The Fourth Principle: Evidence and Risk-Based Decision Making

Look before you leap
Anonymous proverb

He who hesitates is lost
Anonymous proverb

The Third Principle emphasizes having a variety of systems engineers and stakeholders concurrently understanding the system’s needs, envisioning opportunities, exercising prototypes, and developing, verifying, and validating operations concepts, requirements, architectures, and life cycle plans. Without some mechanism to synchronize and stabilize all of this concurrency, this could be a recipe for chaos.

One of the several contributions of the Fourth Principle is to serve as a means to synchronize and stabilize the concurrent activities and system artifacts. As discussed in explaining Figure 5 of the ICSM Distilled overview (The Concurrency View), what is reviewed at ICSM decision milestones is the evidence that the various concurrently-developed artifacts satisfy a combined set of feasibility criteria for the system. This implies that the system’s operational concept, requirements, architecture, work breakdown structure, plans, budgets, and schedules need to be sufficiently complete and consistent (i.e., synchronized and stabilized) to be able to support the production of convincing evidence.

Having evidence serve as the principal decision criterion at milestone decision reviews is a considerable step forward from traditional schedule-based or event-based reviews. An initial step forward in systems engineering and acquisition guidance was to progress from schedule-based major project reviews (the contract says that the Preliminary Design Review is scheduled for September 30, so we’ll have it then, whether we have a preliminary design or not) to event-based reviews (the PDR will be held when there is a preliminary design to review).

This is better, but frequently leads to “Death by PowerPoint and SysML” reviews. These present much design detail, but there is little time to determine whether or not the design will meet the system’s key performance parameters. Such evidence of feasibility is generally desired, but is considered as an optional appendix and not a project deliverable. Thus, it is often neglected when budgets are tight, and when contractual progress payments and award fees are based on producing a design and not evidence of its feasibility. As with schedule-based reviews, there will be a temptation at this point to consider the project innocent of risk until proven guilty, and to proceed into the later phases with a great deal of undiscovered risk, which will be much more expensive to address later.

In an ICSM evidence-based review, the feasibility evidence is a first-class deliverable. As such, its planning and preparation becomes subject to earned value management and is factored into progress payments.
Investments in feasibility evidence have been found to pay off significantly in development rework avoidance. In a regression analysis of 161 software projects, rework due to shortfalls in architecture and risk resolution evidence grew from 18% added effort for small (10,000 source lines of code) projects, to 91% added effort for very large (10 million SLOC) projects [Boehm-Valerdi-Honour, 2008].

The link between evidence-based and risk-based decision making is that shortfalls in evidence are uncertainties or probabilities of loss, and that the fundamental decision quantity for risk exposure is:

$$\text{Risk Exposure } RE = \text{Probability}(\text{Loss}) \times \text{Size}(\text{Loss}).$$

Up to now, we have been using “risk” as the key concept in reducing downstream losses in uncertain situations, but we should now point out that this rather negative viewpoint has a positive counterpart called Opportunity Management, in which

$$\text{Opportunity Exposure } OE = \text{Probability}(\text{Gain}) \times \text{Size}(\text{Gain}).$$

Actually, these two are the duals of each other, as a decision not to pursue an opportunity results in a loss of Opportunity Exposure, whose probability and size of loss can be considered to be a Risk Exposure. This will be elaborated upon in Chapter 12 on Risk-Opportunity Assessment and Control.

As seen in the paragraph above on growth of rework vs. system size, it is less risky to proceed with evidence shortfalls on a small project with a small cost of rework, than on a very large project with a very large cost of rework. Even on a large project, though, shortfalls in evidence are not necessarily reasons to terminate the project or to invest in more evidence. If the Opportunity Exposure OE is high and the window of opportunity is closing rapidly, the risk of delay is high and proceeding at least incrementally with a small amount of evidence can be the best decision.

Thus, we can see that Principle 4 brings all of the other principles together. It involves concerns with the stakeholders’ value propositions in making decisions as in Principle 1; with proceeding incrementally as in Principle 2; and with synchronizing and stabilizing the concurrent activity prescribed in Principle 3. We will elaborate on this at the end of this chapter, along with referencing later chapters in Part IV that elaborate on feasibility evidence planning, preparation, and review; that elaborate on risk assessment and risk control techniques; and that elaborate on how to produce key evidence elements such as cost and schedule estimates; business case analyses; and tradeoff analyses among quality attributes or –ilities.

