Chapter
1

Planguage Basics and Process Control

The Purpose of Planguage

Glossary Concepts

- Planguage
- Standards
- Rule
- Process
- Procedure
- Task

1.1 Introduction: Why We Need a Different 'Systems Engineering' Approach

As the rate of technological change has 'heated up,' I am sure we have all seen that, increasingly, nobody 'knows all the answers.' Previously we could rely on comparatively *stable* environments (technology, workforce, experienced people, politics and economics). People knew how to solve problems because they had solved similar ones before. In addition, the concept of learning by apprenticeship was valid; 'masters' could pass on their wisdom over a time span of years.

Things are currently moving so fast that it is dangerous to assume there is any first-hand knowledge of the technology we are going to use, or of the markets we are going to sell to. Even the organizational and social structures that we are targeting are constantly changing. Authors such as Tom Peters have long since clearly documented these trends and threats (Peters 1992).

So we have to find out 'what works now' by means of practice, not theory. We need to develop things in a different way. We have to learn and to change, faster than the competition.

The fundamental concepts needed now in systems engineering include:

Learning through Rapid Feedback

Feedback is the single most powerful concept for successful projects. Methods that use feedback are successful. Those that do not, seem to fail. Feedback helps you get better control of your project, by providing facts about how things are working in practice. Of course, the presumption is that the feedback is early enough to do some good. This is the main need: rapid feedback.

Dynamic Adaptability

Projects have to be able to respond to feedback and also to be able to keep pace weekly or monthly with changing business or organizational requirements. Projects must continuously monitor the relevance of their current work. Then they must modify their requirements and strategies accordingly. Any product or organizational system should be continuously evolving or it dies. Coping with external change *during* projects and adapting to it *during* projects is now the norm, not the exception. Stability would be nice, but we can't have it!

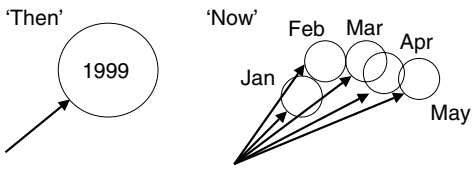


Figure 1.1
Our requirements are changing faster due to external changes.

Capturing the Requirements

It is true of any system that there are several *Critical Success Factors*. They include both performance requirements (such as serviceability, reliability, portability and usability) and limited resource requirements (such as people, time and money). Projects often fail to specify these critical requirements adequately:

- not *all* the critical success factors are identified
- no target *numeric* values for survival and success are stated
- variations in targeted requirements for differing times and differing places, are not addressed:
  - the effect of peak loads, or system growth, on the required levels, is not taken into account
  - the concept of very different attribute levels, being required by different parts of the system, or by different stakeholders, is not considered
- no practical ways to measure the results delivered to stakeholders are specified alongside the requirement specification.

The result is that our ability to manage successful value delivery is destroyed from the outset. It is impossible to engineer designs to meet non-specified or ambiguous requirements. It also is impossible to track changes for such ill-specified requirements.

Focus on Results

The primary systems engineering task is to design and deliver the best technical and organizational solutions, in order to satisfy the stakeholders' requirements, at a competitive cost. Projects must ensure that their focus is on delivering critical and profitable results. Albert Einstein is quoted as saying: "Perfection of means and confusion of ends seem to characterize our age."[1] Unfortunately, this still appears true today. It is the delivery of the required *results* from a system that

[1] Calaprice, Alice [Editor]. 2000. *The Expanded Quotable Einstein*. Princeton University Press. ISBN 0-691-07021-0.

counts. The process used and the technology selected are mere tools in the service of the results.

Interdisciplinary Communication

Clear communication amongst the different stakeholder groups is essential. Common problems include:

- ambiguity, due to specification that lacks precise detail
- critical specifications being 'lost' in overwhelming detail
- technical specification being unintelligible to the management, who reviews it
- inadequate tracking of specification credibility: its source, status and authorization level.

Leadership and Motivation

Clear vision makes a huge difference. Clear vision gives a common focus for logical decision-making. When people understand the overall direction, they tend to make good *local* decisions. Only the critical few decisions need to be made at the top. It is important for all team members to be able immediately to channel their energies in a true common team direction.

Receptiveness to Organizational Change

It is also important for system engineers to know that their organizational culture really supports improvement in systems engineering methods. In other words, that people are actively encouraged to look for improvements and to try out new solutions. Positive motivation can be everything! It is not a case of demanding improvement, more a case of supporting and rewarding people who seek it.

Continuous Process Improvement

The quality guru, W. Edwards Deming considered that: "Eternal process improvement, the Plan-Do-Study-Act (PDSA) cycle, is necessary as long as you are in competition." Having best-practice systems engineering standards in place, measuring conformance to them and continually trying to improve them is necessary if you are to compete well.

The only thing that *should* not change is a great change process.

Practical Strategies for Systems Engineering

Planguage[2] (the specification language and methods described in this book) aims to support all the above concepts with practical ideas and methods; it has numerous practical strategies for projects to adopt.

In-built in all these Planguage strategies is risk management. Handling of risk is fundamental to Planguage. I do not believe that risk management should be a separate discipline. We can deal with risks better when we do so in every detailed specification and plan, and in every systems

Practical Strategies for Risk Management

1. **Quantify requirements**: All critical performance and resource requirements must be identified and quantified numerically.
2. **Maximize profit, not minimize risk**: Focus on achieving the maximum benefits within budget and timescales rather than on attempting to eliminate all risk.
3. **Design out unacceptable risk**: Unacceptable risk needs to be 'designed out' of the system consciously at all stages, at all levels in all areas, for example, architecture, purchasing, contracting, development, maintenance and human factors. This means selecting lower-risk options.
4. **Design in redundancy**: When planning and implementing projects, conscious backup redundancy for outmaneuvering risks is a necessary cost.
5. **Monitor reality**: Early, frequent and measurable feedback from reality must be planned into your development and maintenance processes, to identify and assess risks before they become dangerous.
6. **Reduce risk exposure**: The total level of risk exposure at any one time should be consciously reduced to between 2% and 5% of total budget.
7. **Communicate about risk**: There must be no unexpected risks. If people have followed guidelines, and are open about what work they have done, then others will have the opportunity to fight risks constructively. Where there are risks, then share that information.
8. **Reuse what you learn about risk**: Standards, and other forms of work process guidance, must capture and assist good practice. They must be subject to continuous process improvement.
9. **Delegate personal responsibility for risk**: People must be given personal responsibility in their sector for identification and mitigation of risks.
10. **Contract out risk**: Make vendors contractually responsible for risks. They will give you better advice and services as a result.

Figure 1.2
Practical strategies for risk management.

[2] Pronounced like 'language' with a 'p' as in 'plan.'

implementation process. Figure 1.2 gives a list of strategies for risk management. All these strategies can be found in some aspect of Planguage.

1.2 Practical Example: Twelve Tough Questions

Here are some probing questions for controlling risk. They are powerful tools, which will help you in your everyday work. I call them the 'Twelve Tough Questions' – see the next page.

These 'Twelve Tough Questions' will help you find out 'what people *really* know.' They will help you find out how strong a foundation their opinions and recommendations are based on. From the answers to these questions – or maybe the *lack* of answers – you can see risks; and what needs to be done to reduce them. Try asking these questions when you next review a proposal, or at your next decision-making meeting. You will probably see the power of them immediately. Get your *management* to ask these questions.

