The Spiral Model as a Tool for Evolutionary Acquisition

Dr. Barry Boehm
University of Southern California
Center for Software Engineering

Wilfred J. Hansen
Software Engineering Institute

Recent Department of Defence policy embodied in the new 5000.1 and 5000.2 series of acquisition regulations strongly emphasizes evolutionary acquisition and the use of spiral development for software. Spiral development has been used successfully in many commercial systems and in a good number of defense systems. But some ambiguities in previous spiral model definitions has also led to a good number of unsuccessful projects adopting "hazardous spiral look-alikes." This paper provides clearer definitions of a set of six Spiral Model Essentials or critical success factors for spiral development. It illustrates each with examples, identifies the hazardous look-alikes to avoid, and provides guidelines for using the spiral model in support of evolutionary acquisition.

Since its original publication [1], the spiral development model diagrammed in Figure 1 has been used successfully in many defense and commercial projects. To extend this base of success, the Department of Defense (DoD) has recently rewritten the defense acquisition regulations to incorporate "evolutionary acquisition," an acquisition strategy designed to mesh well with spiral development. In particular, DoD Instruction 5000.2 subdivides acquisition [2]:

> "There are two ... approaches, evolutionary and single step to full capability. An evolutionary approach is preferred. … [In this] approach, the ultimate capability delivered to the user is divided into two or more blocks, with increasing increments of capability." (p. 20)

Here, a block corresponds to a single product release. The text goes on to specify the use of spiral development within blocks:

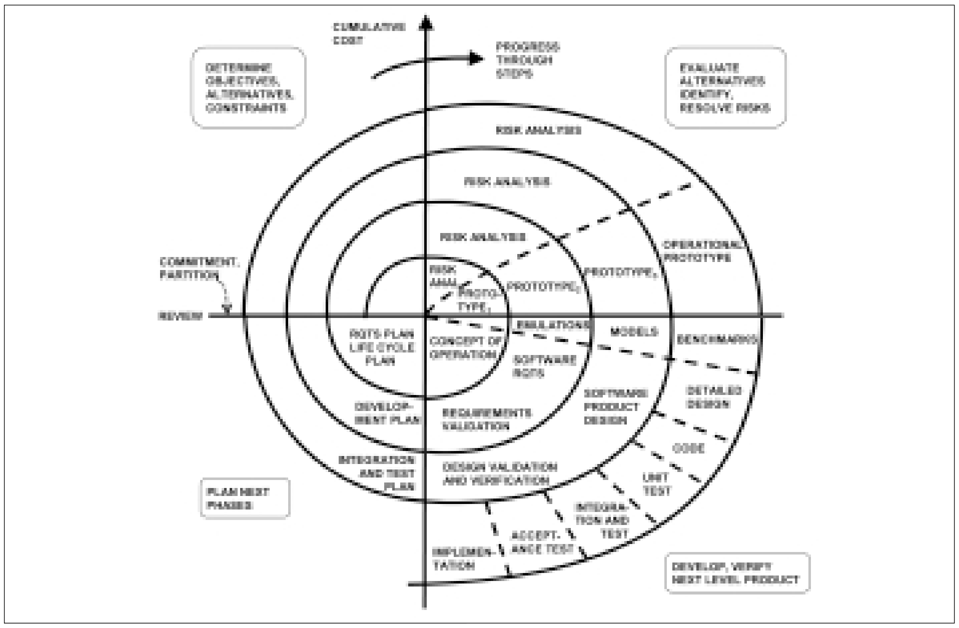
> "For both the evolutionary and single-step approaches, software development shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development." (p. 20)

Given this reliance on the spiral development model, an in-depth definition is appropriate. Two recent workshops provided one.