## 4.1 Failure Story: The Unaffordable Requirement

In the early 1980s, a large government organization contracted with TRW to develop an ambitious information query and analysis system. The system would provide more than 1,000 users, spread across a large building complex, with powerful query and analysis capabilities for a large and dynamic database.

TRW and the customer specified the system using a classic sequential-engineering waterfall development model. Based largely on user need surveys, an oversimplified high-level performance analysis, and a short deadline for getting the To-Be-Determined out of the requirements specification, they fixed into the contract a requirement for a system response time of less than one second. It satisfied the System Requirements Review criteria of being unambiguous, testable, and free of design commitments.

Subsequently, the software architects found that subsecond performance could only be provided via a highly customized design that attempted to anticipate query patterns and cache copies of data so that each user’s likely data would be within one second’s reach (a 1980’s precursor of Google). The resulting hardware architecture had more than 25 super-midicomputers (an earlier term for a computer with performance and capacity between a minicomputer and a mainframe) busy caching data according to algorithms whose actual performance defied easy analysis. The scope and complexity of the hardware-software architecture brought the estimated cost of the system to nearly $100 million, driven primarily by the requirement for a one-second response time.

Faced with this unattractive prospect (far more than the customer’s budget for the system), the customer and developer decided to develop a prototype of the system’s user interface and representative capabilities to
The results showed that a four-second response time would satisfy users 90 percent of the time. A four-second response time, with special handling for high-priority transactions, dropped development costs closer to $30 million. Thus, the premature specification of a 1-second response time neglected the risk of creating an overexpensive and time-consuming system development. Fortunately, in this case, the only loss was the wasted effort on the expensive-system architecture and a 15-month delay in delivery (see Figure 4-1). More frequently, such rework is done only after the expensive full system is delivered and found still too slow and too expensive to operate.

![Graph showing cost against response time](image)

**Figure 4-1. Problems Encountered without Feasibility Evidence**

### 4.1.1 Problem Avoidance with Principle 4

Had the developers been required to deliver a Feasibility Evidence Description (FED) showing evidence of feasibility of the one-second response time, they would have run benchmarks on the best available commercial query systems, using representative user workloads, and would have found that the best that they could do was about a 2.5-second response time, even with some preprocessing to reduce query latency. They would have performed a top-level architecture analysis of custom solutions, and concluded that such 1-second solutions were in the $100 million cost range. They would have shared these results with the customer in advance of any key reviews, and found that the customer would prefer to explore the feasibility of a system with a commercially-supportable response time. They would have done user interface prototyping and found much earlier that the four-second response time was acceptable 90% of the time.

As some uncertainties still existed about the ability to address the remaining 10% of the queries, the customer and developer would have agreed to avoid repeating the risky specification of a fixed response time requirement. The customer and developer would instead define a range of desirable-to-acceptable response times, with an award fee provided for faster performance. They would also have agreed to reschedule the next milestone review to give the developer time and budget to present evidence of the most feasible solution available, using the savings over the prospect of a $100 million system development as rationale. This would have put the project on a more solid success track over a year before the actual project discovered and rebaselined itself, and without the significant expense that went into the unaffordable architecture definition.

For example, the customer and developer would negotiate response time ranges. They would agree on a range between a two-second response time as desirable, and a four-second response time as acceptable with
some two-second special cases. Next they would benchmark commercial system add-ons to validate their feasibility. Finally, they would present solution and feasibility evidence at AP milestone review. The result would be an acceptable solution with minimal delay.

### 4.1.2 Other Lessons Learned

As seen in Figure 4-1, the best architectural solution is a discontinuous function of the response time requirement level. Thus, a “Build it quick and tune it later” strategy can lead to big trouble, as the tuning to increasing workload is likely to run into an unachievable architecture-breaker barrier with extremely expensive rework involved in switching to a more scalable architecture. Similarly, attempts to automate the derivation of a system’s architecture from its functional requirements are likely to commit to an architecture with similar architecture-breaker discontinuities.

