Chapter 10

Evo

There is only one move that really counts: the next one.
—Chess master Jose Capablanca

OVERVIEW

- ❑ Classification of Evo.
- ❑ Workproducts, roles, and practices.
- ❑ Demonstrate Planguage for Evo specifications.
- ❑ Common mistakes, adoption and process mixtures, strengths and weaknesses.

Evo (short for **Evolutionary Project Management**) is perhaps the oldest IID method with a significant agile and adaptive quality, first taking shape in the 1960s and then published in 1976. Evo emphasizes:

- ❑ short iterations, with evolutionary delivery each iteration
- ❑ evolutionary requirements and design
- ❑ adaptive client-driven or value-driven planning
- ❑ quantifiable measurements of value and progress
- ❑ defining all quality requirements with numeric measures
- ❑ optional use of a language, Planguage, for specifications

METHOD OVERVIEW

Classification

In terms of cycle and ceremony, Evo classification is illustrated in Figure 10.1. For average projects, a common length of a timeboxed iteration is one or two weeks.

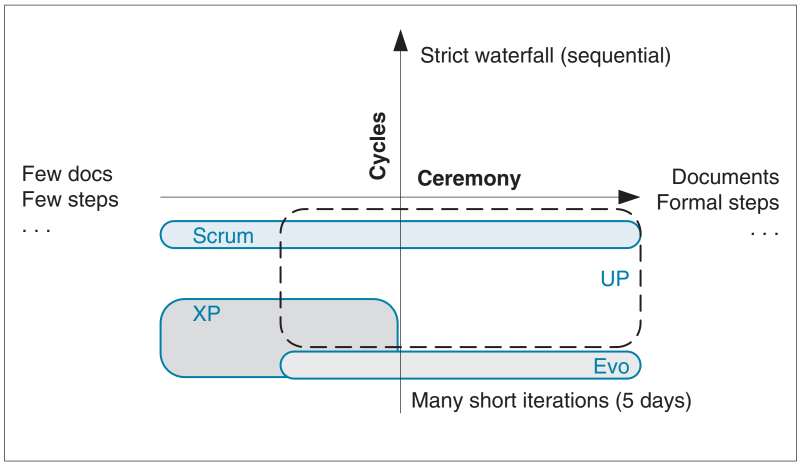


Figure 10.1 Evo on the cycles and ceremony scale.

Evo recommends some initial work to define a "critical top ten" list of measurable project objectives, and when specifications are written, Evo encourages unambiguous precision. It also encourages brevity, promoting one page summaries. Evo avoids big up-front specifications, although evolving specs—that could be part of a small or large set—are acceptable if shown to be valuable.

When describing high-level requirements, a structured language call **Planguage**[1] is possible; it encourages clarity, precision, and measurement. If used, it raises Evo on the ceremony scale.

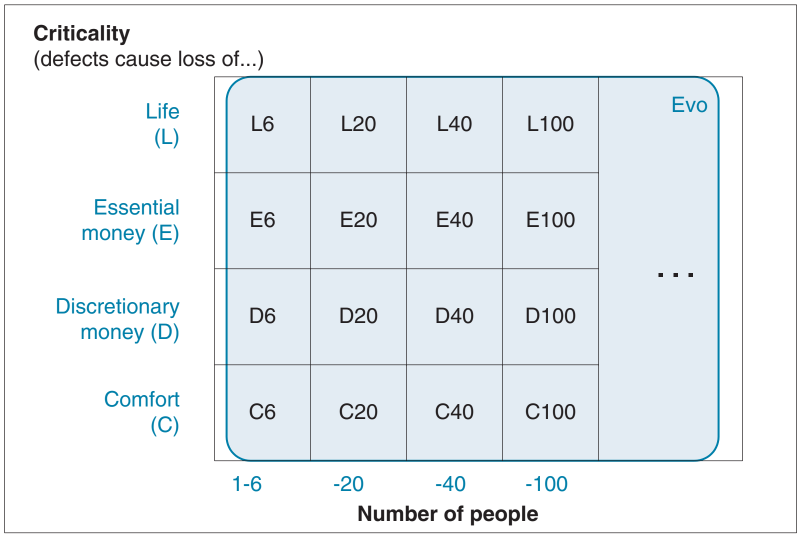
1. Rhymes with "language."

Similar to Scrum, it has only a small set of predefined workproducts, such as an **impact estimation table**. Others may be adopted from different methods as needed.

impact estimation table p. 235

In terms of the Cockburn scale, Evo covers the cells shown in Figure 10.2. Since the 1970s, it has been applied on a wide range of projects of many sizes.

Figure 10.2 Evo on the Cockburn scale



Introduction

Evo [Gilb76, Gilb88] was created by Tom Gilb, a pioneer of iterative and evolutionary development.

I'm including this chapter on Evo—less well known than Scrum, XP, and UP—not only because of its inherent interest, but to balance the historical oversight of this pioneering iterative method and to show that some agile method principles have long been part of Evo, such as a adaptive, client-driven planning of iterations.

Agile Principles p. 28

Gilb has been an advocate for an iterative, light, and adaptive approach to systems development since the 1960s; he first wrote about this in 1976, and his 1988 *Principles of Software Engineering Management* is a milestone early book presenting an evolutionary and iterative process.[2]

Evo's evolutionary emphasis is consistent with the Shewhart/ Deming cycle of Plan-Do-Study-Act (PDSA), and makes reference to PDSA as an underlying conceptual model.

Evo is not just for software. It is applicable in a larger systems engineering context—new software is just one solution to fulfill project objectives. For example, if more education (on the existing software) or operational change has a better value-to-cost ratio than new software, the former approaches are preferred.

It emphasizes—short iteration by iteration—making maximum progress towards the client's current highest-priority requirements, for the lowest cost. And each iteration, delivering into the hands of some stakeholders some useful results, so that early benefit and feedback is achieved. This is the practice of client-driven adaptive planning and evolutionary delivery.

Agile Manifesto p. 28

Evo is pragmatic, has some qualities similar to newer agile methods, is customer focused and results oriented—in the spirit of the Agile Manifesto and Principles. Anything necessary can change (based on the PDSA model) to reach the requirements (function or performance) within the project constraints.

these bold terms are official Evo terms

One of Evo's distinguishing ideas is its emphasis on clearly defining, quantifying, estimating, and measuring the **performance requirements** that need improvement over time.

2. Gilb also wrote the first book on software metrics, coining the term in [Gilb76], and continues to refine Evo, e.g., [Gilb03].

Performance includes **quality requirements** such as reliability, **workload capacity requirements** such as throughput, and **resource savings requirements** such as money. The impact of Evo steps on budgeted resource consumption is monitored both in design activity and iteration project management activity.

example requirements p. 233

Evo requires evaluating proposed solutions for their impact on the state of these requirements, and then actually measuring the impact of those introduced.

This structured approach and emphasis on improving the *performance* characteristics, rather than just on delivering functionality, is a key part of Evo's unique flavor.

Thus, note that in Evo there is explicit recognition that the requirements delivered may be either functions *or* performance objectives (quality, workload capacity, or resource saving).

