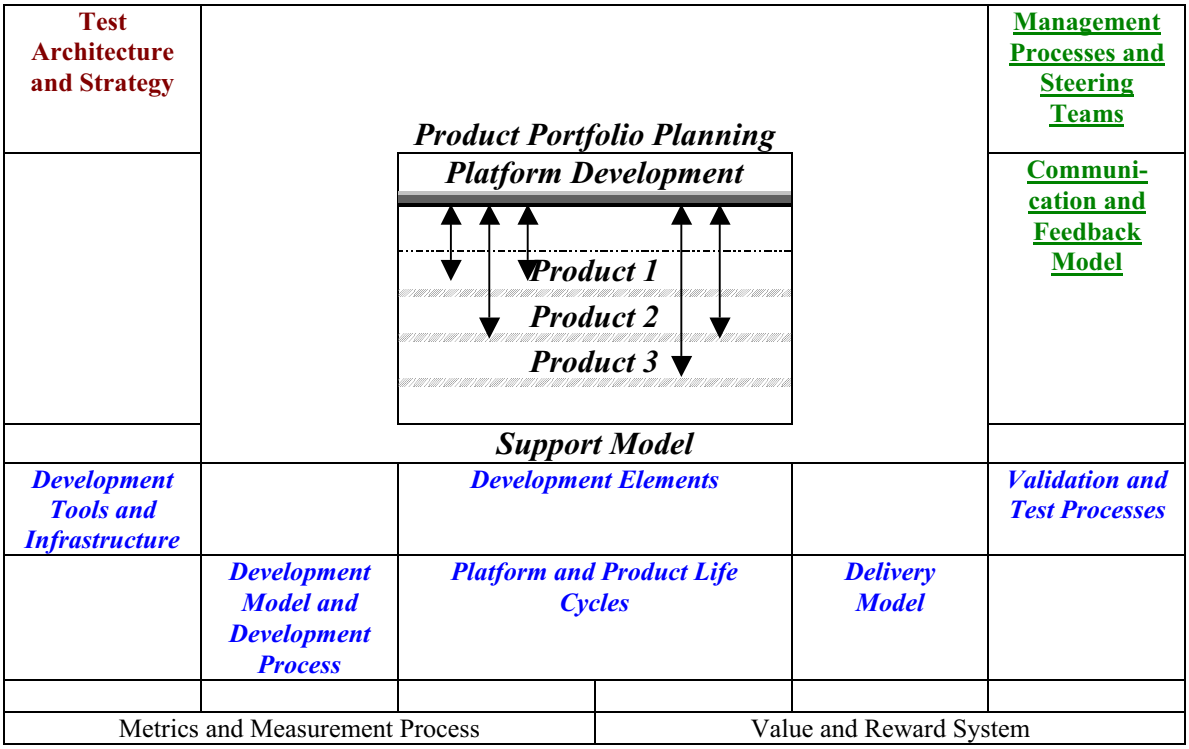



Figure: Major Elements of the platform development model, after JANDOUREK96.

### Application of Platform to Hardware

*Finally regarding your questions, yes synchronization with hardware is an issue ... and we normally address that. Although my article focuses on software almost all of the concepts map across to systems products ... we are also successfully applying them to hardware/firmware products.* ← *Emil Jandourek HP in Email reply to Gilb, July 97*

*"Hardware Projects. [Evo] should be considered for hardware development projects that test evolving ideas, have a modular hardware design that can operate when partially implemented, or need to implement a sequence of progressively more capable products, each of which is a workable, [replicable] system." [SPUCK93, 9.0]*

## *Chapter Summary: 8: The Evo Backroom: Readying components for packaging and delivery.*

For large and complex evolutionary projects, the project team can prepare selected design ideas and steps separately from the customer-interface delivery cycle. When they are ready in this Backroom, the Evo step management process, the Frontroom, can select them for acquisition and finally, deployment to the host system and end user.

## 9: Evo Culture Change

Evolutionary project management is for most companies 'different' from they way they run projects at present. Change, however well justified, always brings with it a certain collection of change problems. Evo is no exception. The process takes time and effort. One saving grace, of Evo, compared to many other techno-cultural change processes, is that the rewards are early and tangible.