Copy this next page (permission granted as long as you include copyright). Carry it with you to your next meeting or frame it on your wall. Use it to arrest fuzzy thinking in your company and client documents.

Twelve Tough Questions

1. **Numbers**
   Why isn't the improvement *quantified*?
2. **Risk**
   What is the degree of *risk* or uncertainty and *why*?
3. **Doubt**
   Are you *sure*? If not, *why* not?
4. **Source**
   Where did you get *that* information? How can *I* check it out?
5. **Impact**
   How does *your* idea affect *my goals and budgets*, measurably?
6. **All critical factors**
   Did we forget anything critical to survival?
7. **Evidence**
   How do you *know* it works that way? Did it '*ever*'?
8. **Enough**
   Have we got a *complete* solution? Are *all* requirements satisfied?
9. **Profitability first**
   Are we planning to do the '*profitable* things' first?
10. **Commitment**
    Who is responsible for failure, or success?
11. **Proof**
    How can we be *sure* the plan is working, *during* the project, *early*?
12. **No cure, no pay**
    Is it '*no cure, no pay*' in a contract? Why not?

A full paper on this is available at www.Gilb.com

1.3 Language Core: Planguage Basics and Process Control

Planguage consists of a specification language and a corresponding set of process descriptions (or methods). The Planguage *language* terms are used together with the Planguage *processes* for specification, analysis, design (also called planning, engineering, architecture or problem-solving) and management of processes, projects or organizations.

Planguage Specification Language

The specification language (usually called simply 'Planguage') is used to specify requirements, designs and project plans. Planguage consists of the following elements:

- **A set of defined concepts**. The Planguage Glossary contains the master definition of concepts as used within Planguage (Examples of concepts: objective, goal and function).
- **A set of defined parameters and grammar**. The Glossary also contains the set of defined Planguage parameters (Examples of parameters: Scale and Meter) used for specification.

  The grammar consists of Planguage syntax rules. These syntax rules are given in this book by example, rather than being formally stated. The aim is to show 'best known practice' of how the Planguage parameters should be specified to be useful. Note the examples given are only 'reasonable examples,' the reader should feel free to add to them, to improve them and to tailor them.

- **A set of icons**. Each icon is used for graphical representation of a specific Planguage concept and/or parameter. Icons may either be keyed in on a keyboard, or drawn. *For example, <fuzzy angle brackets> are used to indicate a 'poor' definition in need of improvement and, the icon, '< >', is in the Glossary under 'Fuzzy'.*

Relevant subsets of the Planguage language are introduced throughout the book in the Language Core section of the chapters. More formal definitions can also be found in the Glossary.

Planguage Process Descriptions

The Planguage process descriptions (or methods) provide recommended best practice for carrying out certain tasks. The reader should consider these defined processes as useful 'starter kits', but should plan to extend, improve and tailor them to their needs,

purposes and experiences. The set of Planguage process descriptions is as follows:

- **Requirement Specification (RS)**. (*See Chapter 2 and sub-processes in Chapters 3, 4, 5 and 6*)
- **Design Engineering (DE)**. Design Engineering is concerned with identifying, selecting and sequencing delivery of design ideas (*see Chapter 7*)
- **Specification Quality Control (SQC)**. SQC is used for evaluating the quality of any technical document and, for identifying and preventing defects (*see Chapter 8*)
- **Impact Estimation (IE)**. IE is used for evaluating designs and monitoring the impact of results on the goals and budgets. It plays a central role in Design Engineering (*see Chapter 9*)
- **Evolutionary Project Management (EVO, also known as Evo)**. Evo is used to deliver results in a series of high-value (highest value/best benefit to cost ratio delivered earliest), small (say, less than 2% of total project development time) evolutionary steps (*see Chapter 10*).

Note:

1. *SQC measures the degree to which a specification follows its specification rules. It directly measures the 'loyalty to engineering standards.'*
2. *Impact Estimation and Evolutionary Project Management measure the power of the design ideas in the marketplace.*

The process descriptions for the above methods can be found in the Process Description section of the relevant chapters.

Standards

As Tom Peters pointed out in *Liberation Management* (Peters 1992), the only remaining reason for having a very large organization is to share 'know-how' about best practices. Standards are an important form of sharing such know-how. (Other examples would be patents, market knowledge and specific customer knowledge.)

Standards can be termed 'Work Process Standards.' They can be usefully classified into specific types of guidance as follows:

- Policies
- Rules
- Process descriptions
- Forms and document formats
- Defined concepts (such as found in the Planguage Glossary)
- Language conventions (such as Planguage grammar)
- Work rates (such as 'checking rate')
- Others.

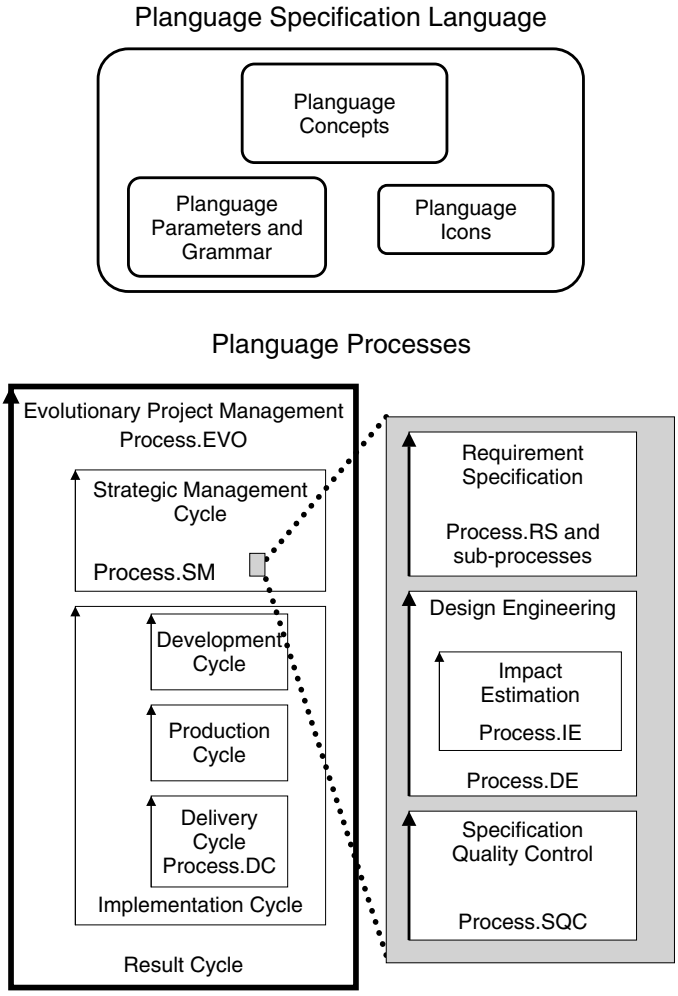


Figure 1.3
Diagram of the components of Planguage. Even more detailed, and more correct (as a consequence of being able to use the feedback from practical experience and, from any information being up to date) specification of the requirements and design ideas is likely to occur *within* the frequent development cycles. Detailed explanation of the Evo result cycles can be found in Chapter 10 (and in the glossary).

For specific examples, see Sections 1.4, 'Rules' and 1.5, 'Process Description.'

Standards can be *generic*, or can be tailored to *specific* tasks (for example, to contracting, testing, writing and installation) and tailored to *specific* stakeholder environments (for example, sub-supplier, novice, top management and customer).