The cost of a system is not necessarily a function of its number of requirements. Changing the required response time by 1 character from 1 to 4 seconds in a 2000-page requirements specification reduced the cost by a factor of over 3. This is discussed further in Chapter 16 on Quality Attributes and Tradeoffs, which emphasizes that unlike functional requirements that have local impacts that additively influence system costs, quality attributes have system-wide impacts that multiplicatively influence system costs.

### 4.2 Success Story: CCPDS-R

A Principle 4 success story is the Command Center Processing and Display System Replacement (CCPDS-R), a project to re-engineer the command center aspects of the US early missile warning system. It covered not only the software but also the associated system engineering and computing hardware procurement. The software effort involved over 1 million lines of Ada code, across a family of three related user capabilities. The developer was TRW; the customer was the Air Force Electronic Systems Center (ESC); the users were the U.S. Space Command, the U.S. Strategic Command, the U.S. National Command Authority, and all nuclear-capable Commanders in Chief. The core capability was developed on a 48-month fixed price contract between 1987 and 1991. While this was admittedly a long while ago in software time, the project closely mirrors current systems being developed in government and the private sector, and so is relevant as an example. A more detailed description of the CCPDS-R project is provided in Appendix D of [Royce, 1998].

The project had numerous high risk elements. One was the extremely high dependability requirements for a system of this nature. Others were the ability to re-engineer the sensor interfaces, the commander situation assessment and decision-aid displays, and the critical algorithms in the application. Software infrastructure challenges included distributed processing using Ada tasking and the ability to satisfy critical-decision-window performance requirements. Many of these aspects underwent considerable change during the development process. The project involved 75 software personnel, most of whom had training in Ada programming but had not applied it to real projects.

CCPDS-R used standard Department of Defense (DoD) acquisition procedures, including a fixed-price contract and the documentation-intensive DoD-STD-2167A software development standards. However, by creatively reinterpreting the DoD standards, processes, and contracting mechanisms, USAF/ESC and TRW were able to perform with agility, deliver on budget and on schedule, fully satisfy their users, and receive Air Force awards for outstanding performance.

#### 4.2.1 CCPDS-R Evidence-Based Decision Milestones

The DoD acquisition standards were acknowledged, but their milestone content was redefined to reflect the stakeholders’ success conditions. The usual DoD-STD-2167A Preliminary Design Review (PDR) to review paper documents and briefing charts around Month 6 was replaced by a PDR at Month 14 that demonstrated working software for all the high-risk areas, particularly the network operating system, the message-passing middleware and the graphic user interface (GUI) software. The PDR also reviewed the completeness,
consistency, and traceability of all of the Ada package interface specifications, as verified by the Rational Ada compiler and R-1000 toolset. Thus, a great deal of system integration was done before the software was developed.

TRW invested significant resources into a package of message-passing middleware that handled much of the Ada tasking and concurrency management, and provided message ports to accommodate the insertion of sequential Ada packages for the various CCPDS-R application capabilities. For pre-PDR performance validation, simulators of these functions could be inserted and executed to determine system performance and real-time characteristics. Thus, not only software interfaces, but also system performance could be validated prior to code development, and stubs could be written to provide representative module inputs and outputs for unit and integration testing. Simulators of external sensors and communications inputs and outputs were developed in advance to support continuous testing. Also, automated document generators were developed to satisfy the contractual needs for documentation.

Evidence of achievable software productivity was provided via a well-calibrated cost and schedule estimation model, in this case an Ada version of the Constructive Cost Model (Ada COCOMO), which was available for CCPDS-R. It was used to help developers, customers, and users better understand how much functional capability could be developed within an available budget and schedule, given the personnel, tools, processes, and infrastructure available to the project. Another major advantage of the Ada COCOMO cost/performance tradeoff analyses was to determine and enable the savings achieved via reuse across the three different installations and user communities.

Since the CCPDS-R plans and specifications were machine processable, the project was able to track progress and change at a very detailed level. This enabled the developers to anticipate potential downstream problems and largely handle them via customer collaboration and early fixes, rather than delayed problem discovery and expensive technical contract-negotiation fixes. Figure 4.2 shows one such metrics-tracking result: the cost of making CCPDS-R changes as a function of time.

For CCPDS-R, the message-passing middleware and modular applications design enabled the project to be highly agile in responding to change, as reflected in the low growth in cost of change shown in Figure 4-1. Further, the project’s advance work in determining and planning to accommodate the commonalities and variabilities across the three user communities and installations enabled significant savings in software reuse across the three related CCPDS-R system versions.