Evo expects that each iteration there is a re-evaluation of solutions which yield the highest value to cost ratio, guided by feedback and estimates. As such, Evo requires active stakeholder participation to steer the project each iteration—client-driven adaptive planning. These practices are part of **evolutionary project management**.

adaptive planning p. 253

Measurable progress is a key principle of Evo, which takes seriously Drucker's maxim: *If you can't measure it, you can't manage it*. Quantifiable measures for performance requirements, and their regular measurement, is required. Unproven improvements, and vague quality goals such as "usable" are discouraged.

In Evo, the value system is that management doesn't schedule the details of the entire project, but they must be able to measure, control, and steer a dynamically evolving project. In other words, adaptive planning.

Planguage p. 231

Evo encourages *precision* and (where relevant) *quantification* in specifications. It does so by encouraging (but not requiring) the use of a compact, structured specification language called Planguage to record requirements—iteratively and incrementally.

It is a misunderstanding to interpret Evo's promotion of high-quality, low-volume critical specifications as an attempt at large up-front analysis. Evo promotes avoiding unnecessary analysis and detail—until it is needed.

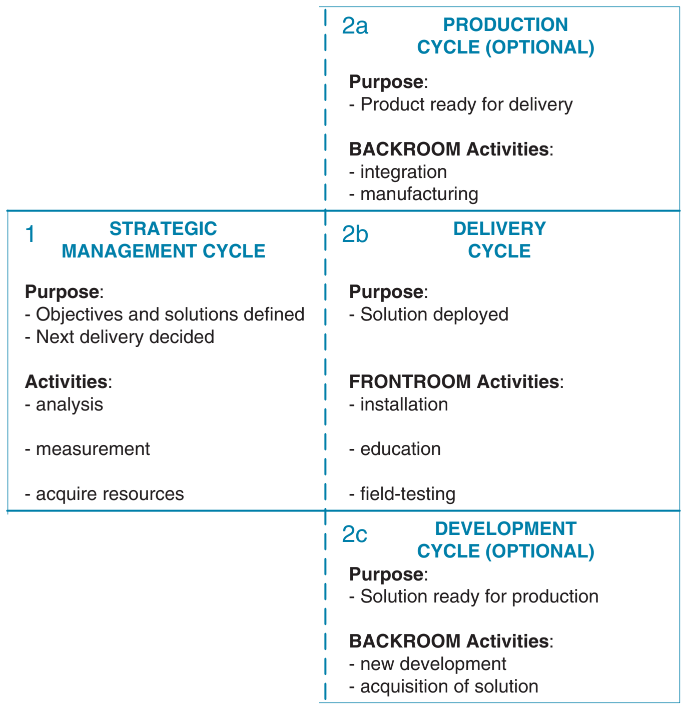
inspection p. 230

Inspection—especially of these specifications—is encouraged in Evo as an economical method to improve quality. Indeed, research verifies this [Russell91], and Gilb has been an active promoter of inspections for decades, including co-authoring the text *Software Inspections*.

Evo also encourages a risk-driven approach, as does the Unified Process. As Gilb has aptly said,

> *If you do not actively attack the risks in your project, they will actively attack you.*

LIFECYCLE



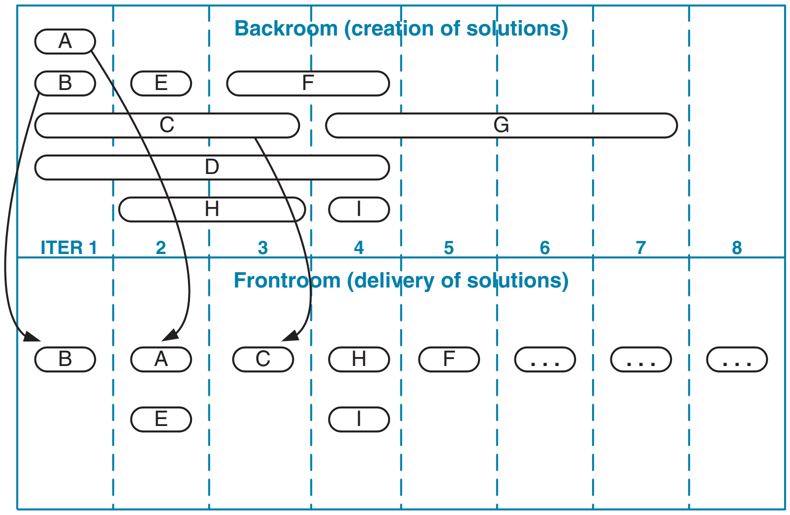
1. In the **Strategic Management** cycle, stakeholders decide which solutions ready for delivery (perhaps from the backroom activities) will actually be delivered, usually based on highest value-to-cost and risk. This activity also includes approving changes to objectives and solutions, analyzing feedback measurements, and obtaining resources.

2. These cycles may be concurrent. Ideally, each week something is delivered to stakeholders for use and feedback. In parallel, timeboxed development and production cycles work on incrementally building solutions ready for delivery, although it may be weeks (or longer) before they are eligible

for delivery. The analogy Evo offers is a business with the following organization:

- **Backroom**—products are prepared, and when ready, are "placed on a delivery shelf" available for delivery.
- **Frontroom**—some eligible products are taken off the shelf and delivered to stakeholders (see Figure 10.3).

Figure 10.3 backroom and frontroom delivery


Backroom (creation of solutions)
A
B
E
F
C
G
D
H
I
ITER 1
2
3
4
5
6
7
8
Frontroom (delivery of solutions)
B
A
C
H
F
. . .
. . .
. . .
E
I

Projects carry on, driven by the goal of maximizing stakeholder value at lowest cost, until there are no more profitable requirements to fulfill.

Niels Malotaux, another Evo consultant, describes the lifecycle of Evo projects from his experience working with clients [Malotaux03]:

1. A project kick-off "Evo Day" that includes the project manager, architect, and all other development team members. Activities include presenting an overview of Evo ideas and practices, explaining the product vision and architectural

ideas, identifying and estimating tasks for the first two-week iteration, and prioritization. Finally, people choose and commit to a set of individual tasks for the next week.