Standards must be *brief* and *non-bureaucratic*. I favor, as a basic rule, limiting standards to *one-page* in length. I have found that my clients stick to a one-page format, finding it very practical. When there is only one page, detail cannot overwhelm people. For example, only the 20 to 25 or so 'most important' standards ideas can fit on a page (to be adopted, new standards must force bad ones 'off the page').

Standards must change as experience dictates. The *owners* of the standards must update the standards specification when better practices are discovered, so that new knowledge is shared and is rapidly put into use. People should be taught and motivated to use the standards, unless they can justify otherwise.

Rules

Rules are standards that provide specific guidance to follow when carrying out a process. They are also used in Specification Quality Control (SQC) to define and detect major defects in a specification. Individual rules should justify their presence in standards by the *potential resource savings* that can be expected from using them.

Process Descriptions

Process descriptions (or methods) are standards that describe the best practice for carrying out work tasks. The process format used for Planguage process descriptions consists of three basic elements:

- **Entry Conditions** – to determine whether it is wise to start the procedure
- **Procedure** – specifying for a task what work needs to be done and how best to do it
- **Exit Conditions** – to help determine if the work is 'truly finished'.

Entry Conditions

It is not good enough to allow employees to simply plunge into a work process and 'just do it.' The conditions must be right for success, not ripe for failure. Entry conditions are a list of what an organization has learned are the necessary prerequisites for avoiding wasted time and, for avoiding the failure of a specific work process.

Before beginning any procedure, its entry conditions must be checked. If the entry conditions are not met, then starting the procedure is high risk. It is likely to be better to remove the negative conditions before

proceeding. Entry conditions should be built on experience of what is high risk and high cost.

Procedure

A procedure is a sequenced list of instructions, describing how to carry out the task. It documents the current recommended best practice.

EXAMPLE P3: For each design idea, estimate its numeric impact on the Scale of all the attributes.
P4: Continue identifying/specifying or refining design ideas until the specified safety margin is reached.

Exit Conditions

Exit Conditions are used to evaluate if the task is reliably and economically completed. They specify the safe and economic conditions for exit from a process to a 'next' process. Exit conditions are also built on experience from previous releases to the next work process.

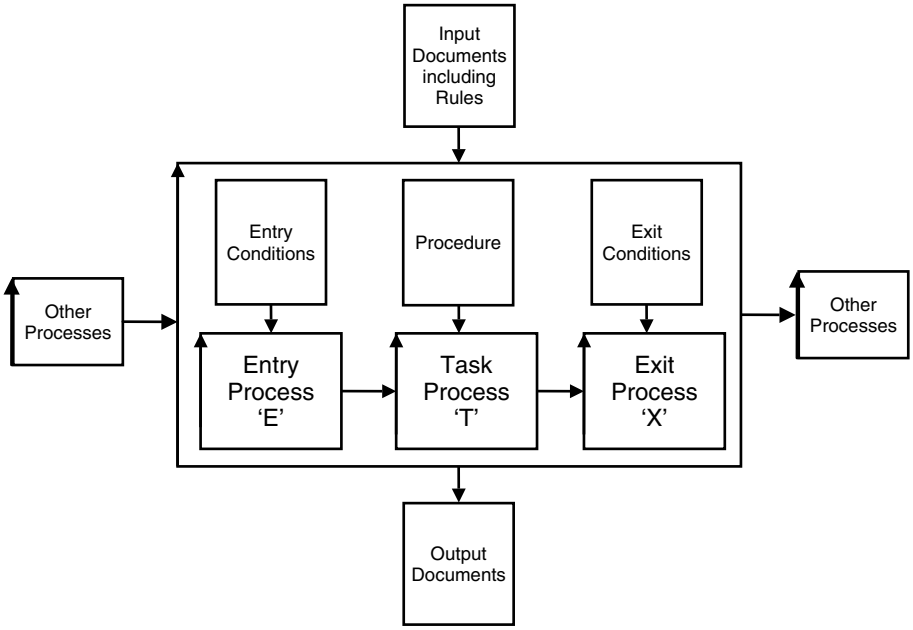


Figure 1.4
Diagram of a simple process showing its sub-processes and its relationship to other processes and documents. The input documents for each process include the rules, the entry conditions, the procedure and the exit conditions. The diagram also shows how the 'ETX' concept for a process is derived. A rectangle is the symbol for a 'written document.' A rectangle with arrow is a 'process' symbol. An example of such a process could be 'Requirement Specification.'

Table 1.1 Description of some of the main generic Planguage parameters, concepts and icons.

Basic Planguage Parameters, Concepts and Icons

Concept or Parameter	*Meaning*	*Used for*	*Note also*
Planguage Term	A term that is part of Planguage.	Structuring specifications.	Glossary contains a set of Planguage terms.
User-Defined Term	A term defined by users.	Identifying 'local' user terms.	It should be short and descriptive.
Type:	Type or category of a term.	Declaring the Planguage category of a user-defined term.	Type can be implicitly or explicitly declared.
Tag:	An identifier for a Planguage term or a user-defined term.	Providing a unique 'local' reference to a term.	Hierarchical tags can be used. These can be used in full (very explanatory) or abbreviated depending on context.
Gist:	A rough, informal, brief description or summary.	Getting consensus initially. Summarizing finally.	Usually not a precise, detailed or complete definition. For a scalar parameter, 'Ambition' can be used to express the ambition level.
Description:	A description.	Explaining terms.	Level of explanation detail should match the context.
Definition:	A definition, usually expressing the relationship to other user-defined terms.	Defining terms.	Synonyms are 'Defined' and 'Defined As.'
Version:	A date stamp. A time stamp can optionally be added.	Identifying a specific instance of a specification.	For example: 'Version: October 7, 2004 10:20.'
Stakeholder:	Any person or organizational group with an interest in, or ability to affect, the system or its environment.	Understanding who has to be consulted or considered when specifying requirements.	Usually a set of several different stakeholders is identified.
Authority:	The stakeholder or role responsible for determining status and enforcing use.	Identifying where the power resides.	The authority has the power to determine and change a specification. Also to control its availability.
Owner:	The stakeholder or role responsible for the overall specification itself.	Identifying the specification owner.	The owner usually is responsible for the updating of a specification.

Continued

Table 1.1 Continued

Basic Planguage Parameters, Concepts and Icons

Concept or Parameter	*Meaning*	*Used for*	*Note also*
Readership:	The stakeholder(s) or role(s) who will or might use the specification.	Identifying the specification user(s) or audience level to communicate to.	A synonym is 'Intended Readership.' A parameter such as 'Specification User' or 'Process User' could be used instead.
Status:	The approval level of the specification.	Identifying which version of the specification is being used.	For example: 'Status: Draft.' See glossary for additional terms to express approval level.
Quality Level:	The quality level of the specification in relation to its rules.	Stating the estimated defect density in a specification.	For example: 'Quality Level: 3 remaining major defects/page.'
Qualifier: [. . .]	A qualifier adds more specific detail to the specification regarding time, place and event conditions, [when, where, if].	States the conditions applying to a specification for it to be valid: the [time, place, event] conditions.	The keyed icon for Qualifier is '[ ]' as in '[Qualifier Condition 1, Qualifier Condition 2, . . . Qualifier Condition n].' The '[. . .]' icon is used far more than the parameter, Qualifier.
Source: <-	Where exactly a given specification or part of it, originated.	Used to enable readers to quickly and accurately check specifications at their origin.	The icon for source is '<-'. Usually the icon is used in specifications, rather than the term 'Source'.
Assumption:	Any assumption that should be checked to see if it is still applies and/or is still correct.	Risk Analysis	Other more precise parameters should be used if possible, for example, Dependency, Risk.
Note: "..."	Any additional comments or notes, which are relevant.	Used to provide additional information likely to help readers.	Any notes are only commentary and are not critical to a specification. 'Comment:' could be used as an alternative.
Fuzzy <. . .>	Identifies a term as currently defective and in need of improvement	Alerting the reader and author that the term is not trustworthy yet or lacks detail.	The keyed icon for fuzzy is '<imprecise word>'. The '<>' icon is always used.
Set Parentheses {. . .}	Identifies a group of terms, linked in some way, forming a set or a list.	Explicitly shows that a set of terms is being specified.	The context explains why the terms are a set. Usually, all terms are of the same Type.