> *"While the benefits can be substantial, implementation of evolutionary development can hold significant challenges. It requires a fundamental shift in the way one thinks about managing projects and definitely requires more management effort than traditional software development methods." [MAY96]*

### *Resistance.*

People are well aware that project management has problems. Things are 'normally' late, over budget, and disappointing [Morris94]. But, in spite of this, they are not embracing Evo, either.

The main reason why Evo is not popular yet, is not 'resistance based on experience'. It is a result of an almost complete lack of information about the *existence* of the method. It is a resistance based not on bad experiences, but on *no* experiences.

Well, the world needs Evo badly, and is waking up to it. This book is another step in the direction of Evo as the 'normal' project management method. The process will undoubtedly take decades.

> *Getting started. At HP*
> *"First Attempts.*
> *The first project was undertaken at HP's Manufacturing Test Division. The project (called project A here) consumed the time of four software developers for a year and a half and eventually was made up of over 120,000 lines of C and C++ code.*
> *Over 30 versions were produced during the eleven-month implementation phase which occurred in one- and two-week delivery cycles. The primary goals in using Evo were to reduce the number of late changes to the user interface and to reduce the number of defects found during system testing.*
> *Project A adapted Gilb's Evo methods.1 One departure was the use of surrogate users. The Manufacturing Test Division produces testers that are used in manufacturing environments. If the tester goes down, the manufacturer cannot ship products. Beta sites, even when customers agree to them, are carefully isolated from production use, so the beta software is rarely, if ever, exercised. Fortunately, the project had access to a group of surrogate users: application engineers in marketing and test engineers in their own manufacturing department. The use of surrogates did not appear to have any negative impact. About two thirds of the way through the project, the rigorous testing and defect fixing that had been done during the Evo cycles was discontinued because of schedule pressures. The cost of this decision was quality. With all efforts focused on finishing, developers began adding code at a rate double that of previous months, and over half of the critical and serious defects were introduced into the code in the last third of the project schedule.*
> *Even though Evo was not used to complete the project, the product was successful and the team attributed several positive results to having used the Evo method for the majority of the project. First, Evo contributed to creating better teamwork with users and more time to think of alternative solutions. Second, the project still had significantly fewer critical and serious defects during system testing. Third, the team was surprised to see an increase in productivity (measured in [non-commentary lines of code] per engineer-month). The project manager attributes this higher productivity primarily to increased focus on project goals." [MAY96]*

### *Evo is a form of process improvement at the project level.*

But, what is learned can be transferred to process and organization level.

> *"Short, frequent Evo cycles have some distinct advantages for internal processes and people considerations. First, continuous process improvement becomes a more realistic possibility with one-to-four-week cycles." [MAY96]*

### *How is the Evo project process improvement transferred to the larger organization?*

- ✓ People wander.
- ✓ Formal Process Descriptions.

- ✓ Training Upgraded
- ✓ Legend and Documentation of the Case

*Example: Om att Lyckas (On Succeeding by Jack Jrvik& Lars Kylberg, Ericsson, Published in English December 1994 by Stellan Nennerfelt, Ericsson internal publication EN/LZT 123 1987, Swedish Version LZT 123 1987 ) study Ericsson (ask Stellan Nennerfelt for permission to reproduce and get electronic English version). Request sent July 97.*

**At HP Evo process improvement are spread from project to organization :**

- ✓ Via the Platform process
- ✓ Using the Corporate Consultants and Trainers
- ✓ Using HP Journal articles
- ✓ Using Internal publications

## *Key Roles:*

Here are some lists of roles and tasks key players on the Evo team can be expected to play, courtesy of Todd Cotton, HP.