### 4.2.2 Other Innovative CCPDS-R Practices

USAF/ESC and TRW agreed that the contract award fee for good performance would not just go into the TRW corporate profit coffers. Instead, a significant part was set aside for individual project performer bonuses. This not only enhanced motivation and teamwork, but made the CCPDS-R project personnel turnover the lowest in TRW large-project history.

![Cost of changes vs. Time: CCPDS-R](image)
The architecture of the system was organized around the performers’ skill levels. In particular, previous project experience at TRW and elsewhere had shown high risks of having inexperienced personnel deal with concurrency and Ada tasking constructs. For CCPDS-R, the concurrency architecture and Ada tasking parts of the software were done by experienced Ada developers. The junior programmers were given sequential modules to develop, while being trained by TRW in Ada tasking and concurrency skills for the future.

4.3 Feasibility Evidence as a First-Class Deliverable

The risks of proceeding into development without evidence of feasibility are clear from the Unaffordable Requirement failure story in Section 4.1, and from numerous other failed projects. Other good examples are the Master Net project in Section 2.1 and the Total-Commitment approach to agent-based RPVs in Section 3.1. Thus, it is important to treat it as a first-class project deliverable, and not as an optional appendix to be dropped at the first budget or schedule crunch.

4.3.1 What Feasibility Evidence Is

Figure 4-3 repeats Figure 6 in the ICSM Distilled overview as the content of a Feasibility Evidence Description (FED). Chapter 11 includes a FED Data Item Description (DID) for use in outsourcing contracts, and risk-based guidance and checklists for tailoring it up from an extremely simple starting point as a collaborative activity between the customer and developer.

Once the tailored DID is accepted on the contract, its evidence content becomes a first-class deliverable. There should be a plan and a budget for generating its content, and mechanisms such as periodic reviews and an earned value management capability or an agile burndown equivalent for tracking its progress with respect to its plan. As additional understanding of the project risks accumulates, the items in the DID can be rebaselined up or down. A portion of the contract’s award fee should be allocated to the successful development and independent review of the resulting feasibility evidence. Chapter 15 on Agreements, Contracts, and Incentives provides further context for the use of the DID.

<table>
<thead>
<tr>
<th>Feasibility Evidence Description Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evidence <em>provided by developer</em> and <em>validated by independent experts</em> that if the system is built to the specified architecture, it will:</td>
</tr>
<tr>
<td>Satisfy the requirements: capability, interfaces, level of service, and evolution</td>
</tr>
<tr>
<td>Support the operational concept</td>
</tr>
<tr>
<td>Be buildable within the budgets and schedules in the plan</td>
</tr>
<tr>
<td>Generate a viable return on investment</td>
</tr>
<tr>
<td>Generate satisfactory outcomes for all of the success-critical stakeholders</td>
</tr>
<tr>
<td>Resolve all major risks by treating shortfalls in evidence as risks and covering them by risk management plans</td>
</tr>
<tr>
<td>Serve as basis for stakeholders’ commitment to proceed</td>
</tr>
</tbody>
</table>

Figure 4-3. Feasibility Evidence Description (FED) Content

4.3.2 What Feasibility Evidence Development Isn’t

Feasibility evidence development is not Evidence Appreciation. Frequently, systems engineers involved in developing models and simulations to evaluate feasibility become so engrossed in the elegance and detail of
their models that they do not complete them until after the key decision milestone for which they are needed is past. As with many other systems engineering and development “how much is enough” questions, the best way to address this is to balance the risks of doing too little evidence development vs. the risks of doing too much.

Some key drivers in this regard are the project’s size and criticality, which point toward having more evidence; and the project’s rate of requirements and technology change; which point away from generating a lot of evidence that will quickly become obsolete. These decision drivers are quantified in Figure 4-4 below.

Also, obtaining evidence isn’t equivalent to generating evidence. Often, a well-prepared phone call to a representative previous-project user of a prospective COTS product being evaluated for project use will produce superior evidence of feasibility or infeasibility than will weeks of COTS product exercise. As a first-class deliverable, the FED’s development should be preceded by careful planning and evaluation of alternative ways of obtaining evidence.