1.4 Rules: Generic Rules for Technical and Management Specification

Here are some very basic generic rules, for *any* type of specification. You will find that in spite of their 'obviousness' and simplicity, they are quite powerful. Most of my clients use some variation of these 'by choice'.

Tag: Rules.GS.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Note: These rules are rather lengthy, as additional explanatory text is present. Readers should abbreviate as appropriate.

R1:[3] **Tag**: Specifications must each have a unique identification tag.

R2: **Version**: Specifications must each have a unique version identifier. By default, use the date (and maybe also, time), as the version identifier.

EXAMPLE Version: October 7, 2004 09:00.

R3: **Unique**: Specifications shall exist as *one official 'master' version only*. Then they shall be re-used, by cross-referencing, using their identity tag. Duplication ('copy and paste') should be strongly discouraged.

R4: **Owner**: The person or group responsible for authorizing a specification should be stated ('Authority' would be an alternative or supplementary parameter, though it is a different concept!).

R5: **Status**: The status for using a specification should be given.

EXAMPLE Status: SQC Exited.

R6: **Quality Level**: All specifications shall explicitly indicate their current quality level, preferably in terms of the measure of 'number of remaining major defects/page' against the relevant official standard which applies.

---

[3] The number is a rule tag (or identification, if you like) and the word after the colon is an equivalent alternative tag for referencing the rule. The following references are possible Rules.GS.R1, Rules.GS.Tag, Standards.Rules.GS.Tag and other combinations. The dot indicates that what follows is part of a set of things named by the term preceding the dot. For example, GS is part of a set of things called Rules.

EXAMPLE Quality Level: Less than 1 remaining major defect/page.

EXAMPLE Quality Level: Undetermined.

R7: **Gist**: Where appropriate, specifications should be briefly summarized by a Gist statement. For performance requirements, 'Ambition' is a preferred alternative.

R8: **Type**: The type of every concept within specifications should be clear. It should be explicitly specified after every new parameter tag declaration unless the type will be immediately obvious to the intended readership.

EXAMPLE *ABC1*: Type: Function.

R9: **Clear**: Specifications should be 'clear enough to test' and 'unambiguous to their intended readers.'

R10: **Simple**: Complex specifications should be decomposed into a set of elementary, tagged specifications.

R11: **Fuzzy**: When any element of a specification is unclear then it shall be marked, for later clarification, by <fuzzy angle brackets>.

R12: **Comment**: Any text which is secondary to a specification, and where no defect in it could result in a costly problem later, must be clearly identified. It can be written in *italic text* statements, or headed by suitable warning (such as Note, Rationale or Comment), or written in ''quotes,'' and/or moved to footnotes. Non-commentary specification shall be in plain text. *Italic* can be used for emphasis of single terms in non-commentary statements. Readers should be able visually, at a glance without decoding the contents, to distinguish between 'critical' and *'non-critical'* specification.

R13: **Source**: Specification statements shall contain information about their source of origin. Use the '<-' icon and state the source person and the date, or the source document with detailed statement reference.

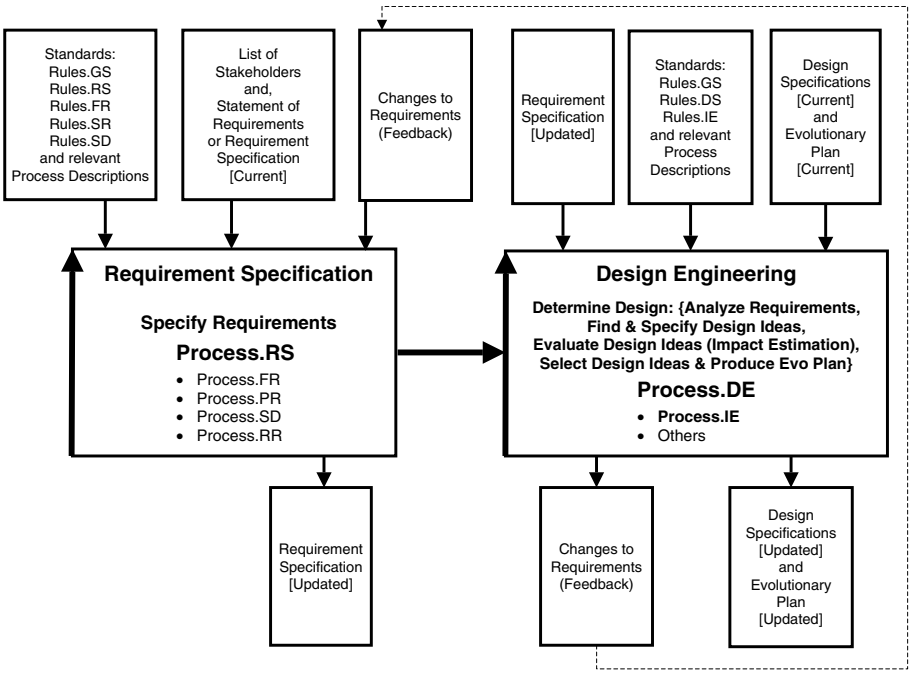
R14: **Assumptions**: All known assumptions (and any relevant source(s) of any assumptions) should be explicitly stated.

The 'Assumption' Planguage parameter can be used for this purpose. But there are also a number of alternative ways, such as {Risk, Source, Impacts, Depends On, Comment, Authority, [Qualifiers], If}. In fact, any reasonable device, suitable for the purpose, will do.

R15: **Risks**: You must specify any factors, which constitute known or potential risks. You must identify risks explicitly.

There are a wide variety of devices for doing so, including the explicit Planguage statement: 'Risks.'

EXAMPLE Goal [Market Y]: 60%.
Risks: Market Y will have more competition than now.



Notes:
Iteration of the processes has been allowed for by including existing specifications as potential inputs. Qualifying square brackets have been used around descriptive words, which are added to assist understanding. The aim is to show how the rules and process descriptions discussed in this book fit together. This diagram shows procedure steps P1 and P2 of the Generic Project process (Process.GP). These same processes are used during Manage Evolutionary Project (Process.GP.P3) – that is during Evolutionary Project Management – in order to update the requirements, the ideas and the Evo plan (see Figure 1.6).

The abbreviations used in this figure (and in the rest of the CE book) are as follows:

GP	Generic Project	RR	Resource Requirements
GS	Generic Specification	DS	Design Specification
RS	Requirement Specification	DE	Design Engineering
FR	Function Requirements	IE	Impact Estimation
SR	Scalar Requirements	EVO	Evolutionary Project Management
PR	Performance Requirements	SM	Strategic Management
SD	Scale Definition	DC	Delivery Cycle

Figure 1.5
An overview of the Planguage-defined requirement and design processes.

