Principles of Software Engineering Management

Tom Gilb and Susannah Finzi
Addison-Wesley, 1988.

Chapter 1 The pre-natal death of the Corporate Information System (CIS) project

The invisible target principle
All critical system attributes must be specified clearly. Invisible targets are usually hard to hit (except by chance).

The all-the-holes-in-the-boat principle
Your design solutions must satisfy all critical attributes simultaneously.

The clear-the-fog-from-the-target principle
All critical attributes can be specified in measurable testable terms, and the worst-acceptable level can be identified.

The learn-before-your-budget-is-used-up principle
Never attempt to deliver large and complex systems all at once; try to deliver them in many smaller increments, so that you can discover the problems and correct them early.

The keep-pinching-yourself-to-see-if-you-are-dreaming principle
Don't believe blindly in any one method; use your methods and common sense to measure the reality against your needs.

The fail-safe minimization principle
If you don't know what you're doing, don't do it on a large scale.

Chapter 2 Overview

The disaster principle
Disasters don't happen by accident; they are entirely credible to our own management.

The right stuff principle
The right solutions will always fail to produce the right results if you have not defined exactly what the 'right results' are, and then made sure you had the right solutions to achieve those results.

Green's manager principle
Managers must manage.

Einstein's over-simplification principle
Things should be as simple as possible, but no simpler!

The third wave principle
You may forget some critical factors, but they won't forget you.

The multidimensional tools principle
If your tools can't operate in all critical dimensions, then your problems will.

The common sense principle
Common sense is uncommon.

The thinking-tools principle
Dynamic environments require thinking tools instead of unthinking dogmas.

Software
Software is all things which are not hardware in the system.

No man is an island
Software has no life independent from hardware, and must consider the properties of the hardware systems on which it resides, as well as the people involved.

Engineering
Engineering is a process of design, and trial-construction, of something, which aims to produce a system with a specified set of quality-and-cost attributes. We also accept the notion that engineering is the application of heuristics to solving problems.

Software engineering
Software engineering is primarily a design process; construction and use confirm that the right design ideas have been found.

Software engineering specialists
The software engineering discipline is already so complex that specialists in sub-disciplines are required to find the best designs.

The bricklayer principle
Calling a programmer a software engineer does not make him an engineer any more than calling a bricklayer a construction engineer makes him an engineer.

The real software engineer principle
A real software engineer can optimize any single attribute to become ten times better than it would otherwise have been.

The software engineering process principle
Software engineering has multiple measurable requirements as input, and appropriate design solutions as output.

Chapter 3 What is the real problem?

The principle of fuzzy targets

Projects without clear goals will not achieve their goals clearly. (You can't hit the bullseye if you don't know where the target is!)

> **Practical hint**
> If you intend to fail, or fear that failure is inevitable, then stick to unclear goals to hide your incompetence.

> **Practical hint**
> If you can think of several possible alternative specifications for getting what you want, then what you are specifying is solutions. Ask yourself, 'What do I really want?' These are your real goals. Alternatively, if you can ask, 'Would I be willing to drop this specification if I got what I really want, or if this specification were in conflict with what I really want?,' then your specification is probably just a solution, not a real requirement.

The principle of the separation of ends and means

Avoid mentioning solutions in your goal statements.

The principle of unambiguous quality specification

All quality requirements can and should be stated unambiguously.

Kelvin's principle

I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.

Shewharts' measurable quality principle

The difficulty in defining quality is to translate future needs into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price the user will pay.

The principle of the obvious

'Obvious' things, which 'everybody knows' cannot be left to take care of themselves.

The Achilles' heel principle

Projects which fail to specify their goals clearly, and fail to exercise control over even one single critical attribute, can expect project failure to be caused by that attribute.

> **Practical hint**
> You will probably need to create a scale of measure for the quality concept. If you thing you don't need to, then perhaps you don't understand the problem yet. If you are not feeling too creative today, try looking at Chapter 19 or possibly Chapter 9 for some ideas.

Hint: go out and get your hands dirty with the real users of the present systems. There are always comparable systems to look at, and these give clues about existing system measures and their levels.

The evil circle principle

If requirements are unclear, incomplete or wrong, then the architecture will be equally wrong.

If the architecture is wrong, then our cost estimates will be wrong.