*"For the development process to progress in a smooth and efficient manner, it is helpful to define and assign ownership for three key roles: project manager, technical lead, and user liaison. On larger project teams, these roles may be shared by more than one person. On smaller teams, a person may play more than one role." COTTON96*

**Project Manager tasks under Evo: ←COTTON96**

Many aspects of the project manager's role become even more critical with Evo:

- ✓ Work with marketing team on requirements
- ✓ Work with customers on requirements
- ✓ Keep decision-making focussed on meeting requirements
- ✓ Identify key project risks, so as to address them in early steps
- ✓ Document all commitments and dependencies, to ensure proper sequencing
- ✓ Solicit and address any concerns the team has about Evo

*"In any case, the first delivery should be released in no more than two Evo cycles from the start of implementation. The first reason for this is that many of the concerns developers have with evolutionary development are best addressed by doing Evo. Second, if the start of Evo deliveries is delayed too long, the risk that delivery will never happen (until manufacturing release) is increased.' [MAY96]*

- ✓ Articulate how Evo will contribute to the project's success
- ✓ Define and manage the decision-making process to be more explicit, faster
- ✓ Invest in evolution of the architecture where appropriate.

*"There is also valid concern about adopting a new method at the beginning of the development process. Few teams are willing to make a full commitment to a new method when they have little experience with it. There may even be organizational changes anticipated if the organization is looking for large-scale productivity gains through formalized reuse.*
*Development teams and managers want some way to manage the risks associated with making so many simultaneous changes to their development environment. Evo can help manage the risks. The repeating cycles during the implementation phase provide for continual review and refinement of each parameter of the development environment. Any aspect of the development environment can be dropped, modified, or strengthened to provide the maximum benefit to the team." [COTTON96]*

**Technical Manager tasks under Evo: ←COTTON96**

- ✓ Responsible for architecture management
- ✓ Tracking and resolving technical issues
- ✓ Key role defining detailed task plans for each Evo cycle (feasibility, impact)

*"Finally, the inevitable change in expectations when users begin using the software system is addressed by Evo's early and ongoing involvement of the user in the development process. This can result in a product that better fits user needs and market requirements." [MAY96]*

**User liaison tasks under Evo: ←COTTON96**

- ✓ Manage teams interaction with users
- ✓ Set up user feedback process

> *First, users tend to focus on what they don't like, not what they do like. To keep this from being discouraging for the developers, it might help to provide a standard feedback form that elicits "Things I liked" followed by "Things I didn't like." [MAY96]*

- ✓ Define expectations of users
- ✓ Locate and qualify users
- ✓ Co-ordinate initial user training
- ✓ Collect user feedback
- ✓ Track user participation and satisfaction
- ✓ Inform users about development team's response to their feedback

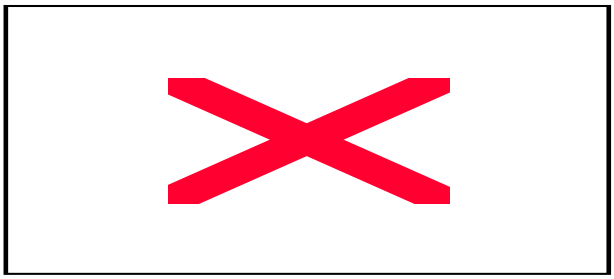


Fig. Amount of user feedback during (a) the conventional development process and (b) the evolutionary development process (Evo). [MAY96]

## *Motivation.*

> *"Some of the gains in productivity seen by project teams using Evo have been attributed to higher engineer motivation. The long implementation phase of the waterfall life cycle is often characterized by large variations in engineer motivation. It is difficult for engineers to maintain peak productivity when it may be months before they can integrate their work with that of others to see real results. Engineer motivation can take an even greater hit when the tyranny of the release date prohibits all but the most trivial responses to customer feedback received during the final stages of system test." COTTON96.*

> *"Because this approach to development may be new to the team, it is extremely important from a motivational perspective that these first few implementation cycles be successful." Todd Cotton, HP*

> *"A technique used widely within Hewlett-Packard is to adopt a naming scheme for the implementation cycles. One team used the names of wineries [alphabetically] from their local Northern California region. As they completed each cycle, their project manager would buy a bottle of wine from that winery and store it away. Once several cycles were completed, the team would celebrate by taking the wine to a fine restaurant for lunch." COTTON96*