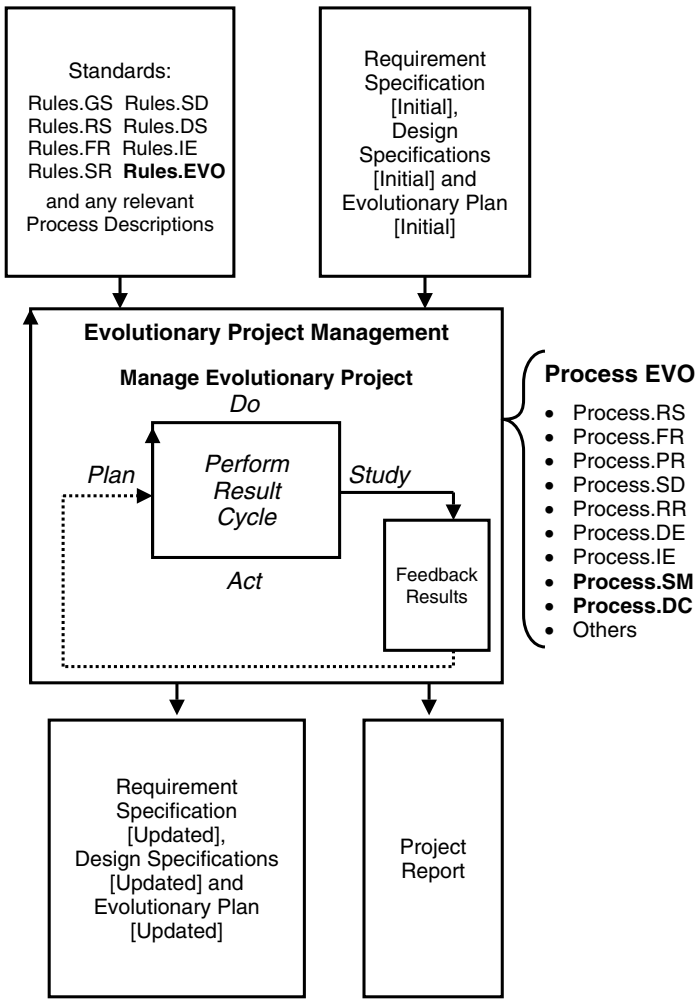


Figure 1.6
An overview of the defined Planguage process, which supports Evolutionary Project Management, Process.GP.P3 or in more detail, Process.EVO in Chapter 10.

1.5 Process Description: Generic Project

Process: Generic Project

Tag: Process.GP.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: A process specification giving an overview of the entire Planguage process for a project.

Entry Conditions

E1: The **Generic Entry Conditions** apply (*see separate specification for Generic Entry Conditions below*).

The *raw* requirements should have been gathered. The known sources of requirements should be identified and listed. These include:

- all the critical stakeholders
- all the currently identified requirements with detailed sources (use '<-' and, state who or which document) and any justification for these requirements (use the Rationale parameter).

Procedure

P1: **Specify Requirements** [**Initial**]: Specify the initial top-level requirements (*see Chapters 2, 3, 4, 5 and 6 as appropriate*).

P2: **Determine Design** [**Initial**]:

P2.1: **Analyze the Requirement**: Consider the stakeholder value and the delivery order for the requirements. Identify any constraints and any conflicts. Establish the scope for the system design.

P2.2: **Find and Specify Design Ideas**: Identify and specify the initial top-level design ideas to meet the requirements (*see Chapter 7*).

P2.3: **Evaluate Design Ideas**: Estimate the impacts of all the design ideas on all the requirements (*see Chapters 7 and 9*).

Re-do P1 to P2.3, until a reasonable balance between requirements and costs is obtained.

P2.4: **Select Design Ideas and Produce Evo Plan**: Produce an initial overview, long-term evolutionary plan of the sequence of Evo steps. That is, a plan for starting early delivery of required results by implementing the design ideas in a series of small result cycles. Each result cycle using, say 2% of total project time. (*That is, each result cycle is an Evo step. Note, an Evo step contains one or more design ideas.*)

Determine the sequence of step delivery of the potential Evo steps. Do this by calculating for each potential step, the performance to cost ratio, or ideally you would use the 'stakeholder view' of the value to cost ratio (the value being the benefits the stakeholders consider they will obtain from the system improvements). Ideally, sequencing should be in order of descending ratios, but consideration needs to be given to any associated dependencies (*see Chapters 7 and 10*). *Note this plan will be modified, within the result cycles, using the feedback provided by the results of implementing the design ideas (see below).*

P3: **Manage Evolutionary Project**: Iterate Plan-Do-Study-Act (PDSA) evolutionary result cycles until the exit conditions (below) are met. *Each result cycle implements the next Evo step and provides feedback to modify the design, and maybe, to adjust requirements to more realistic levels (within each result cycle, the processes Specify Requirements and Determine Design are reiterated to carry out any more detailed work required as part of the implementation of the Evo step, and to cater for any changes required as a result of the feedback), (see Chapter 10, 'Evolutionary Project Management').*

Note: When using Evo, as long as the Evo result cycles are delivering to the planned levels, the need for initial management review is considerably decreased (if not eliminated) as the resource commitment for each delivery step is only about 2% of the project total.

Exit Conditions

X1: The **Generic Exit Conditions** apply (see separate specification for Generic Exit Conditions below).

X2: Cease doing Evo steps (P3) when either the stakeholder requirements are met, or resource budgets are exhausted. In other words, stop when the performance requirements are met at planned levels, or when resources (budgets) are 'used up' at their planned levels.

Generic Entry and Exit Process and Conditions

Here is a process that can be used as a generic entry process or a generic exit process. The benefit of having one master generic process is that it is easier to review and update.

Process: Generic Entry or Generic Exit

Tag: Process.GE.E or Process.GE.X.

Version: October 7, 2004.

Owner: Systems Engineering Process Owner.

Status: Draft.

Gist: A generic process description that applies by default to all entry and exit processes.

Procedure

P1: Check all the conditions that apply.

P2: Note which conditions cannot be met.

P3: Decide if we can or must ignore specific failed conditions (waver).

P4: Attempt to correct, or help others to correct, any failed conditions in need of correction.

P5: Report status of the process in writing.

P6: Help management understand the reasons for and the risks of ignoring the problem of any failed or waved conditions.

P7: If management insists on overriding your advice, make sure the responsible manager, after being informed of the risks, is documented as overriding the formal process intentionally. (Make sure we know who to blame later and then they take the responsibility.)

P8: For exit only: Ensure any process improvement suggestions have been submitted to the relevant process owners.

P9: Allow exit/entry when all conditions are either met or waived.

Note: To simplify matters, no entry or exit conditions have been specified for this process!

Generic Entry Conditions

Scope: For systems engineering, all specification entry processes.

Owner: Systems Engineering Process Owner.

User: Specification Author [Default User: SQC Team Leader].

E1: All logically necessary input information for complete and correct specification is available to the specification author. This includes up-to-date documentation regarding specification standards.

E2: All input documents have successfully exited from their own quality control process.

Note: This usually implies between 0.2 and 1 maximum remaining major defect(s)/page (A page is 300 words of non-commentary text). 'Remaining major defects' is explained in Chapter 8, 'Specification Quality Control.'

E3: The specification author is adequately trained or, assisted by a qualified person.

E4: The specification author agrees that they are ready to successfully carry out the specification work.

E5: There is appropriate approval, including funding, for the specification process to proceed.