Feasibility evidence development is also not Analysis Paralysis. Often, COTS or cloud services evaluations become caught in a delay loop in which a new vendor announcement becomes an excuse to delay making a decision until the actual glories of the vendor announcement are available and can be evaluated. If users need new capabilities soon or the product being developed has a short market window, the project can’t afford to wait. At best, it should do a quick assessment of how much of the announcement is likely to be vaporware, and how the system could be architected to accommodate the new COTS product or service if and when its glories become reality.

4.3.3 How Much Feasibility Evidence Development is Enough?

Size, criticality, and volatility are key decision drivers for focusing on agile or architected approaches. But critical questions remain about how much architecting and feasibility evidence development is enough for a particular project. Here we provide a quantitative approach that has helped projects address this question. It extends the ROI of investments in feasibility evidence described in [Boehm-Valerdi-Honour, 2008].
The graphs show the results of a risk-driven “how much feasibility evidence is enough” analysis, based on the COCOMO II Architecture and Risk Resolution (RESL) factor. This factor was calibrated along with 22 others to 161 project data points. It relates the amount of extra rework effort on a project to the percent of project effort devoted to software-intensive system architecting and feasibility evidence development. The analysis indicated that the amount of rework was an exponential function of project size.

A small (10 thousand equivalent source lines of code, or KSLOC) project could fairly easily adapt its architecture to rapid change via refactoring or its equivalent, with a rework penalty of 14% between minimal and extremely thorough architecture and risk resolution. However, a very large (10,000 KSLOC) project would incur a corresponding rework penalty of 91%, covering such effort sources as integration rework due to large-component interface incompatibilities and critical performance shortfalls.

Actually, the RESL factor includes several other architecture-related attributes besides the amount of architecting investment, such as available personnel capabilities, architecting support tools, and the degree of architectural risks requiring resolution. Also, the analysis assumes that the other COCOMO II cost drivers do not affect the project outcomes.

The effects of rapid change (volatility) and high assurance (criticality) on the sweet spots are shown in the right hand graph. Here, the solid lines represent the average-case cost of rework, architecting, and total cost for a 100-KSLOC project as shown at the left. The dotted lines show the effect on the cost of architecting and total cost if rapid change adds 50% to the cost of architecture and risk resolution. Quantitatively, this moves the sweet spot from roughly 20% to 10% of effective architecture investment (but actually 15% due to the 50% cost penalty). Thus, high investments in architecture and other documentation do not have a positive return on investment due to the high costs of documentation rework for rapid-change adaptation.

The dashed lines at the right represent a conservative analysis of the effects of failure cost of architecting shortfalls on the project’s effective business cost and architecting sweet spot. It assumes that the costs of architecture shortfalls are not only added rework, but also losses to the organization’s operational effectiveness and productivity. These are conservatively assumed to add 50% to the project-rework cost of architecture.
shortfalls to the organization. In most cases for high-assurance systems, the added cost would be considerably higher.

Quantitatively, this moves the sweet spot from roughly 20% to over 30% as the most cost-effective investment in architecting for a 100-KSLOC project. It is good to note that the “sweet spots” are actually relatively flat “sweet regions” extending 5-10% to the left and right of the “sweet spots.” However, moving to the edges of a sweet region increases the risk of significant losses if some project assumptions turn out to be optimistic.

Again, the effects of other factors may affect the location of a given project’s “how much evidence is enough” sweet spot. A good cross-check is to use the Constructive Systems Engineering Cost Model, COSYSMO [Valerdi, 2008], to estimate the project’s amount of needed systems engineering effort. A third approach is to use the risk-based decision heuristic of balancing the risks of doing too little evidence generation with the risks of doing too much (the balance between “Look before you leap” and “He who hesitates is lost.”).

4.4 How Much of Anything Is Enough?

The risk-based decision heuristic of balancing the risks of doing too little evidence generation with the risks of doing too much can be applied to most decisions involved in system definition, development, and evolution. How much system scoping, planning, prototyping, COTS evaluation, requirements detail, spare capacity, fault tolerance, safety, security, environmental protection, documenting, configuration management, quality assurance, peer reviewing, testing, use of formal methods, and feasibility evidence is enough? The best answer can generally be found by considering and balancing the risks of doing too little with the risks of doing too much. And the answer will generally not be the same for all parts of the system. The higher-risk parts of the system will need more attention to detail than the lower-risk parts, in order to reduce both the probability and size of loss involved in getting it wrong.