> *"…the opportunity to show their work to customers and hear customer responses tends to increase the motivation of software developers and consequently encourages a more customer-focused orientation. In traditional software projects, that customer-response payoff may only come every few years and may be so filtered by marketing and management that it is meaningless." [MAY96}*

> *"Finally, the cooperation and flexibility required by Evo of each developer results in greater teamwork. Since scheduling and dependency analysis are more rigorous, less dead time is spent waiting on other people to complete their work." [MAY96]*

## *How can Evo spread lessons to parallel projects and later projects?*

<to be written, suggestions welcome!>
<success breeds desire, continuous process improvement, establishing policies, reward system based on meaningful delivery of results>

## *Critical Success Factors*

**Here are the critical success factors according to HP [MAY96]**

- ✓ Clear vision
- ✓ Project planning
- ✓ Fill key organizational roles
- ✓ Manage the developers
- ✓ Select and manage users
- ✓ Shift management focus
- ✓ Manage builds
- ✓ Focus on key objectives

> *"Microsoft people manage the evolution of products, projects and the overall organization with a remarkable minimum of politics and bureaucracy" MS, page 401*

## Some detail about the critical factors: (based partly on MAY96)

**Clear vision**

The requirements need to be clearly stated. Evo is all about moving towards fulfillment of those requirements. If they are unclear or incomplete, then Evo will be a train on the wrong track.

**Project planning**

You need to plan so that project teams get rapid feedback on their deliveries. You need to make sure that useful deliveries are really being made to users who can and will give feedback. You need to make sure of these two things early in the project. You need to make sure the project team gets positive practical experience early with Evo. You need to plan for:

- ✓ The project team
- ✓ Step users
- ✓ The group who will make decisions about feedback from deliveries

> *"The big secret that we did when we finally started shipping things on time [is that] we finally put time in the schedule. Not the 'lazy and stupid' buffer, but a sufficient amount of time [20%-50%] for unexpected things to happen. Once you admit you don't know everything about the future, it's like an alcoholic that says, 'Yes, I have a problem.'. Once you admit you have the problem that you can't predict the future perfectly, then you put time in the schedule, and it's actually quite easy to ship on time – if you're hard core on cutting features or scaling back features." Chris Peters, Microsoft MS, page 204-5*

You need to make sure the cycles are short enough so that the project team is motivated to make changes in response to customer feedback.

> *"Most users would like to have some kind of response to their requirements while they are still in the position they occupied when they stated their needs" [SPUCK93, 7.5]*

**Fill key organizational roles**

In addition to a project leader, you need a technical manager function and a user liaison function.

The technical manager resolves the daily tactical issues and handles co-ordination activities. The user liaison trains users, collects their feedback and communicates changes in the status of the project.

**Manage the developers**

Help the team understand why we are doing Evo. Address their concerns with the method.

> *"We believe, however, that Microsoft is distinctive to the degree to which it has introduced a structured concurrent and incremental approach to software product development that works for small as well as large-scale products. Furthermore we believe that Microsoft is a fascinating example of how culture and competitive strategy can drive product development and the innovation process. The Microsoft culture centers around freverently antibureaucratic PC programmers who do not like a lot of rules, structure or planning. Its competitive strategy revolves around identifying mass markets, quickly introducing products that are 'good enough' (rather than waiting until something is perfect), improving these products by incrementally evolving their features, and then selling multiple product versions and upgrades to customers around the world." MS, 15*

Help the project team avoid burnout caused by their own underestimates of what they can do in a cycle.

*"Although evolutionary development may seem intuitively obvious, implementing it in a traditional [development] life cycle environment should not be undertaken lightly. Much of the challenge has to do with managing people. The following steps have also contributed to success at HP:*

- *Establish a credible business reason for using Evo*
- *Discuss the method and the rationale for using Evo with the development team*
- *Ask for feedback*
- *Develop an initial plan that addresses as many concerns as possible*
- *Ask the development team to try Evo for a couple of releases and then evaluate future use.' [MAY96]*

**Select and manage users**

Get a user base as close to external customers as possible, even if you need to use in-house users to begin with. Make sure the user base is representative of your total population target for the product or system. Make sure they have realistic expectations with respect to the time they need to spend, the benefits they will get, the confidentiality of what they learn, and the possibility that they will experience teething troubles.