Generic Exit Conditions

Scope: For systems engineering, all specification exit processes.

Owner: Systems Engineering Process Owner.

User: Specification Author [Default User: SQC Team Leader].

X1: The specification author claims to have followed the specified process description standard.

X2: The specification author claims to have followed all generic and specific rules, which apply.

X3: Relevant SQC has been carried out and the quality level of each output specification meets its stated SQC criteria. By default, the quality level for any specification is that no more than 0.2 major defects/page[4] may remain. (A page is 300 words of non-commentary text.)

Note, for some processes, there will be an explicit statement on SQC criteria, which overrides this generic exit condition.

X4: As an additional optional measure, a cursory check of the specification by the author's supervisor shows that there is reasonable compliance with applicable rules. In practice, no major defects should be found when a relevant sample (size and content) of the specification is SQC checked for 15 minutes.

X5: Any process improvement suggestions identified have been submitted to the relevant process owners.

1.6 Principles: Generic Project

Principles are *teachings*, which you can use as guides to sensible action. Here is a set of fundamental principles:

1. **The Principle of 'Controlling Risk'**
   There is lots of uncertainty and risk of deviation from plans in any project.
   You cannot eliminate risk. But, you can document it, plan and design for it, accept it, measure it and reduce it to acceptable levels.
   You may want to avoid risk, but it doesn't want to avoid you.
2. **The Principle of 'Storage of Wisdom'**
   If your people are not *all* experienced or geniuses,
   You need to store *their* hard-earned wisdom in *your* defined process.

[4] A maximum of 0.2 remaining major defects/page is a very high standard. Beginners should try for about 2.0 and work towards better levels.

Capture wisdom for reuse,
Fail to write it, that's abuse!

3. **The Principle of 'Experienced Geniuses'**
   If you *do* have any experienced geniuses, don't just let *them* save projects;
   They should share *their* wisdom with colleagues, on how to avoid failures.
   Those who learn the hard way,
   Should share their easy way.
4. **The Principle of 'Grass Roots Experience'**
   Your grass roots people will *know* what is wrong with your work standards,
   So let *them* suggest improvements, every day.
   The soldier who has the boot on knows where it pinches.
5. **The Principle of 'Short and Sweet'**
   Keep your standards short and sweet,
   A single page will do the feat.
   Brevity is the soul of wit,
   All essentials, a page do fit.
6. **The Principle of 'Don't Refuse to Reuse'**
   Reuse good specifications, and don't repeat them,
   Once said suffices, no repetitious vices.
   Write once, use many.
7. **The Principle of 'High Standards'**
   Have high standards for your work process entry, to save yourself grief,
   Have high standards for your work process exit, to your friends' great relief.
   Note work standard conditions for success,
   Respect them; even in duress.
8. **The Principle of 'Quality In, From the Beginning'**
   Quality needs to be designed into processes and products,
   Cleaning up bad work is a loser, but cleaning early is better than late.
   A stitch in time still saves nine,
   But an ounce of prevention is still worth a pound of cure.
9. **The Principle of 'No Simpler'**
   The optimum guidance lies somewhere between anarchy,
   And too much bureaucracy.
   Things should be as simple as possible,
   But no simpler.[5]

[5] "Physics/theories/things should be as simple as possible, but no simpler". Reputed quote of Albert Einstein. Nobody seems able to prove he actually said it, but it is acknowledged to be in his spirit. Calaprice, Alice [Editor]. 2000. *The Expanded Quotable Einstein.* Princeton University Press. ISBN 0-691-07021-0.

10. **The Principle of 'Intelligent Insubordination'**
    A work process 'standard' isn't a law, just good advice,
    Ignore it, if you've better 'words from the wise'.
    Rules were made to be broken wisely.

1.7 Additional Ideas

Continuous Process Improvement

Conventional ways of getting control over systems engineering projects include:

- **resource** allocation adjustment (time, people, talent, money, sponsorship)
- **ambition** level adjustment (performance to fit within budgets)
- shift of **responsibility** (outsourcing, purchasing, contracting, democratization)
- **priority** management (sacrificing some things to get others, tradeoffs).

There is a less-understood *addition* to these ideas: **process control**. It is to get control over results by getting control over the *work processes* producing the results. In concept, this is Statistical Process Control with its famous Plan-Do-Study-Act (PDSA) cycle as taught by Shewhart, Deming and Juran (Deming 1986).

'Process control' is sufficiently well known within manufacturing. However, surprisingly, it has not become conventional practice within systems engineering. There are two main areas where its use is lacking.

First, process control is rarely exploited in the area of *project management*. This is in spite of there being 'software' literature, which documents good experience with process control since the

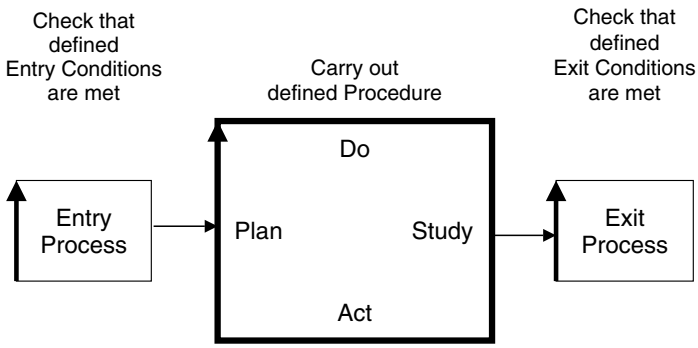


Figure 1.7
A *simplified* PDSA process cycle diagram as a basis for work process control, consisting of an entry process, a procedure and an exit process.

1970s: for example by Harlan Mills at IBM (Mills 1980), references such as (Gilb 1988) and, even military IT standards within the USA (such as MIL-STD-498 in 1994) (see Larman and Basili, 2003, for historical overview.) The problem is that this 'software' documentation is little known, having simply not been adequately recognized in mainstream *project management* literature. (In fact, there appears to be almost no reference at all to evolutionary project delivery and process control. The Waterfall method unfortunately dominates, according to my informal bookshop surveys and speaking with professional project management people.)

Secondly, the PDSA cycle concept is also underutilized in systematic *process improvement*. Use of numeric feedback for control is often not understood and not practiced. This is, however, being addressed in emerging standards for systems engineering and in US DoD 'Mandatory Guidelines' (DoD Evolutionary Acquisition 1998).

The key concept is that if a well-defined process is followed, then the process output performance levels will be a consistent and predictable result of that process. If attempts are then made to *change* the process, we can assume that systematically changed performance results (hopefully, better levels and lower variability) can safely be attributed to the process change, not chance. Unfortunately this powerful concept is frequently ignored. The false dogma is often spread that defined repeatable processes lead to quality. (In fact, this is only the initial stage of achieving a stable process, which is then ready for process change, as the prior stability enables proof of the cause-and-effect of the change.)

It is important that work process **standards** be the vehicle for *continuous, systematic work process practice improvement (productivity improvement)*. They must not remain static, when there is better know-how. They must not stand in the way of improvement. They must lead the way and teach the way. They must be *easily changed* and *frequently changed* to incorporate better ideas quickly, and *easily adapted* to suit changing circumstances or tailoring for local circumstances. The actual usage of work process standards must be measured, motivated and taught by using Specification Quality Control (SQC) sampling. SQC measures specification conformance to two classes of work process standards: official rules and exit/entry conditions.

Normally no more than one significant deviation (one major defect/page) from the specification rules should be allowed. Yet without SQC, 100 or more major defects/page will be your fate. This may seem astounding to people who have not measured it, but this, in my experience, is the norm in most organizations throughout the world.