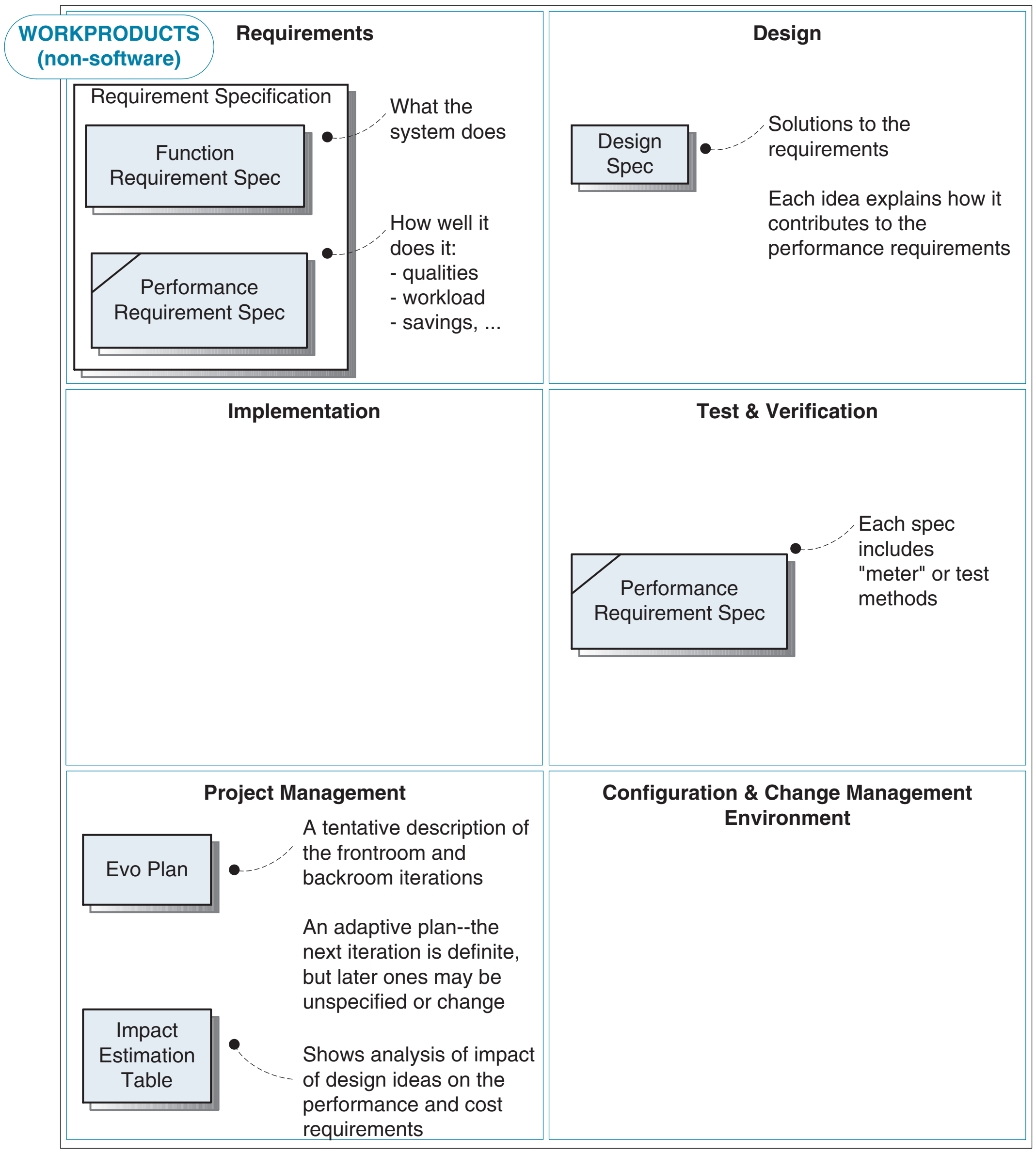
2. Execution of the two-week iteration.[3]

3. On the last day of the iteration:

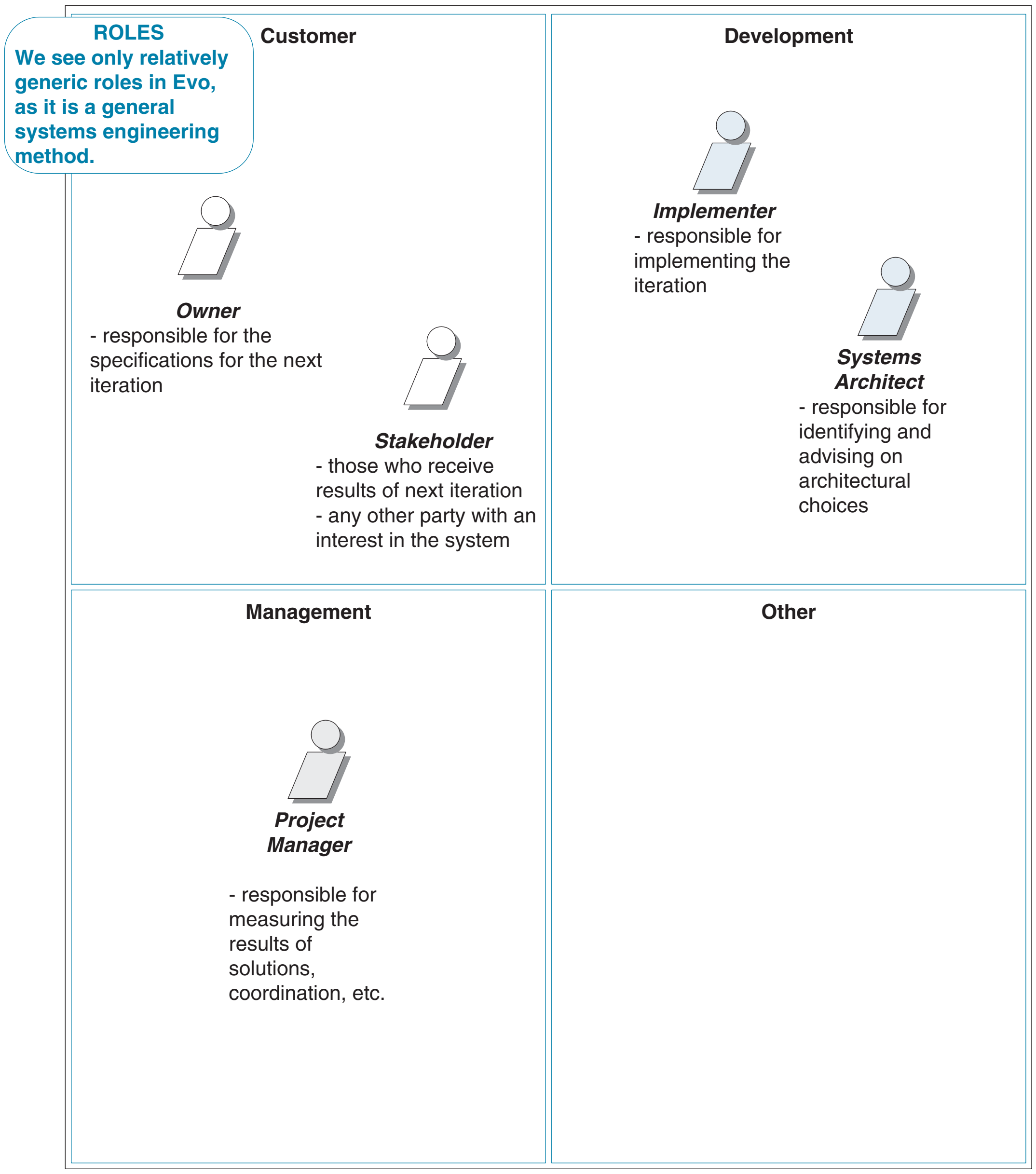
   - First, the project manager visits each developer and discusses the task results and completion. If things were not completed, there is reflection on the causes.
   - Second, the project manager discusses the project status with stakeholders (e.g., the product manager). Requirements are revisited and re-prioritized. Those chosen for the next iteration are analyzed and specified in greater detail, with measurements and so forth.
   - Third, the project manager and development team generate a new set of tasks. Again, developers choose and commit to the highest-priority tasks for the next week. In a team meeting, experiences of the last iteration may be discussed for process improvement ideas, and the product vision and evolving architecture may again be summarized or refined, to promote a common team goal.