If the cost estimates are wrong, then people will know we are badly managed.

If the high-level requirements and architecture are wrong, then the detailed design of them will be equally wrong.

If the detailed designs are wrong, then the implementation will be wrong.

So we end up re-doing the entire project as badly as the last time, because somebody will cover up the initial failure, and we will presume that the methods we used initially were satisfactory.

Chapter 4 What is a solution, and what is not?

The principle of a solution worth considering

A solution worth considering is one whose positive contribution to your requirements outweighs its negative ones.

The holy cow principle of solutions

Solutions are only valid as long as they best serve our current requirements, but no longer.

The principle of solution flexibility

A solution idea can be specified as a requirement, or as an answer to a requirement, depending on your priorities.

The unholy solution principle

Solutions are never holy; they can and should be changed in the light of new requirements, conflicts with other solutions, or negative practical experience with them.

The net result principle

Solutions should be selected and judged on their practical ability to contribute well to high-priority needs.

The hidden solution principle

Solutions should never be specified or implied in a goal statement, unless they really are the high-priority goal itself.

The management by results principle

Management must avoid the imposition of solution ideas, and instead concentrate on goal-priority specification.

Chapter 5 Evaluating solutions

The side-effect principle

You must find out by how much all your critical attributes are impacted by the proposed solution.

The principle of fuzzy numbers

Even if we don't know for sure, we can make a rough estimate, and improve it later.

The uncertain certainty principle

Uncertainty must certainly be stated in no uncertain terms.

Chapter 6 Estimating the risk

The risk principle
If you don't actively attack the risks, they will actively attack you.

The risk sharing principle
The real professional is one who knows the risks, their degree, their causes, and the action necessary to counter them, and shares this knowledge with his colleagues and clients.

The risk prevention principle
Risk prevention is more cost-effective than risk detection.

The promise principle
Never make promises you cannot keep, no matter what the pressure.

The written promise principle
If you do make any promises, make them yourself, and make them in writing.

The promise caveat principle
When you make a promise, include your estimate of how much deviation could occur for reasons outside of your control, for reasons within your control, and for reasons others in the company can control.

The early reaction principle
When something happens during the project that you did not foresee, which increases deviation from planned risk, immediately raise the issue, in writing, with your constructive suggestion as to how to deal with it.

The implicit promise principle
If you suspect someone else – your boss or a client – of assuming you have made promises, then take the time to disclaim them, and repeat the promises you have made, if any, in writing.

The deviation principle
When indicating possible deviation, make a list of the possible causes of deviation, as well as a list of the actions you could take to control those risks.

The written proof principle
Hang the following sign near your desk: if you haven't got it in writing from me, I didn't promise it.

The principle of risk exposure
The degree of risk, and its causes, must never be hidden from decision-makers.

The asking principle
If you don't ask for risk information, you are asking for trouble.

The 'why not the best?' principle
The 'best imaginable' can be a reality, if you are willing to risk or sacrifice any other planned attributes.

The uncertainty motivation principle
Uncertainty in a technical project is half technical and half motivational, but with good enough motivation, uncertainty will not be allowed to lead to problems.

Ng's visibility principle
We don't trust it until we can see it and feel it.

The reality principle
Theoretical estimation is as accurate as our oversimplified estimation models backed by obsolete historical data. The real thing is a somewhat more reliable indicator.

Chapter 7 An introduction to the 'evolutionary delivery' method

The 'small is beautiful' principle
It is easier to see and deal with the effect of one small increment of the solution, than it is to understand the impact of the entire solution at once.

The principles of Tao The Ching
That which remains quiet, is easy to handle.

That which is not yet developed is easy to manage.

That which is weak is easy to control.

That which is still small is easy to direct.

Deal with little troubles before they become big.

Attend to little problems before they get out of hand.

For the largest tree was once a sprout,
the tallest tower started with the first brick,
and the longest journey began with a first step.

Capablanca's next-move principle
There is only one move that really counts: the next move.

The Norwegian mountain survival principle
You need never be ashamed of turning back in time.

The juicy bits first principle
If you deliver the juiciest bits of the project first, you will be forgiven for not providing all they dreamt about, or for not doing it as cheaply and quickly as they hoped.