Why Process Control?

- Use of Best Practice
- Reuse of Ideas
- Predictable Output from Stable Process
- Rapid Dissemination of Changes
- Ability to detect any 'Bad' Process Changes
- Process Measurement and Benchmarks

Not just 'having a Process', but using it as 'a vehicle for Change'.

Continuous work process improvement for a large organization can involve making changes to company standards and practices at the rate of 1,000 ideas implemented per year – as documented at IBM, Research Triangle Park (Mays 1995) and IBM Rochester (Minnesota) Laboratories (IBMSJ 1994). This process change rate seems to result in annual productivity increases of about 40%, as recorded for example at Raytheon Defense Electronics over several years (Dion 1993; Haley et al. 1995; see also Section 1.8 below; over the years studied, a total productivity increase of 270% was reported).

EXAMPLE **Calculating the effect of detected defects, if uncorrected, on the timescales of a project**
At a major U.S. multinational in October 1999, eight managers did a sample SQC on an 82-page system requirement specification. The only rules used were, 'clear, unambiguous, no design specifications in the requirements.' They found about 60 major defects/page.
Assumptions: My SQC experience has determined that:

- only about one third of the defects that are really there will be found by staff inexperienced in using SQC at the first pass
- each defect will result in 'an order of magnitude' extra work to fix when found downstream.

It is also assumed that there are 200 days per year and 8 hours per work-day (1600 hours/year).
Using these assumptions, it can be calculated that the project will incur 82 (number of pages) × 60 (number of defects/page), × 3 (as only a third effective in finding defects), × 10 (number of hours/defect) additional hours correcting defects = 147,600 hours or approximately 92 person years.
We can assume the probability of a major defect actually resulting in an average 10 hour delay is about 25%–35%. So at 25% we would lose 36,900 hours.
For a project with 10 programming staff, this meant roughly two years' delay.
We later that afternoon were told that the project using this 'approved' specification was actually at least one year late, probably 2 years late. This had in fact been predicted fairly accurately by our analysis, before we were told the reality!
In such an environment, simply continuing to fix specification defects as they are detected is not the sensible option. *Continuous process improvement* needs to be used to drive down the number of defects being injected into the specifications.

See also Chapter 8 and the Glossary for further detail on Specification Quality Control (SQC) and the Defect Prevention Process (DPP).

1.8 Further Example/Case Study: Continuous Process Improvement at Raytheon

This Raytheon case study outlines how measurable process improvement can be brought about using Specification Quality Control (SQC) and Continuous Process Improvement. Within Raytheon's Equipment Division, software process improvements have yielded:

- a 7.70 US dollars return on every dollar invested
- a greater than two-fold (2.7×) increase in productivity
- as measured by the Software Engineering Institute (SEI) Capability Maturity Model (CMM), an evolution from Level 1 (Initial) through Level 2 (Repeatable) to Level 3 (Defined) process maturity (and later beyond that).

More detail can be found in Raymond Dion's account of the software process changes within Raytheon (Dion 1993; Haley et al. 1995).

Background

Raytheon, a diversified, international, technology-based company, is one of the 100 largest corporations in the US. The Equipment Division is one of eight divisions, and 11 major operating subsidiaries within Raytheon, with annual sales that comprise about 13% of the corporation's $9.1 billion annual sales. In early 1988, many Software Systems Laboratory (SSL) projects were delivered late and over budget. That year, the SSL rated itself at CMM Level 1 (Initial), using the SEI capability-assessment questionnaire.

Aim

As a result, in mid-1988, the Equipment Division started a process-improvement initiative within the SSL. Within the initiative, four working groups directed the major activities: Policy and Procedures, Training, Tools and Methods, and Process Database (metrics). The initiative's fundamental aim was the continuous improvement of the development and management process. Their strategy was to use a three-phase cycle of stabilization, control and change in accordance with the (PDSA) principles of W. Edwards Deming and Joseph Juran.

Financing the Improvements

Discretionary funding (overhead, independent research and development, and reinvested profit) was the chosen solution. However, in

order to convince management to persevere, this approach required two important ingredients. First, there had to be some *short-term benefit to ongoing projects* and, second, there had to be a *meaningful quantification of what the benefit was.*

Measuring the Effects

In launching the initiative, they had to consider how individual small improvements, implemented more or less in parallel, would interact to produce a net loss or gain. They decided it would be easier to measure the overall effect of change on the 'bottom line'.

Calculating Savings

Raytheon used Philip Crosby's approach (Crosby 1996) to analyze a database of 15 projects. The analysis, indicated that they had eliminated about $15.8 million in *rework costs* through the end of 1992 (four and a half years). (Hewlett Packard, using this author's SQC methods, reported similar results (Grady and Van Slack 1994)). The Raytheon appraisal costs (a term meaning cost of auditing, testing, reviews and inspections) had increased by 5%. The increased rigor with which they conducted design and code inspections (SQC), accounts for some of this increase, but most of the Raytheon result is due to a 30% *decrease* in total project cost, which has pushed up appraisal cost *proportionally.*

Early delivery of one Raytheon project was reported to have given them a $10 million bonus from their customer. It was considered entirely due to the initiative. There were several other tangible benefits from the initiative. The saving in rework costs was only *one* of them.

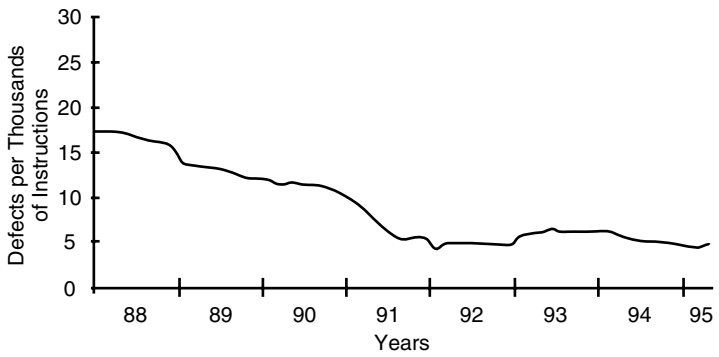


Figure 1.8
Another benefit from the effort: overall product quality, measured by software defect density, improved by about 3 to 1, from 1988 to 1995 at Raytheon (Haley et al. 1995).