The University of Southern California (USC) Center for Software Engineering and the Carnegie Mellon University Software Engineering Institute held two workshops last year to study spiral development and identify a set of critical success factors and recommended approaches. Their results appear in two reports, [3, 4] and are available on the workshop Web site www.sei.emu.edu/cbs/spiral2000

The first author's presentations at these workshops defined spiral development and are followed below. The definition was first converted to a report [5], where details, suggestions, and further references can be found. Additionally, a follow-on article appearing in a later CrossTalk issue, will address the relationships among spiral development, evolutionary acquisition, and the Integrated Capability Maturity Model.

Figure 1: *Original Spiral Development Diagram*



Spiral Development Definition and Context

We can begin with a high-level definition of the spiral development model:

> The spiral development model is a *risk-driven process model* generator that is used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a *cyclic* approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of *anchor point milestones* for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

The highlighted terms deserve further explanation:

- *Risks* are situations or possible events that can cause a project to fail to meet its goals. They range in impact from trivial to fatal and in likelihood from certain to improbable. Since risk considerations dictate the path a development must take, it is important that those risks be cataloged candidly and completely. See the references for taxonomy of risks [6] and a method for identifying them [7].
- A *process model* answers two main questions: What should be done next? How long should it continue? Under the spiral model the answers to these questions are driven by risk considerations and

vary from project to project and sometimes from one spiral cycle to the next. Each choice of answers generates a different process model.

- The *cyclic* nature of the spiral model is illustrated in Figure 1. Rather than develop the completed product in one step, multiple cycles are performed with each taking steps calculated to reduce the most significant remaining risks.
- Each *anchor point milestone* is a specific combination of artifacts and conditions that must be attained at some point. The sequence of three anchor point milestones — "LCO," "LCA," and "ICO" — is defined in Spiral Essential 5 (to be discussed). These milestones impel the project toward completion and offer a means to compare progress between one project and another.

Many aspects of spiral development are omitted in the above definition. The remainder of this paper expands the definition by describing six essential aspects that every proper spiral process must exhibit. The essentials are sketched in Figure 2. Each subsequent section describes a Spiral Essential, the critical success factor, reasons why it is necessary, and the variant process models it allows. Examples are given. Other process models that are precluded by the Spiral Essential are described. Because these may seem to be instances of the spiral model, but lack necessary essentials and thus risk failure, they are called "hazardous spiral look-alikes."

Spiral Essential 1: Concurrent Determination of Key Artifacts (Operational Concept, Requirements, Plans, Design, Code)

For a successful spiral effort, it is vital to determine balanced portions of certain key artifacts concurrently and not sequentially. These key artifacts are the operational concept, the system and software requirements, the plans, the system and software architecture and design, and the code components, including COTS, reused components, prototypes, success-critical components, and algorithms. Ignoring this Essential by sequentially determining the key artifacts will prematurely overconstrain the project, and often extinguish the possibility of developing a product satisfactory to the stakeholders.

Variants: Within the constraints of this Essential, variation is possible in the product and process internals of the concurrent engineering activity. For a low technology, interoperability-critical system, the initial spiral products will be requirements-intensive. For a high technology, more standalone system, the initial spiral products will be prototype-code intensive. Also, the Essential does not dictate the number of mini-cycles (e.g., individual prototypes for COTS, algorithms, or user-interface risks) within a given spiral cycle.

Example: One-Second Response Time

Examples of failure due to omission of this Essential include premature commitments to hardware platforms, incompatible combinations of COTS components, and requirements whose achievability has not been validated. For instance, in the early 1980s, a large government organization contracted with TRW to develop an ambitious information system for more than a thousand users. This system was to be distributed across a campus and offer powerful query and analysis access to a large and dynamic database. Based largely on user need surveys and an oversimplified high-level performance analysis, TRW and the customer fixed into the contract a requirement for a system response time of less than one second.