The mountain goat principle
Take one step at a time up the slippery mountainside, and make absolutely sure that each hoof is on solid ground before you take the next step.

The 'how little?' principle
An ideal next evolutionary step costs as little as possible to implement, and gives as much as possible in the way of results to the end user.

The never-too-small-for-evolution principle
When you think your project is too small for evolutionary step delivery, you have probably misjudged the real size of your project.

The self-fulfilling truth principle
Evolutionary development estimates tend to come true, because if they were false, but become critical, we can correct the project trajectory to hit them.

Lindbloom's scientific muddling principle: The iron law of incrementalism (Lindbloom, 1980)

If a wise policymaker proceeds through a succession of incremental changes, he avoids serious lasting mistakes in several ways.

Chapter 8 Function specification

The functional requirements principle

The functional requirement specification lists essential things which the product must do, and which must be delivered at specified times. It is different from the quality-and-resource attributes required, and from the solutions selected to reach those attributes.

The binary function principle

A functional specification is 'present' or 'absent' from a plan or a real system. There is no such thing as a degree of presence or absence of a function.

The explosion principle

Function ideas can be defined in more detail by exploding them into constituent ideas.

The delineation principle

A functional specification is not necessarily a specification to build something new. It can be used to delineate the existing functions whose attributes are to be impacted by planned changes at planned points in time.

> **Practical hint**
> I recommend that every single function idea or function specification begin on a new line, and be tagged by a unique identification of some sort.

The essentials-first principle

Divide functions into 'things which can be implemented early,' and 'things which can be delivered later.'

The divide-to-communicate principle

When making functional explosions for purposes of presentation or analysis, you can divide them into simple categories such as existing groups within your organization or other categories which may make it easier for other people to understand your plans.

Chapter 9 Attribute specification

The critical control principle
All critical attributes must be specified and controlled throughout the project and product lifetime.

The measurability principle
All attributes can and should be made measurable in practice.

The attribute hierarchy principle
It is often convenient to express attributes as a hierarchy of attributes and sub-attributes.

The result-oriented attribute principle
The attributes should be specified in terms of the final end-user results demanded.

The prerequisite principle
The initial attribute specification must be made early (day one, hour one) in the project, before any attempt is made at solution specification.

The fluid attribute-level principle
Don't ever try to freeze exact attribute requirements. You must expect changes by the user during development, and because of the uncontrolled side-effects of your real system.

The overview principle
The attribute requirement specification should be written on one page, and in a consistent language.

The 'set' principle of hierarchical measurement
The lowest measurable levels of attribute specification are the sets of measures which define all higher level attribute names which group them.

The principle of detail
You must define things to whatever level of detail is necessary to control the critical parameters of your system.

The iterative specification principle
Write down your first thoughts, then improve upon them continuously. (It is easier to refine writings than thoughts, especially when other people are going to help you.)

> **Practical hint**
> Imagine some level which is obviously totally unacceptable under any circumstances. Imagine improvements along the scale until you begin to wonder if the level you are imagining might be acceptable to some users, under certain conditions. You are now in the right area. If necessary, define several worst case levels, using the qualifier (in parentheses).
> For example:
> Worst (for beta test site users) = 80%
> Worst (for initial paying customers) = 95%
> Worst (one year after initial customer delivery) = 99%

Chapter 10 Solutions: how to find and specify them

The solution determination principle
Attributes determine solutions.

The principle of full coverage
You must have enough solutions to meet all your objectives.

The principle of what really determines the solutions
Attributes determine solutions. But functions and their environment determine attribute requirements.

The S = f(A) principle
Solutions are a function of all attribute requirements.

The infinitely complex principle
The process of finding a complete set of solutions for all attribute requirements is complex and is never perfected.

The imperfect design principle
The perfect design solution will never exist. You won't ever have time to find a perfect design solution, but you can continue working toward one for the system lifetime.

The moving target principle
Since real attribute requirements will be forced to change in time, the design must be correspondingly adaptable to be able to meet them in the future.

> **Practical hint**
> Don't worry about getting the design solution complete or perfect. You can't, anyway.
>
> Do a reasonable job, so that you reduce wasted time in evolutionary implementation steps.
>
> Be prepared to learn from evolutionary implementation experience, to change your design whenever necessary in order to keep it in accordance with the real needs and the real technology.
> You can't get it right the first time, but you can get a good start, and then you can get it more right as you evolve the system.
>
> The most important design effort initially is to place a large number of open-ended design strategies in place, so that your learning and change process will be comfortable.