3. Malotaux has found that two-week delivery iterations are more sustainable than one-week delivery iterations.

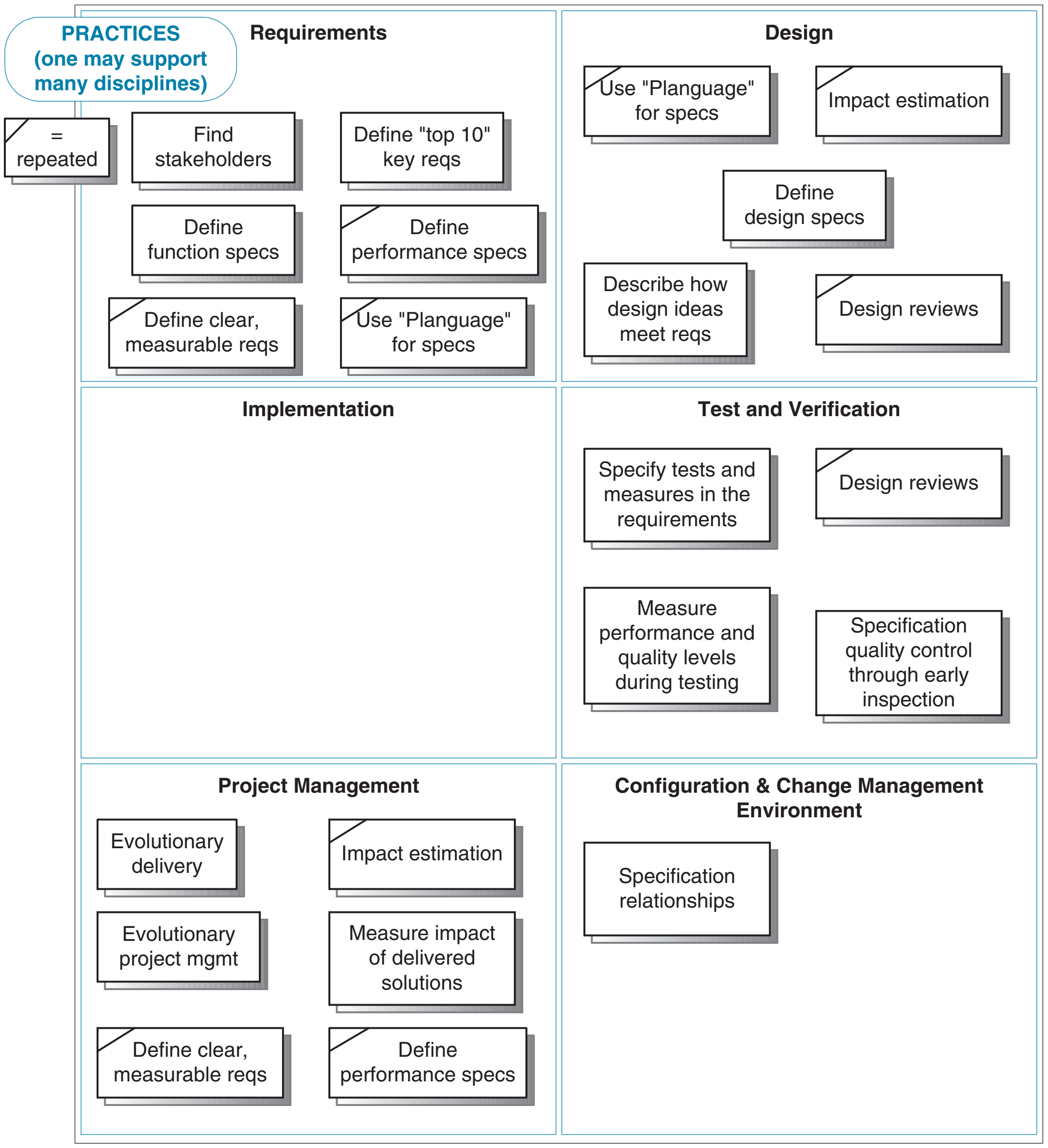
Workproducts, Roles, and Practices


WORKPRODUCTS
(non-software)
Requirements
Requirement Specification
Function Requirement Spec
Performance Requirement Spec
What the system does
How well it does it:
- qualities
- workload
- savings, ...
Design
Design Spec
Solutions to the requirements
Each idea explains how it contributes to the performance requirements
Implementation
Test & Verification
Performance Requirement Spec
Each spec includes "meter" or test methods
Project Management
Evo Plan
A tentative description of the frontroom and backroom iterations
An adaptive plan--the next iteration is definite, but later ones may be unspecified or change
Impact Estimation Table
Shows analysis of impact of design ideas on the performance and cost requirements
Configuration & Change Management Environment

Roles


ROLES
We see only relatively generic roles in Evo, as it is a general systems engineering method.
Customer
Owner
- responsible for the specifications for the next iteration
Stakeholder
- those who receive results of next iteration
- any other party with an interest in the system
Development
Implementer
- responsible for implementing the iteration
Systems Architect
- responsible for identifying and advising on architectural choices
Management
Project Manager
- responsible for measuring the results of solutions, coordination, etc.
Other

Practices


PRACTICES (one may support many disciplines)
= repeated
Requirements
Find stakeholders
Define "top 10" key reqs
Define function specs
Define performance specs
Define clear, measurable reqs
Use "Planguage" for specs
Design
Use "Planguage" for specs
Impact estimation
Define design specs
Describe how design ideas meet reqs
Design reviews
Implementation
Test and Verification
Specify tests and measures in the requirements
Design reviews
Measure performance and quality levels during testing
Specification quality control through early inspection
Project Management
Evolutionary delivery
Impact estimation
Evolutionary project mgmt
Measure impact of delivered solutions
Define clear, measurable reqs
Define performance specs
Configuration & Change Management Environment
Specification relationships

Core Practices

Evo applies to systems engineering in general—not only software development—although software projects are a common domain of application.

Requirements Practices

As with the other IID methods covered, Evo promotes evolutionary requirements analysis. Yet, when requirements and design ideas are written, Evo requires analysis with respect to a measurable evaluation of the value and impact of requirements and designs. Evo is infused with the practice and value of *measurable*, *measuring*, and *adaptive response* to the results.

Practice	Description
Find stakeholders	Both internal and external, friendly and foe, and across the lifecycle of the system.
Define "top 10" key reqs	Evo, as with other IID methods, encourages an early definition (in Planguage) of "critical top ten" *high-level* requirements. They need not all be decomposed into fine details, although those facing early implementation may be. Each iteration, they are reviewed and refined.
Define function specs	Evo functions describe what the system does. Evo does not promote major up-front detailed functional requirements analysis, but it does require at least clear definitions for the next iteration, *optionally* described in the **Function Requirement Specification**, using Planguage. *example p. 232*

Practice	Description
Define performance specs	Evo promotes describing system performance—how well the system works, its benefits, and how it affects the environment. Written and refined incrementally. Performance attributes are attached to functions. Specifically, Evo performance attributes fall into three categories: 1) quality—how well it performs (usability, reliability, …), 2) workload capacity, and 3) resource savings. The Performance Requirement Specification captures this information using Planguage. *example p. 233*
Define clear, and (where possible) measurable, specs	When specifications are written, do so in a manner and language which exposes and minimizes misunderstanding or ambiguity. The Evo Requirement Specification examples illustrate this. Evo promotes a balance between too little and too much detail in requirements. It wants clarity and detail for the key specifications you have chosen to implement in the next short iteration. Other more speculative or unassigned requirements can wait. Evo's performance specifications should have measurable impact, which should be identified. *examples p. 231*
Use Planguage for specs	Planguage is Evo's structured language for specifications in both requirements and design. It is optional, but encouraged. Evo includes **Planguage templates** for its requirements and design specifications. *notation p. 231, examples p. 231*