1.9 Diagrams/Icons

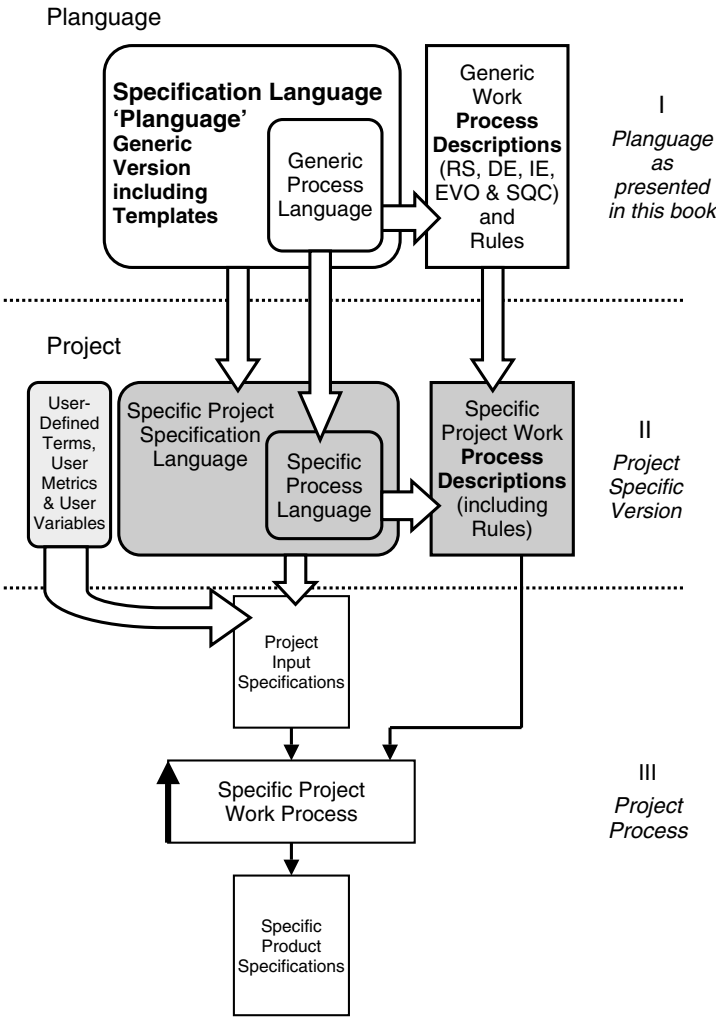


Figure 1.9

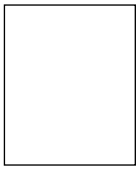
I. At the top of the diagram, the two main, generic components of Planguage, the specification language and the process descriptions are shown. (These two components correspond to the version of Planguage presented in this book.)

II. In the middle of the diagram, the specific version of Planguage (the project specification language and project process descriptions) selected for use by a project is shown. This specific version will have been tailored by the project. In addition, a project will have user-defined data. The user-defined data will always be unique to a project. It comprises the user-defined terms, actual numeric values (user metrics) and any user-assigned, non-numeric variables of the project specifications.

III. The bottom of the diagram is a generic model of a project process. It shows how the various components of the project specific version of Planguage (and the product data) map onto the project process.

Some Basic Planguage Icons

Document or Specification



A rectangle

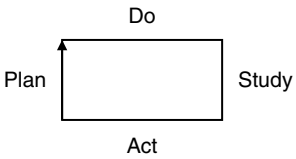
Process

A rectangle with an upwards pointing
arrow on its left hand side.
The arrow reminds us of the cyclical nature of processes.

Plan-Do-Study-Act Process Cycle

The four sides of the process icon symbolically represent the Shewhart process-cycle definition of 'Plan-Do-Study-Act'.

The process **input/output axis** is vertical and the process **control axis** is horizontal.[6] Specifications and other input materials are diagrammed as entering from the north and exiting from the south. Previous processes are connected from the west and subsequent processes are entered from the east. These conventions are independent of the PDSA activities, since one can enter and exit to and from any of these four process task types. (That is, you can step on and off the cycle at any point.)



[6] The traditional view as shown by Deming is a circle form with four arrows. I have chosen the rectangle as it is easier to generate and has other nice properties. I hasten to point out that Deming taught that it did not matter where in the cycle one entered a PDSA process, nor where one exited, though he was not so concerned with exit, as he viewed the cycle as an eternal process-control cycle, as long as there were competitive pressures to improve things. I believe this is true and, so, I hope my choice of representation does not inhibit the reader from entering and exiting processes wherever convenient or realistic (P, D, S or A).

Figure 1.10
Example of a corporate policy standard.

Notes Supporting the Example of a Corporate Policy Standard

1. **Quantify Critical Success Factors**:
   All critical success factors (function, performance and resource) for any activity (planning, systems engineering and management) shall be expressed clearly, unambiguously, measurably and testably at all stages of consideration: presentation, evaluation, construction and validation.
2. **Evaluate Risk**:
   In any planning or systems engineering work we shall explicitly document all notion of suspected or possible elements of risk or uncertainty, so nobody reading it can be in the least doubt as to the state of our certainty and knowledge.
3. **Assess Change Impact – To Exercise Control over Multiple Dimensions of Performance and Budget**:
   All design ideas (strategies, system components, processes or other devices) shall be evaluated with regard to their effects on all the critical objectives and budgets. Initially, this should be by estimates, which are based on facts and experience. On delivery, the design ideas shall then be evaluated by actual measurements taken as early and as frequently as possible.

4. **Ensure Change Control – Configuration Management and Traceability**:
   All statements of objectives, budgets, design ideas, and estimates and measures of the impact of design ideas on objectives and budgets shall be captured with explicit detailed information as to their sources, so that detailed change control is made effective and efficient.

5. **Perform Evolutionary Project Management**:
   All projects whether concerning organizational issues or product development, shall be controlled by a Plan-Do-Study-Act process control cycle. They shall have small increments of cost and time (in the 2% to 5% range normally) before attempting to deliver useful customer increments of function and/or performance improvement (at least some sort of field trial). Where there is any choice of incremental step content we shall choose the increment which gives the greatest quantified impacts in total on all critical customer or project objectives, with least resource expenditure.

6. **Ensure Continuous Work Process Improvement**:
   Practical priority will be given to measurable continuous improvement of all work processes in systems engineering, management and other company activities. Plans for type and degree of improvement will be budgeted; and progress towards improvement objectives will be measured. The ambition level will be world-class levels and to be the leader in any area. As a practical matter all employees are expected to participate in analysis of current defects found by quality control (for example, specification quality control (SQC) and test) and to spend effort improving the current work environment to eliminate 50% of the current defects every year over the next few years.

7. **Evaluate Specification Quality**:
   All documents, capable of producing a significant impact on our performance levels, must be evaluated using the best available quality control process. These documents must meet an appropriately high quality standard (that is a low numeric value for the 'maximum possible remaining major defects/page' as specified in our written standards and policies) before being released to any internal or external customer for serious use. The ultimate release level shall be state of the art (between 0.3 and 3.0 remaining major defects/page).

1.10 Summary: Planguage Basics and Process Control

This chapter has provided an introduction to Planguage and, hopefully, set the rest of the book in context. The main Planguage concepts

introduced in this chapter have been *processes* and *continuous process improvement* through use of process *standards*. Many examples of process standards will be found throughout this book. They aim to provide practical, step-by-step advice on how to implement Planguage.

Planguage is not a prescription of how I feel you should do things. It is a framework for you to discover how you best can do things yourself. Planguage is *open for change* from any source, at any time, for any good reason. It is intended to be totally in tune with the need for *continuous improvement* of all competitive systems and processes.

If Planguage doesn't save time and effort and improve quality, it fails. Don't use it! Please do not misunderstand Planguage as if it is an 'imposition of a lot of bureaucratic detail.' I hate bureaucracy as much as you do! But I hate failure even more. So, I am willing to use the Planguage disciplines; I find that they pay off and make my professional life easier and more successful. (Note: The Planguage methods actually work in *most* problem-solving situations; they can even be used in your *personal* life too!)

Planguage is concerned with *getting control over things*. If you want to be more in control of your work, Planguage has many practical techniques to help you. It takes some *learning*. It takes some *work to implement*. It takes *time to change the culture* around you.

In fact, human culture changes can be frustratingly slow; they can take years! But if you don't start this evolutionary process *now*, *this* week, *this* project, then the problems will get worse, not better. Can you afford to ignore the evidence from several major corporations, such as Raytheon, that continuously improved best practice standards can lead to substantial improvements in your team productivity annually over the next few years?