The tag principle
Untagged ideas never die, but they do just fade away.

Practical hint
In large systems you will want automated support to keep track of tags and find their cross-reference. A data dictionary system or database may be a useful start. Some users will want to extend this design control concept to a longer term configuration control scheme, for multiple delivered and maintained versions of the software.

If the documentation you have to work with does not have tags, then create them for your own protection: create tags for the most elementary design statements; use a hierarchical design statement and tagging method.
The most powerful motivator for thorough use of tags is the use of Fagan's inspection method. When you have to cross check a number of voluminous specifications and designs for detail, then the tag saves you a lot of time scanning many documents to find what you actually want.

Chapter 11 Solution evaluation

Practical hint

The IE table can be nicely put onto a conventional spreadsheet program on a personal computer. It can also be used then to produce charts of design progress.

The principle of impact estimation tables

Estimating the impacts of many solutions on all objectives is filled with sources of error, all of which together amount to a smaller error than the error of not trying to estimate at all.

Practical hint

We have at times used spreadsheet software to calculate the above measure, and to give some impression of the strong solutions and weak ones. Remove the weakest ones first.

Practical hint

Impact estimation side-effects can serve as a systematic argument for or against a particular solution when making presentation to colleagues.

The principle of side-effect estimation

Side-effect estimation brings out the good news, and the bad news, early.

Practical hint

Most managers seem to feel comfortable when the safety factor is two or better. This implies a minimum of 200% for quality attributes and a maximum of 50% for resource attributes, in the impact estimation sum. Use these as starting safety factors until you get more experience.

The safety factor principle

To hit the bull's eye at least once, use a better bow than seems necessary, allow three arrows at least, and borrow Robin Hood if you can.

Practical hint

Be prepared to backup the detailed numbers in the top level presentation with lower level estimates. This level of presentation is suitable for graphic bar chart presentations.

The estimation hierarchy principle

Deeper estimation gives better generalization, or the more trees you actually count, the more sure you are about the woods.

The fundamental principle of estimation

Perfect estimation of complex systems costs too much.

Practical hint

Don't ever look back. Do not attempt to compare your impact estimation numbers with the real system results. You will cause yourself unnecessary

grief. The main point is to motivate you to change your design, not to predict its attributes. The only thing you want to compare with reality is your high-priority targets.

Of course we need to learn from past mistakes of estimation, but the impact estimation table is ot the right place to do this because it is at too early stage of design. Too many changes are made after the estimates are made.

Practical hint
Your system design standards should contain rules for when you require justification of an estimate in writing. Inspection checklists for the impact estimation should ask questions like:

1. Are all estimates obviously reasonable?
2. Are all controversial or large estimates (over 5%) satisfactorily justified in writing?

Get at least one independent person to sign off on any impact estimation (inspector, peer review or a manager).

Practical hint
Get a thorough inspection done at least once on an impact estimation table, then review the experiences with your team before deciding whether you have time to do this or not. Look for the indirect side-effects of this process to give the clearest benefit, such as getting goals clarified and definitions of solutions clarified.

The principle of early estimation
You don't have to estimate the impact of design suggestions – but if you don't, you will find out when you implement them just how bad they are.

The expert principle
Experts know they don't know, the others try to fool people that they do.

Practical hint
Demand that everyone who proposes an idea ensures (not necessarily by doing it personally) that an impact estimation, with justification, is available for at least the 'top ten' attributes, before they seriously push the idea or suggest it to others. If they don't, you will have to.

Don't let this stop creative brainstorming – but do let it be the brainstorming filter.

The estimator principle
Design solutions alone have no value except when we can estimate contribution to our design objectives.

Practical hint

If people hesitate to commit themselves to numbers initially, use the impact analysis language to get them moving, discussing and communicating.

The impact analysis principle

Even superficial systematic analysis of solution completeness will turn up many defects in our design.

Practical hint

Keep it public. Keep the solution comparison model and its evaluations and their basic facts open and available to everyone. There will always be forces which push for secrecy, discover the best solutions.