Project Management Practices

Practice	Description
Evolution-ary project manage-ment	Key ideas include: – evolutionary delivery to stakeholders for real use and feedback – small steps (ideally bi-weekly, or between 2–5% of total project financial cost and time) – steps with highest quality-to-cost ratios given highest priority for delivery – the existing system is preferred as the initial system base – feedback modifies future plans and requirements; adaptive planning and evolving specifications – total systems approach; do anything that helps – early results-orientation *discussion p. 227*
Evolution-ary deliv-ery	Evolutionary delivery emphasizes delivering a partial solution into production early, in order to obtain early business value, and feedback to guide and evolve future deliverables. A common delivery frequency in Evo is weekly, or more specifically, every 2–5% of duration and budget. In Evo, the solution chosen for delivery in the next iteration is based on highest value-to-cost ratio and early risk reduction. Each iteration's solution can be of a different type. For example, within a project to replace an older mainframe payroll application, early deliverables could be quick-fix operational changes in the existing system, or adding a Web-based front end to the old system, while work on the new system is underway in the backroom. *discussion p. 227*

Practice	Description
Measure impact of delivered solutions	Evo embraces Shewhart and Deming's core principle of improvement: PDSA. Plus, Drucker's maxim that you can't manage what you can't measure. The *study* step requires measuring, each iteration, the effect of the solution on the objectives. This data is used to help drive evolutionary project management (*act* in response to *study*), iteration by iteration. *Evo plan table p. 229*

Design Practices

Practice	Description
Define design specs	Design ideas are also recorded in Planguage, in the **Design Specification**, and are incrementally evolved, as with requirements specs. *example p. 234*
Impact estimation	A method to numerically analyze and compare the effectiveness of design ideas to meet cost and performance requirements—the qualities, workload capacity, and resource savings. The results are expressed in an **Impact Estimation Table**. *example p. 235*
Describe how design ideas meet reqs	The design specifications in Evo should explain why and to what degree they fulfill the requirements. This information is used in impact estimation, and discourages "resume-driven design" in which over-engineered or unfocused designs arise that are not really pertinent to business goals and value. *example p. 234*

Test and Verification Practices

Practice	Description
Specify tests and measures in the reqs	The *study* step in Plan-Do-Study-Act step requires measurements or **meters**, in Evo terms. Although new meters can always be adopted, Evo recommends that during performance analysis, the meters for that performance attribute be defined, within the Performance Requirement Specification. *example p. 233*
Specification quality control through early inspection	When goals or specifications are written, research shows that defects and misunderstanding are likely. Research also shows that early inspection is a powerful, cheap tool to reduce those defects. Note that **specification defects** have a precise meaning in Evo: failure to observe a formal, written, required specification rule. Gilb is an expert in the effective use of inspection—which is *not* the same as informal review. The Evo quality control practice includes *sampling*, and application of the **Defect Detection Process**, and **Defect Prevention Process**. *details p. 230*

Configuration & Change Management

Practice	Description
Specification relationships	The Planguage specification templates contain relationship sections to support requirements traceability. *example p. 234*

Evolutionary Project Management

As with Scrum, XP and UP, Evo's project management philosophy is adaptive rather than predictive planning. And, as with the other

adaptive planning p. 253

methods, there is still attention to the long-term vision, objectives, and a robust architecture. Some controlling principles:

- ❑ **Financial Control**—An iteration should be between 2–5% of the total initial financial budget before delivering some measurable results. This excludes larger capital costs that must be incurred in an iteration, such as buying a server, as these are "backroom" expenses.
- ❑ **Deadline Control**—A delivery (or frontroom) iteration should be between 2–5% of total project time, with a lower-bound of one or two weeks. This leads to the official Evo rule of thumb of one-week iterations for a one-year project. The Evo consultant Niels Malotaux has found two-week delivery iterations are more sustainable than one week.
- ❑ **Value Control**—Choose design ideas for the next iteration that deliver the best stakeholder value for costs.

With these control guidelines, the next iteration is chosen in response to the latest measurements and evolving understanding of the requirements. A misstep that doesn't deliver expected value consumes no more than (say) 2% of resources.

Future iterations may be tentatively assigned to specific design ideas, and ordered with respect to dependencies, but Evo encourages only very light investment in this kind of predictive planning, as it is central to Evo to adapt the plan at each step.

impact estimation table p. 235

Unless there is a specific stakeholder request for the next iteration, Evo recommends the use of impact estimation table analysis to choose design ideas for the next step.

For tracking and adapting, Evo also recommends the use of an impact table to record the results of delivered solutions, and to indicate the steps of the Evo plan. See Table 10.1 for a simplified example after the first iteration.

Table 10.1 simplified Evo plan and results table

Target Requirements	Iteration 1 (plan, actual)	Iteration 2	Cumulative to date
Responsive Browsing	5%, 2%[a]	10%, __	2%
System Reliability	10%, 5%	20%, __	5%
Capital Costs	0%, 0%	5%, __	0%
Development Costs	2%, 2%	2%, __	2%

a. the percentage of the final target

the capitalization in Evo implies these are terms defined in Planguage elsewhere

Regarding evolutionary delivery: A common Evo project management question is, "If I'm making a new plane (for example), how can I deliver it for use by stakeholders in weekly increments?" Although evolutionary delivery of software is often possible—such as bi-weekly refinement to a Web site, or new updates which can be downloaded—this of course will not apply to new products with long development lead times. In this case, Evo's approach is to work on their development in the *backroom*. It could be months before something from the backroom is available for delivery. Meanwhile, Evo still requires that *something* of measurable value be delivered to stakeholders each *frontroom* iteration (e.g., every two weeks). For example, early documentation samples, improvements to the existing system or operational environment, and so forth.

backroom / frontroom p. 218

The last point underlines Evo's total systems approach: Do anything that helps. It is not limited to new software or hardware constuction. Gilb believes there is an expensive and risky tendency to avoid looking at the existing system (when there is one) for the desired improvements—sometimes due to technologists' delight in new technologies—and thus he promotes in Evo a preference for considering the existing system as the base for improvement.

Specification Quality Control (SQC) Through Early Inspection

When specifications are created (iteratively), Evo recommends the use of classic systems engineering process control through sampling and inspection. Evo promotes defect removal in specs, done with agility, through its Defect Detection Process and Defect Prevention Process.

Evo's SQC draws from IBM's research and practice [e.g., MJHS90], and Gilb's experience; he is co-author of *Software Inspection* (which emphasizes specification inspection).

A key idea in SQC is that specifications are not informally inspected for any kind of fault; rather, there is only a search for defects—meaning a violation of a written rule from a rule set or checklist that the "checker" is working against. Here's a simplified defect rule set[4]:

- ❑ **Clear**—They must be unambiguously clear to the intended readers.
- ❑ **Scale**—Performance and cost requirements must specify a scale of measure to define the concept.