If there is any meta-principle underlying the four principles and other practices in the ICSM, it is this Meta-Principle of Balance: Balancing the risk of doing too little and the risk of doing too much will generally find a middle course sweet spot that is about the best you can do.

Of course, there is nothing new about it. It’s what Herb Simon was talking about in preferring satisficing to optimizing; what Aristotle was talking about with the Golden Mean; what the Confucians talk about with the Doctrine of the Mean, and want the Buddhists talk about with the Middle Way. With all of these advocates, it seems like a pretty good underlying meta-principle.

4.5 Summing Up the Principles

The definition of the spiral principles has gone through several iterations, including for example a joint effort by USC and the CMU Software Engineering Institute that resulted in the definitions of six Spiral Invariants and six Hazardous Spiral Look-Alikes (Boehm-Hansen, 2001). Subsequent continuing experience in applying the Spiral Invariants led to their consolidation into the four principles just explained in Chapters 1-4.

Table 4-1 summarizes how the four principles reinforce each other to more rapidly and cost-effectively deliver value to a system’s stakeholders. The rows on Table 4-1 show how the application of each principle improves the cost-effectiveness of applying the other three principles, in terms of value delivered to the stakeholders across the system’s evolving life cycle. The columns show how each principle’s cost-effectiveness in improved by the other three principles.
For example, Row 1 begins by showing how applying Principle 1 on Stakeholder Value-Based System Definition and Evolution improves the cost-effectiveness of applying Principle 2 on Incremental Commitment and Accountability by better identifying the system’s success-critical stakeholders and enabling them to more rapidly and fully understand each other’s value propositions, and to work out better decisions on which stakeholders are best suited to have primary and secondary responsibilities and authority to perform the system’s life cycle definition, development, and evolution functions.

Similarly, Column 1 begins by showing how applying Principle 2 improves the cost-effectiveness of applying Principle 1, in that the incremental commitment of the stakeholders to their responsibilities and authority as the life cycle proceeds, leads to more rapid and incremental mutual understanding and buildup of mutual trust in each other, which leads to more agility in responding to the continuing flow of changes impacting the system’s definition and evolution.

This strengthening of Principle 1 is what enables it to improve the cost-effectiveness of Principle 3 shown in Row 1, in enabling more rapid convergence on mutually satisfactory system solutions. Thus, Principle 3 is strengthening Principle 1 in Column 1 by speeding up the growth of stakeholders’ mutual understanding of each others’ value propositions … and so on with Principle 4 and the other parts of Table 4-1. And underlying it all is the Meta-Principle of Balance: Balancing the risk of doing too little and the risk of doing too much will generally find a middle course sweet spot that is about the best you can do.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>↓Applying</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Stakeholder Value-Based System Definition and Evolution</td>
<td>Understands other stakeholders’ value propositions</td>
<td>Rapidly converges on mutually satisfactory solutions</td>
<td>Focuses evidence generation on highest-value issues</td>
<td>Focuses evidence generation on highest-value issues</td>
</tr>
<tr>
<td>2. Incremental Commitment and Accountability</td>
<td>Builds trust in other stakeholders, increases agility in responding to change</td>
<td>Triage of changes enables consistent, feasible near-term and longer-term decisions</td>
<td>Ensures adequate resources for evidence development</td>
<td>Ensures adequate resources for evidence development</td>
</tr>
<tr>
<td>3. Concurrent Multi-discipline System Definition and Evolution</td>
<td>Speeds up understanding of other stakeholders’ value propositions</td>
<td>Enables more rapid and continuing focus on highest-level issues</td>
<td>Enables more cost-effective, timely generation of evidence and the resulting decisions</td>
<td>Enables more cost-effective, timely generation of evidence and the resulting decisions</td>
</tr>
<tr>
<td>4. Evidence and Risk-Based Decision Making</td>
<td>Provides stakeholders with evidence of value</td>
<td>Avoids commitments to infeasible solutions</td>
<td>Synchronizes and stabilizes concurrency</td>
<td>Synchronizes and stabilizes concurrency</td>
</tr>
</tbody>
</table>

The Meta-Principle of Balance:

Balancing the risk of doing too little and the risk of doing too much will generally find a middle course sweet spot that is about the best you can do.