The principles of solution comparison

Solutions must be compared on the basis of their impact on all critical objectives. Anything else is a false comparison.

or

The best solution is the one that is best for your objectives, there is no generally best solution.

Chapter 12 The inspection process: early quality and process control

The principle of Fagan's inspection rules

Fagan developed inspection at IBM, therefore his inspection rules had to contribute to net profit in order to survive.

and

If you think an inspection rule is unnecessary, you have probably misunderstood the method.

> **Practical hint**
>
> Follow inspection rules completely until you can prove that dropping them or varying them gives better measurable results. If you simplify or modify the method before you can measure it, then you won't understand why Fagan did it that way.

> **Practical hint**
>
> Make sure that the introductory year of using inspection in your environment is led by champions of the cause. Make sure that the champions are well trained. Send them on a public course on the subject. Make sure they are well read in inspection literature. Make them responsible for seeing that the moderators do their job properly, as reflected in the statistics collected about effort and effect.

The moderator principle

Inspections without trained moderators will only have moderate success.

> **Practical hint**
>
> Some companies plan inspections up to the lunch hour or end of office hours, hoping that people will get together in their own time for this purpose. Those who have instituted this on company time report that it pays off in terms of constructive and creative ideas.

The inspection-steps principle

If you think some of the inspection steps are too time consuming, and you drop them, you are in danger of losing more time than you save. (You can find out for sure by analyzing you statistics.)

The single page principle

If a subject is really important, and you understand it really well, then you can state it on a single page.

> **Practical hint**
>
> Statistics about costs of finding and fixing defects are vital for defending and improving the method of inspection and other software engineering methods in your development process. If you fail to collect and analyze them, you risk leaving bad methods in too long, and risk not recognizing and selling better methods to colleagues and management. Don't fail to collect and use the statistics; it is a necessary overhead.

The statistics principle
Inspection without statistics is like night driving without headlights; you may not see obstacles or opportunities until it is too late.

> **Practical hint**
> Publish optimum rate curves on your internal newsletter, and publish examples of estimated and real losses due to inspections being conducted outside the best rates of speed. Tell people where information about optimum rates can be found, and give an idea of the valid rates.

The inspection-rate principle
Inspection rates are to defect-finding efficiency what automobile speed rates are to fuel efficiency; too slow or too fast are both wasteful – and you have to measure carefully to find the optimum speeds for different conditions.

> **Practical hint**
> If you collect inspection statistics and compare them with test statistics and operational statistics, you will be able to measure the benefits of inspection yourself, very quickly. See the Omega project example of very early-in-initial-use attempts to quantify the benefits of inspection.

> **Practical hint**
> Make sure you measure and predict savings in manpower, money and time to your management in order to justify your investment in starting and continuing an inspection effort. One client of mine neglected to do this, and dropped inspection. One year later they discovered that 400 similar programs were ten times cheaper to maintain than 400 similar non-inspected programs. (ICI, UK.)

The universal inspection principle
Inspection can be used effectively on any technology or management documentation. It is not limited to software.

> **Practical hint**
> You can demonstrate the power of inspection to your management by inspecting marketing and management documents. This device can be used to get understanding and goodwill from management. Remember, your own projects should be integrated with both the highest company planning and marketing strategy documents.

The principle of highest level inspection
If you fail to inspect the higher levels of planning and goal-setting, then inspection at the lower levels will only serve to confirm errors made earlier!
or
What is put into a design-or-planning process should always have exited successfully form inspection beforehand.

Practical hint

You must be prepared to raise the clarity of planning, requirements specification, and design documentation substantially in order to exploit inspection. Inspection will in itself stimulate this improvement by making poor practices more publicly embarrassing.

The invisible-defects-don't-count principle

If you don't understand exactly what someone says, you cannot be sure if they are wrong.
or
There is a good reason why politicians make vague promises.

Chapter 13 Evolutionary delivery planning and implementation

Practical hint

Use cross-reference tags in your evolutionary plan specification to systematically check that all solution and function specifications are to be found in some step of your plan. You can refer to a large set of the ideas by means of a single high level tag, at rough step planning stages.

The principle of open-ended architecture

All solution ideas will to some degree allow change in a measurable way.

Each solution idea has multiple ease-of-change attributes.

The expected range of each solution idea's ease-of-change attributes can be noted and used to select them for new designs.