Other key practices in SQC include:

- ❑ Two to five checkers for an inspection.
- ❑ Specification pages are *sampled* for inspection; the entire document is not checked. If the sampled defect level is above a threshold, the specification is not released for use.
- ❑ The checkers do not volunteer solution or correction advice to the author. They only note issues. It is up to the author to determine solutions or take the initiative to ask the checkers for suggestions.

4. Adapted from [Gilb03].

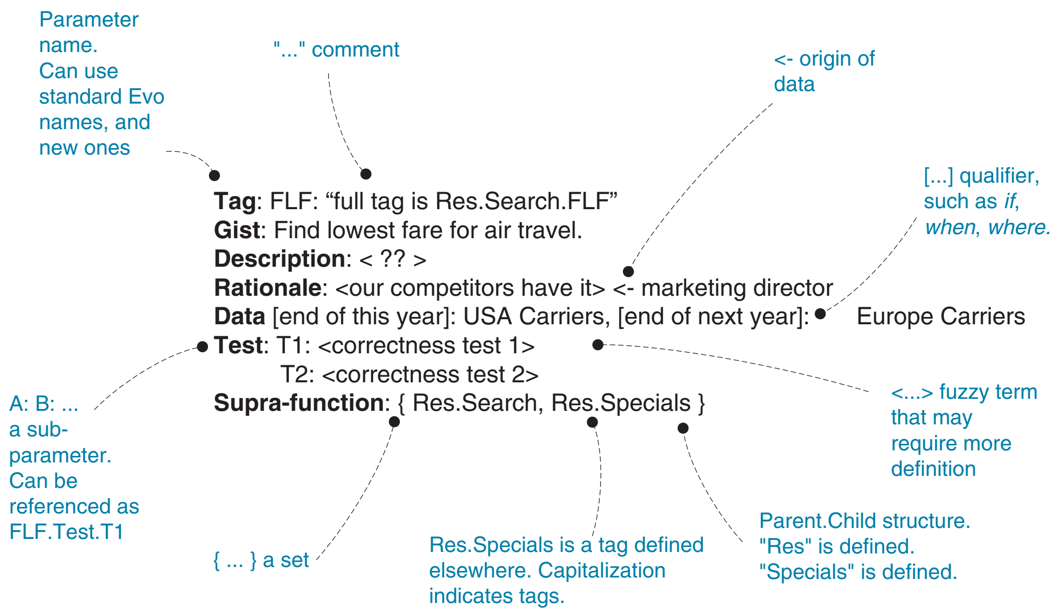
Defect *prevention* in Evo is a process improvement activity that comes from collecting inspection data, reflecting on the results, and experimenting with changes in source workproduct creation.

Planguage

Planguage is Evo's compact specification language. Figure 10.4 shows common notation for one partial specification.

examples: See "Workproducts" on p. 231.

Figure 10.4 Planguage


Parameter name. Can use standard Evo names, and new ones
"..." comment
<- origin of data
[...] qualifier, such as *if*, *when*, *where.*
Tag: FLF: "full tag is Res.Search.FLF"
Gist: Find lowest fare for air travel.
Description: < ?? >
Rationale: <our competitors have it> <- marketing director
Data [end of this year]: USA Carriers, [end of next year]: Europe Carriers
Test: T1: <correctness test 1>
T2: <correctness test 2>
Supra-function: { Res.Search, Res.Specials }
A: B: ... a sub-parameter. Can be referenced as FLF.Test.T1
<...> fuzzy term that may require more definition
{ ... } a set
Res.Specials is a tag defined elsewhere. Capitalization indicates tags.
Parent.Child structure. "Res" is defined. "Specials" is defined.

Workproducts

Full description of Evo's workproducts and how they can be expressed in Planguage is beyond the scope of this introduction. Nevertheless, the following examples provide a sample of Evo's flavor. More detailed examples are given than for the

Scrum, XP, and UP chapters, as Evo examples are less well-known and less widely available.

Planguage specifications are incrementally developed over the iterations, and only to the extent that doing so adds value.

Function Requirement Specification

Individual functions are recorded in an Evo Function Specification, using Planguage. These could be a high-level top-ten list of functions, or detailed and decomposed functions. The following example illustrates standard parameters (e.g., "Gist") from the Evo Planguage template. Some statements are purposefully undefined, both for brevity and to emphasize the normal process of partial and evolving specifications in Evo. All capitalized tag elements (e.g., Call Center) refer to other specifications previously defined, probably hyperlinked and clickable. Observe that opinions or "facts" in a specification are sourced to a party; Evo expects claims to have some substantiation, or at least explicit acknowledgment that they are wild guesses.

Planguage p. 231

Tag: FLF:
Type: Function Specification
======= Basic Information =================
"version, status, owner, stakeholders are elided"
Gist: Find lowest fare for air travel.
Description: <input: dates, airports, carriers. output: flights sorted by cost>

Relationships: Evo supports requirements traceability in this section.

============= Relationships ===============
Supra-functions: Res.Search
Sub-functions: none
Is Impacted By: { Call Center, Web Front End }
Linked To: Supports: Res.Booking

Measurement: Goals in Evo should be testable and measurable.

============= Measurement ===============
Test: T1: <correctness test 1>
============= Priority and Risk Management ==

Rationale: <Our competitors have it> <- Marketing Director
Assumptions:
A1 [before end of next year]: Competitor X doesn't upgrade
A2: < ?? >
Dependencies: Res.DB
Risks: R2, R6
Priority: Must be in first public release <- Marketing Director
============= Specific Budgets =============
Financial Budget: < ?? >

Performance Requirement Specification

Individual performance requirements (quality, workload capacity, resource saving) are recorded in the Planguage form shown in this example.

Tag: Responsive Browsing:

Type: Workload Capacity Requirement: Response:
Budget: < ?? >
============= Basic Information ========
"version, status, owner, stakeholders are elided"
Ambition: <Many> Res.Users with <acceptable> response time.
============= Measurement ===========
Scale: Average HTTP response time in seconds
Meter: Automated HTTP server monitor
============= Targets ===============
Goal
[First Release]: response under 3 seconds for up to 1,000 requests per second <- Marketing,
[Second Release]: response under 2 seconds for up to 1,000 requests per second <- Marketing
============= Constraints ===============
Fail [First Release]: response over 6 seconds <- Marketing
============= Benchmarks ===============

Measurement: Illustrating the quantifiable emphasis in Evo.

Past [Old System, last year]: response under 5 seconds for up to 1,000 requests per second <- ABC Research Report
Record [CompetitorY, this year]: response under 1 second for up to 3,000 requests per second <- ABC Research Report
============= Relationships ===============
Is Impacted By: Res.DB.Response <- DBA
Impacts: Usability
============= Priority and Risk Management ==
Value <this level will retain 95% of first-time users> <-Marketing
"assumptions, dependencies, etc."

Relationships: Illustrates the performance requirement relates to other performance requirements (or perhaps, directly to functions).

Design Specification

Design ideas are expressed in the Planguage template form demonstrated in this next example.