- Users state the requirements up front
- Users review specification and design documents
- Users prioritize delivery capabilities
- Users provide detailed requirements to implementers
- Users review implementation in progress
- Users assist implementers in detailed cost/capability tradeoffs
    - * Work is partitioned to individual user interests
    - * Implementation is essentially 'build to cost'
- Users test the system
- Users accept the system
- Users operate (abuse) the system
- Users assess the system
- Users change the requirements based on operational experience

**"Users are involved" [SPUCK93]**

**Shift management focus**

Instead of spending 95% of effort on 'building a system', management must shift focus so that they are probably spending more like one third of their time getting feedback, and another third making decisions, the last third managing actual building of the system. The focus is on 'doing the right thing' more than doing (a possibly wrong thing) 'right'. Working smarter, not harder.

**Manage builds**

If a product increment is going to be used every two weeks [HP] or daily and 6-10 weeks [Microsoft builds, and milestones] or several months [JPL], then all aspects of it need to be ready for use. Integrated. Tested. Complete from a user (internal or external) point of view.

*"Fries described how the daily build process ensures team coordination. 'It's really important. We couldn't have this big a group without out that [daily build process] , because we are still cooperating. We're trying to work like a small team. But we're not a small team. And we need the work that other people are doing. We need the product to be basically working all the time too, or it's going to interfere with your area. You can't have a guy who is working on drawing break typing, so that no one can type, so they can't type in to get to the area that they're trying to work on.' " . Ed Fries Microsoft development manager for Word in MS, page 269*

| Process or Project | Early Deliveries | Middle Deliveries | Late Deliveries |
|---|---|---|---|
| Documentation | User requirements documents | Requirements, user and administrator documents | All documents |
| Testing | Operational tests | Functional and performance tests | Requirements verification tests |
| Tracing | Informal | Formal | PCA/FCA Physical and |

| | | | |
|---|---|---|---|
| | | | Functional Configuration Audits |
| Product assurance | Good practice suggestions | Process monitoring | Full inspection |
| System performance measures | Good engineering practice | Improvements | Formal verification |
| Configuration management | Requirements as built | Formal at system integration | All facets formal |

Progressive Formality [SPUCK93]

> *"By the final delivery, all required documents are complete and of high quality. Of course throughout the succession of deliveries, attention is given to capturing information as it becomes available. It makes little sense to set things aside in informal documents when the final formal documents are 'living', i.e., available and evolving." [SPUCK93]*

**Focus on key objectives**
Keep a management eye on the 'gaps' between currently updated requirements, and current achievements. The larger the gap, the higher the priority to close the gap in coming cycles.

"When to Select Evolutionary Development
- When inserting technology into ongoing operations
- When you want to get capability into user hands early
- When you automate manual or untried operations and want regular feedback or to try intermediate products

> *"Doing daily builds is just like the most painful thing in the world. But it is the greatest thing in the world, because you get instant feedback"*
> *Lou Perazzoli, software engineering manager for Windows NT, in MS, page 268*

- When most Research and Development is done; when implementation is mostly engineering
- When you want to keep user/sponsor interest/motivation high
- When you need better management control
  - Confirm organization is performing
  - Partition work in time, as well as subsystem domain
- When a system architecture can be selected early and with some confidence ….
- When you need to control response to evolving requirements and/or budgets."

[SPUCK93, 9.0 JPL]

## *Summary: Chapter 9: Evo Culture Change.*

<to be written>

This mini edition of the manuscript is missing large amounts of text covering
Glossary (over 70 pages, almost identical with Requirements-Driven Management Book)
Bibliography
Rules
Procedures
Principles
Cases
In all over 100 pages.

I left the table of contents as it was and regenerated the index

But the point was to give an easily sampleable guts of the book. If you want more you can have it ( it takes 35 minutes to upload on my PC but I hope to get it put on a web site, so ask) Tom