The need for open-endedness is relative to a particular project's requirements.
Each open-ended solution idea has side-effects which must ultimately be the basis for judging the ideas for possible use.

You cannot maximize the use of open-endedness – but must always consider the balance of all solution attributes against all requirements.

You cannot finally select one particular open-ended design idea without knowing which other design ideas are also going to be included.

There is no final set of open-ended design ideas for a system; dynamic change is required and inevitable because of the external environment change.

Open-endedness will, by definition, cost less in the long term, but not necessarily more in the short term.

If you don't consciously choose an open architecture initially, your system's evolution will teach you about it the hard way.

The principle of evolutionary delivery

All large projects are capable of being divided into many useful partial result steps.

The only critical step is the next one.

Evolutionary steps should be delivered on the principle of 'the juiciest one next.'

Result delivery (not the construction activity) is the only point.

Open-ended architecture should be at the base, otherwise the step transition cost will be unnecessarily high.

Any step sequence you plan will be changed by the facts you learn as you deliver the early steps.

Maximizing your real progress towards your specified goals is the only measure of successful evolutionary delivery.

The evolutionary process can lead to change of our technical design solutions, or change of your lower-priority requirements so that you can reach your higher-priority goals instead.

You don’t need to recreate a minimum working system before making your first improvements, if you use an already existing system to make a start with.

If the evolutionary delivery method doesn’t work, then you haven’t been doing it properly.

Chapter 14 The management of software productivity

The user as judge principle
The end users themselves, not the producers, should be the final judge of productivity in the sense of software quality.

The never ending judgement principle
Software systems need to be judged on a continuous basis throughout their lifetime – not just by the first user, the first month.

The multiple test principle
Software systems should have formally defined acceptance test criteria which are applicable at all times for all critical qualities.

The principle of software productivity
If is not the software itself which is productive. The interesting results are created by people who make use of the software.

If you can't define it, you can't control it.
The ore precisely you can specify and measure your particular concept of productivity, the more likely you are to get practical and economic control over it.

Productivity is a multi-dimensional matter
Productivity must be defined in terms of a number of different and conflicting attributes which lead to the desired results.

Productivity is a management responsibility
If productivity is too low, managers are always to blame – never the producers.

Productivity must be project-defined; there is no universal measure
Real productivity is giving end users the results they need – and different users have different result priorities, so productivity must be user-defined.

Architecture change gives the greatest productivity change
The most dramatic productivity changes result from radical change to the solution architecture, rather than just working harder or more effectively.

Design-to-cost is an alternative to productivity increases
You can usually re-engineer the solution so that it will fit within your most limited resources. This may be easier than finding ways to improve the productivity of people working on the current solution.

A stitch in time saves nine
Frequent and early result-measurements during development will prevent irrelevant production.

The ounce of prevention (which is worth a pound of cure)
Early design quality control is at least an order of magnitude more productive than later proeuct testing. This is because repair costs explode cancerously.

Do the juicy bits first

There will never be enough well-qualified professionals, so you must have efficient selection rules for sub-tasks, so that the most important ones get done first.

Chapter 15 Some deeper and broader perspectives on evolutionary delivery and related technology

Chapter 16 Ten principles for estimating software attributes

The dependency principle
All system attributes are affected by all others.

The sensitivity principle
Even the slightest change in one attribute can cause uncertainly large changes in any other attribute.

The zeroth law of reliability generalized
You can reach almost any ambitious level if you are willing to sacrifice all the other attributes.

The design-to-price principle
You can get more control over costs by designing to stay within interesting limits, than you can by passively trying to estimate the costs resulting from a design which gives priority to other objectives.

The iterative estimation principle
You get more control over estimation by learning from evolutionary early-and-frequent result deliveries, than you will if you try to estimate in advance for a whole large project.

The quality determines cost principle
You cannot accurately estimate the costs of anything when cost determining quality attributes are unclearly defined.

The natural variation principle
All system attributes can be expected to vary to some degree throughout their lifetime.

The early bird principle
Any method which gives you early feedback and correction of reality is more likely to give you control over the final result than big-bang methods.

The activist principle
Estimation methods alone will not change a result which is off the track. Active correction must be a part of your methodology. (Action, not estimation, produces results)

The 'future shock' principle
Data from past project might be useful, but it can never be as useful to you as current data from your present project.