Tag: Server Cluster:
Type: Design Idea
============ Basic Information ==============
"version, status, owner, stakeholders are elided"
Gist: Cluster of 10 application servers with an IP sprayer.
Description: < ?? >
============= Design Relationships ===========
Design Constraints: { Use Moon Spark 5000s, Use Java Technologies, Use Open Source }
Sub-Designs: < replication, fail over >
============== Impacts Relationships ==========

Impacts: design ideas must be connected to functions and/or performance requirements

Impacts [Functions]: { Res.Search, Res.Transaction, Res.Browse }

Impacts [Intended]: { [Good] Responsive Browsing, [Good] System Reliability, <more> }

Impacts [Cost]: { Operations Budget, [if not open source] Development Budget }

Impacts [Other Designs]: { Deployment Model, Data Model }

Value: < meeting responsiveness and reliability goals will maintain customer retention at 95% <- Marketing Director >

== Impact Estimation of Design on Selected Requirements ==

Tag: Responsive Browsing

Type: Performance Requirement Cross Reference

Scale: Average HTTP response time in seconds

Scale Impact: under 3 seconds for up to 1,000 requests per second

Scale Uncertainty: ± 1 second <- Jill Jones

Percentage Impact: [if Use Moon Spark 5000s] 100%[5]

Percentage Uncertainty: ± 33%

Evidence: CompetitorX has this configuration and response

Source: Jill Jones (Chief Architect)

Credibility: 0.5 as Jill worked for CompetitorX on similar project

Impact Estimation: a design idea should contribute to performance objectives. Its impact on each is analyzed.

Note that claims are sourced, and uncertainty and credibility estimated.

Tag: System Reliability

"repeat analysis using the above set of parameters"

============== Priority and Risk Management ====

"assumptions, dependencies, risks, priority, issues are elided"

Impact Estimation Table

This tool is used in Evo to analyze the impact of alternative (or complementary) design ideas on performance requirements. Barring "obvious" priorities for the next iteration as indicated by stakeholders, this table is used to rationally choose the set of design ideas to implement next, based on the best benefit-to-cost ratio. Note that the horizontal and vertical summing of impact percentages do not always accurately predict a result; they may or may not provide a sense of aggregate impact. For example, can one sum the *Responsive Browsing* impact of both a server cluster and high-performance hardware? Perhaps…

5. From some baseline (such as "Past") in the requirement.

Table 10.2 simplified impact estimation table

Design Ideas -> Requirements	**Server Cluster**	**High-performance hardware**	Sum of Impact[a]
Responsive Browsing Baseline: 5 sec. Goal: 3 sec.			
Scale and % impact[b]	3 ± 1 sec. 100% ± 50	4 ± 1 sec. 50% ± 50	150% ± 100
Evidence and Credibility	CompetitorX has this configuration and response <- Jill Jones 0.2	Moon Microsystems has customers achieving this <- Moon Sys Eng. 0.1	
System Reliability Baseline: 3000 hours MTBF. Goal: 3500			
Scale and % impact	3200 ± 200. 40% ± 40	3100 ± 200. 20% ± 40	60% ± 80
Evidence and Credibility	CompetitorX has this config and “suspected” reliability <- Jill Jones 0.2	Moon Microsystems has customers achieving this <- Moon Sys Eng. 0.1	
Sum of Impact[c]	140%	70%	
Capital/Dev Cost Baseline: $0 USD. Budget: $200K			
Amount and %	$20K ± 10K. 10% ± 5	$100K ± 10K. 50% ± 5	60% ± 10
Evidence and Credibility	Bob’s friend guesses this cost on another project <- Bob Bones 0.1	Moon firm quote <- Moon Sales Rep. 1.0	
Benefit-to-Cost Ratio[c]	14 (140% / 10%)	1.4 (70% / 50%)	
Impact Credibility Adjust Cost Credibility Adjust	0.84 (14 * .3 *.2)[d] 0.08 (0.84 * .1)	0.01 (1.4 * .1* .1) 0.01 (0.01 * 1.0)	

a. Sum of impacts on a requirement may or may not be cumulative.
b. The % impacts are with respect to the baseline.
c. The sum of impacts of one design idea may or may not be cumulative. The total may or may not work as an estimate for comparison.
d. Multiplying probabilities is a heuristic to reduce total to a reasonable magnitude.

There is a lighter alternative (for prioritization) to these tables that Evo also offers: the use of simple benefit-cost estimates: Each design idea is given a 0-9 ranking for both benefit and cost. Ideally, this is in a group “delphi” ranking session. The best benefit-to-cost ratio ideas are implemented next.

Other Practices and Values

Evo has many detailed practices, tips, and guidelines. A sample of points:

- ❑ **Open-ended architecture**—To support evolving or changing designs, and evolutionary delivery, Evo encourages open-ended architectures that encourage easier extension. That is, at predictable variation points, some kind of protection is introduced, such as an interface, data-driven declarations, and so forth.
- ❑ **Safety factor**—The estimated impact of a design should deliver an estimated impact with a defined safety factor, default factor 2 (200% over the target level from the baseline).
- ❑ **Client-driven planning**—If you are uncertain which step to do next, ask your dominant stakeholder.
- ❑ **Whatever adds value**—Rather than a “we are building it” paradigm, focus on “what can I do for my stakeholders next week?” The techniques (such as Planguage and Impact Estimation) are only support to keep this focus, and should not get in the way.

Values

Evo’s key values include:

- ❑ Learn rapidly by realistic measurement.

- ❑ Deliver real value to stakeholders early, frequently, at every step.
- ❑ Be humble about complex systems: simplify and attack problems one small step at a time
- ❑ Delegate power to the ultimate user, by focusing on end results and not methods and well-intended bureaucracy.
- ❑ Admire, applaud and reward a team based on the flow of measurable results: stakeholder value versus costs.

Common Mistakes and Misunderstandings

or, How to Fail with Evo

Error: Adoption mistakes—Lack of management support. Lack of training in concepts and methods. Lack of clear quantified management objectives as the basis for evolving towards Evo methods. Lack of clarity about the management objectives of using the method — and how to measure these improvements in practice. Lack of a good successful pilot project to prove it works in your environment. Lack of dramatic motivation to change from older methods.

Error: Lack of focus on results—Self-explanatory.

Error: Giving up or not believing short iterations are possible—Giving up too easily when managers or engineers claim they cannot find small early steps (they need training, motivation and help). Giving up too early and falling back on old habits.

Error: Lack of management encouragement—When a team starts delivering something of value every short iteration, that's often a revolutionary event. Management needs to praise and encourage this result, rather than take it for granted.

Error: Failing to use value/cost priority—Not choosing solutions based on highest value-to-cost.

Error: Customers not involved—Evo is customer and results-driven; they need to participate in providing feedback on the results of evolutionary deliveries, and in steering the next iteration.

Error: No measurements—It is a mistake to avoid regular measurement of the impact of delivered solutions. Frequent numeric measurement is a significant shift for many managers, but central to Evo.

Error: Iterations too long—Evo frontroom iterations should be 2–5% of total project time, with a lower bound of one or two weeks.

Error: Each iteration does *not* end in a delivery—Evo is about evolutionary delivery on a "weekly" basis to real stakeholders for useful results, even when backroom development may take months.