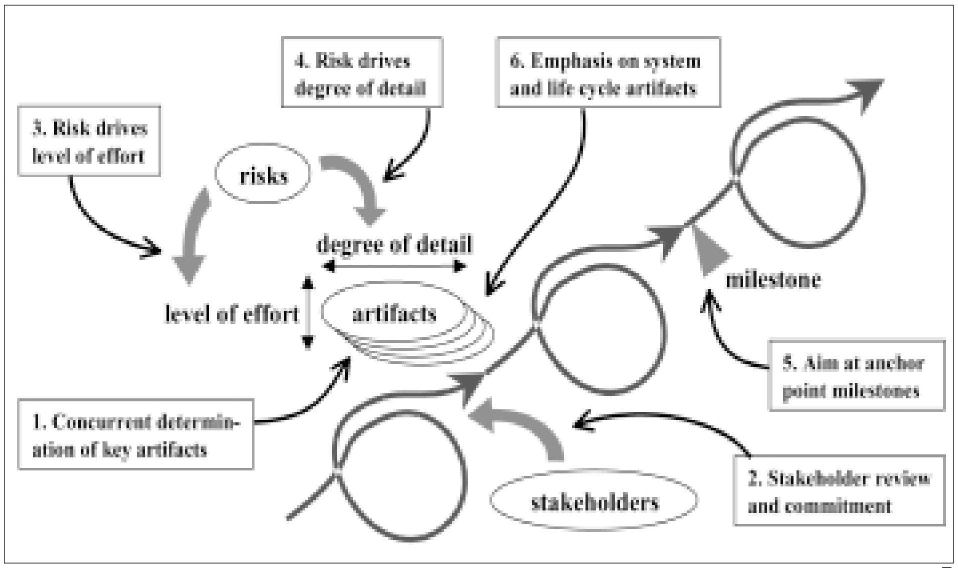
Two thousand pages of requirements later, the software architects found that subsecond performance could only be provided via a highly customized design that attempted to cache data and anticipate query patterns to be able to respond to each user within one second. The resulting hardware architecture had more than 25 super-minicomputers caching data according to algorithms whose actual performance defied easy analysis. The estimated cost was $100 million; see the upper arc in Figure 3 (See page 6).

Faced with this exorbitant cost, the customer and developer decided to develop and user-test a prototype. The results showed a four-second response time with 90 percent user satisfaction. This lower performance could be achieved with a modified client-server architecture, cutting development costs to $30 million as shown by the lower arc in the Figure 3 [8]. Concurrent prototyping would have saved two years' time and large-project effort.

Hazardous Spiral Look-Alike: Violation of Waterfall Assumptions

Essential 1 excludes the use of an incremental sequence of waterfall developments in the common case where there is a high risk of violating the assumptions underlying the waterfall model. These assumptions are that the requirements are *pre-specifiable*, *slowly changing*, and satisfactory to all stakeholders, and that a well understood *architecture* can meet these requirements. These assumptions must be met by a project if the waterfall model is to succeed. If all are true, then it is a project risk *not* to specify the requirements: the spiral-dictated risk analysis results in a waterfall approach for this project. If any assumption is false, then a waterfall approach will commit the project to troublesome assumptions and requirements

Figure 2: *Pictorial Sketch of the Six Spiral Essentials*


4. Risk drives degree of detail
6. Emphasis on system and life cycle artifacts
3. Risk drives level of effort
risks
degree of detail
milestone
level of effort
artifacts
5. Aim at anchor point milestones
1. Concurrent determination of key artifacts
2. Stakeholder review and commitment
stakeholders