Chapter 17 Deadline pressure: how to beat it

The deadline mirage principle
Rethink the deadline given to you; it may not be real.

The solution mirage principle
Rethink the solution handed to you; it may be in the way of on-time delivery.

The other viewpoint principle
Rethink the problem form other people's point of view; it will help you simplify your problem and convince them to agree with you.

The expert trap principle
Don't trust the experts blindly; they will cheerfully lead you to disaster. Be skeptical and insist on proof and guarantees.

The all-at-once trap principle
Remember, nobody needs all of what they asked for by the deadline. They would simply like you to provide the miracle if possible.

The real needs principle
Don't damage you credibility by bowing to pressure to make impossible promises. Increase your credibility by fighting for solutions which solve the real needs of your bosses and clients.

The ends dictate the means principle
If the deadline is critical and seems impossible to reach, don't be afraid to change to solution.

The principle of conservation of energy
If deadlines are critical, make maximum use of existing systems and 'known technology.' Avoid research-into-unknowns during your project.

The evolutionary delivery principle
Any large project can be broken down into a series of earlier and smaller deliverables. Don't give up, even if you have to change the technical solution to make it happen. Keep you eye on results, not technologies.

The 'don't blame me' principle
If you succeed using these principles, take the credit. Give your boss and these ideas some credit in a footnote. If you fail, you obviously didn't apply these principles correctly. If you must blame somebody, don't mention my name, mention your boss's. (Management is always at fault.)

Chapter 18 How to get reliable software systems

The first principle of solutions

There are usually a lot of them. You have to find an appropriate set of solutions to the total (not just for reliability) set of attributes-objectives which you have decided to aim for.

The second principle of solutions

If is almost impossible to know exactly what the exact attributes of a specified solution are in advance. But you can know approximately.

The first defense principle

The first defense is to not expect to get any particular level of attributes until you can measure that you have them.

The second defense principle

Make it an integrated par to of your design process to validate the solution ideas in practice before you promise them to anybody else.

Chapter 19 Software engineer templates

Practical hint

Don't' be too concerned with defining the attribute category itself. The important thing is to get a specification somewhere of all critical attributes.

Chapter 20 Principles for motivating your colleagues to use quality metrics

Practical hint

You might have to use written caveats. The simplest one is to practice giving estimates with explicit uncertainty factors like: 60 ± 20% or 60 - 80% or 60?? However, entire pages may need a 'rubber stamp' like 'Exploratory Uncommitted Estimates Only.'

Or you might include in such documents the following: 'Warning: the numbers in this document do not represent predictions or promises. They are used to improve communication about possibilities in a complex system. Unless otherwise explicitly indicated, you can expect realities to be very different from these numbers. Final numbers may depend on changed priorities, new technological decisions and implementation experiences.'

Practical hint

My favorite initial 'teaching' trick is immediately to connect the proposed technical solutions to the quality metrics using the impact estimation table. This is not a solid reality, of course. But it does force people to thing about the meaning of both their objectives and the technological solutions they are considering.

Practical hint

Make it clear that the estimates of requirements of impacts are not tied to the individual who first mentions the estimate. The team must accept the responsibility for the number. Hidden anonymously among peers ('they didn't know any better either!'), timid but creative individuals might just venture a valuable initial opinion.

One technique to dramatize that the numbers are not static is to modify them intentionally, using such simple tools as spreadsheet software, to do impact estimation or to do attribute specification. Ask a lot of 'What if?' questions, by changing the data on the spreadsheet, to get people thinking; but also to get the message across that the numbers can be changed as easily as a budget suggestion.

Explain that the numbers are highly dependent on many factors, which are not pinned down yet (like the budget for the development, or the other quality requirements and their priority). Explain that numbers are the best available tool for rationally exploring a complex technological design.

Explain that no early requirements, wishes or impact estimates are holy if we decide that something else has a higher priority. There is no need to feel

guilty if the final numbers five years from now are different from what anyone suggests now. But, numbers, at least approximate ones, are necessary for getting us on a controlled and efficient iteration path towards our future, even if that future involves dynamically changing requirements and technologies.

Chapter 21 The Omega project: inspection experience

Chapter 22 The production planning case