Error: Predictive planning—It is a misunderstanding to create, *at the start of the project*, a believable plan laying out exactly how many iterations there will be for a long project, their lengths, and what will occur in each. This is contrasted with Evo or adaptive planning. The Evo team and customer plans the next iteration, and then planning adapts iteration by iteration, based on measurement and feedback.

Sample Projects

Gilb's view is that any project applying IID and evolutionary delivery is an example of Evo. This of course covers thousands of projects. For an early example, the mid-1970s LAMPS project described on p. 83 is considered an Evo project in Gilb's classification.

Process Mixtures

None of the other IID methods covered emphasize weekly evolutionary delivery, and related Evo project management measurement.

Evo + Scrum

Most Scrum practices are compatible with Evo. The Scrum meeting, common project room, and demos to external stakeholders at the end of each iteration enhance Evo's feedback goals. The Scrum backlog and progress tracking approaches are also applicable additions. Scrum does not discuss specific specification methods, and thus Evo's Planguage is still applicable.

Evo's measurement emphasis is compatible; indeed, Jeff Sutherland, one of the Scrum creators, takes a strong interest in measurement when applying Scrum.

Scrum's unchanging 30-day iteration length is not consistent with Evo—Evo iterations are usually shorter.

Evo + UP

The UP is especially for software development, and usually for projects involving multiple iterations before production delivery. Consequently, the UP could be applied to Evo backroom development work. However, Evo's evolutionary delivery and project management styles are not exactly in the same spirit as the UP, although both share an interest in early identification and mitigation of risks.

The UP has its own set of workproducts and approach to requirements capture: the Use-Case model (and thus, use cases), and Supplementary Specification for description of functions, features, and non-functional requirements. Evo Planguage elements, such as the Performance Requirement Specification, may be used within the UP Supplementary Specification.

Evo's measurement emphasis is compatible or acceptable with the UP.

The upper bound of UP's 2–6 week iteration length is not consistent with Evo—too long.

Evo + XP

XP values and spirit regarding specifications are not exactly compatible with Evo. XP's value of avoiding written or precise requirements, and preferring oral communication between developers and requirement donors is different than Evo's emphasis that when a specification is required, it be written with clarity and measurable qualities. However, Evo allows a scaling down of precision on small projects; the important Evo point is value to the client, and precision is an optional means to that end.

On the other hand, many XP development practices may be consistently applied with Evo, such as test-driven development, pair programming, and so forth.

XP's emphasis on early results and customer-driven adaptive planning is also consistent with Evo. The XP practice of stand-up meeting, common project room, and whole team together supports Evo's feedback goals.

XP's 1–3 week iteration length is consistent with Evo.

Adoption Strategies

As always, coaching by an experienced method expert on the first project is recommended. Evo is results oriented, so not much is sacred in its adoption—other than frequent evolutionary delivery and project management.

Clear, precise, and measurable (though evolutionary) requirements are not that common or enthusiastically developed. One approach to motivate their adoption is to focus early on evolutionary delivery, which of course demands understanding the design ideas, requirements, alternatives, and priorities. Thus, after a few iterations, the participants themselves will better appreciate the value in adopting something like Planguage and greater requirements precision, in order to guide choosing their next step and evaluating the results of the prior one.

Gilb recommends the use of pilot projects to demonstrate the value and viability of Evo.

Fact versus Fantasy

Impact estimation tables are not consistently used by Evo adopters. This may be due to their requiring more analysis and complexity than the priority problem often warrants. As mentioned, a less detailed 0–9 scale for benefit and costs ratios is an Evo alternative.

One-week evolutionary delivery iterations are difficult to sustain; two weeks is easier.

Gilb reports that a significant number of Evo adopters find quantification of their most critical objectives difficult without some coaching.

Evo’s PDSA emphasis requires not only estimation and planning, but measuring. Yet, this last step is often dropped under the pressure of work, which of course makes Evo planning less useful.

Strengths versus “Other”

Strengths

- ❑ Early, visible results; frequent delivery to stakeholders.
- ❑ Measuring the impact of solutions and guiding improvement by measurement data, rather than only by informal guess.
- ❑ Customer participation and steering.
- ❑ Worker engagement and satisfaction from seeing their solutions quickly implemented.
- ❑ Planguage is a simple and compact approach to requirements specification.
- ❑ Evolutionary and incremental requirements and development, and adaptive behavior.
- ❑ Emphasizes quality through proven inspection methods and through continual process improvement based on measurement and data.
- ❑ Practices from other methods (e.g., Scrum or XP) easily included.

Other[6]

- ❑ Management and requirements overhead of estimating impacts and measuring results.

6. Could be viewed as a weakness, strength, or deliberate desirable exclusion depending on point of view.

- ❑ As with Scrum, minimal guidance within software-specific disciplines, as Evo is a general project management and systems engineering method.

History

Gilb started some Evo practices in the early 1960s, while consulting (and living) primarily in Europe. In 1976, he wrote about iterative development, evolutionary delivery, and evolutionary project management in his book, *Software Metrics*. This was rather unique in a period dominated by waterfall lifecycle promotion. In the late 1970s, he authored a series of column articles in *Computer Weekly UK* that reiterated and further explored these practices; these articles are arguably the earliest popular press on the subject of IID and adaptive, evolutionary development.

In April 1981, Gilb published "Evolutionary Development" in *ACM Software Engineering Notes*, and in July 1985 published "Evolutionary Delivery versus the 'Waterfall Model'" *ACM Sigsoft Software Requirements Engineering Notes*. These are some of the earliest ACM or IEEE publications related to the subject of IID and adaptive, evolutionary development.

In the 1980s he was also exposed to the work of Deming, and realized that Deming's values and Shewhart's PDSA model captured the intent of Evo.

As mentioned in the introduction, in 1988 Gilb published *Principles of Software Engineering Management*, a milestone early book describing an adaptive, iterative, and evolutionary process, well ahead of its time.

Since then, his early work and Evo have influenced many other methods: XP, Scrum, and the UP all owe debts to Gilb's work. The popular book *Rapid Development* [McConnell96]—which examines many key best practices in software development—cites Gilb's work in 14 sections.

What's Next?

The next chapter examines some method practices in more detail, and introduces other common tips. The final chapter is a FAQ.

Recommended Readings

- ❑ Gilb's 1988 *Principles of Software Engineering Management* is an important step in studying Evo. His 2003 *Competitive Engineering* presents updated refinements, and the details of Planguage; it is the current basis for studying Evo.
- ❑ *Software Projects: Evolutionary versus Big-bang Delivery*, Felix Redmill, John Wiley & Sons, 1997. Redmill learned Evo from Gilb in the 1980s and managed projects with it. This book describes his experience and lessons learned.
- ❑ Free online articles and draft books by Gilb—on Evo subjects—are available at his Web site: *www.gilb.com*.
- ❑ Useful elaboration and refinements for Evo are also available for download from Niels Malotaux at *www.malotaux.nl*.

Supporting or related texts that are recommended include:

- ❑ *Software Inspection*, by Tom Gilb and Dorothy Graham.
- ❑ *Out of the Crisis*, by W. Edwards Deming.
- ❑ *The Deming Management Method*, by W. Edwards Deming and Mary Walton.
- ❑ *Quality Is Free: The Art of Making Quality Certain*, by Philip Crosby.