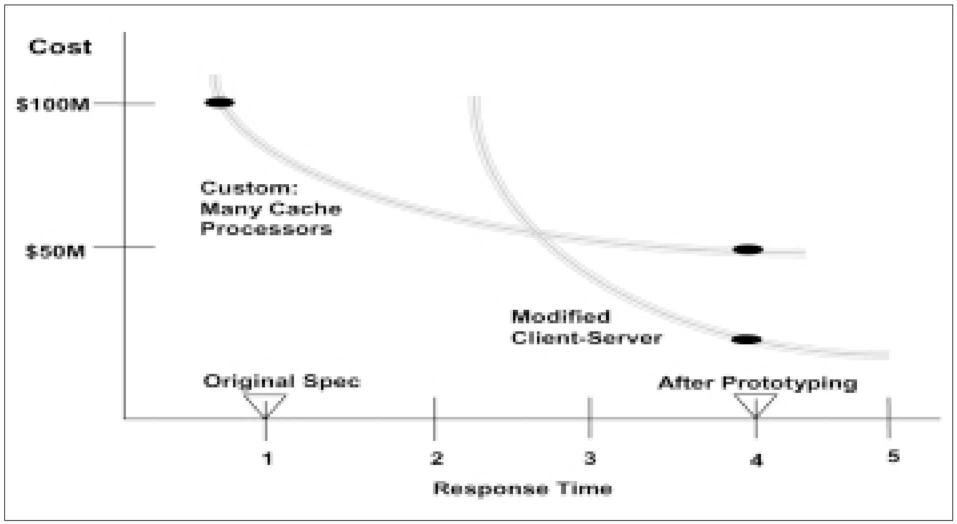


Figure 3: *Two System Designs: Cost vs. Response Time*

mismatches. Here are typical cases that violate waterfall assumptions:

- Requirements are not generally *pre-specifiable* for new user-interactive systems, because of the IKIWISI syndrome. When asked for their required screen layout for a new decision-support system, users will generally say, "I can't tell you, but I'll know it when I see it (IKIWISI)." In such cases, a concurrent prototyping/requirements/architecture approach is necessary.
- *Rapidly changing* requirements are well illustrated by electronic commerce projects, where the volatility of technology and the marketplace is high. The time it takes to write detailed requirements is not a good investment of the scarce time-to-market available when it is likely the requirements will change more than once downstream.
- The *architecture* and its implications were the downfall of the one-second response time example.

Spiral Essential 2: Each Cycle Does Objectives, Constraints, Alternatives, Risks, Review, and Commitment to Proceed

Spiral Essential 2 identifies the activities that need to be done in each spiral cycle. These include consideration of critical stakeholder objectives and constraints, elaboration and evaluation of project and process alternatives for achieving the objectives subject to the constraints, identification and resolution of risks attendant on choices of alternative solutions, and stakeholders' review and commitment to proceed based on satisfaction of their critical objectives and constraints. If all of these are not considered, the project may be prematurely committed to alternatives that are either unacceptable to key stakeholders or overly risky.

Variants: Spiral Essential 2 does not mandate particular generic choices of risk resolution techniques, although guidelines are available [9]. Nor does this Essential mandate particular levels of effort for the activities performed during each cycle. Levels must be balanced between the risks of learning too little and the risks of wasting time and effort gathering marginally useful information.

Example: Windows-Only COTS

Ignoring Essential 2 can lead to wasted effort in elaborating an alternative that could have been shown earlier to be unsatisfactory. One of the current USC digital library projects is developing a Web-based viewer for oversized artifacts (e.g., newspapers, large images). The initial prototype featured a tremendously powerful and high-speed viewing capability, based on a COTS product called ER Mapper. The initial project review approved selection of this COTS product, even though it only ran well on Windows platforms, and the Library had significant Macintosh and UNIX user communities. This decision was based on initial indications that Mac and UNIX versions of ER Mapper would be available "soon." These indications proved unreliable, however, and the anticipated delay became quite lengthy. So after wasting considerable effort on ER Mapper, it was dropped in favor of a less powerful but fully portable COTS product, Mr. SID. The excess effort could have been avoided had the review team included stakeholders from the Mac and UNIX communities on campus who would have done the necessary investigations earlier.

Hazardous Spiral Look-Alike: Excluding Key Stakeholders

Essential 2 excludes the process model of organizing the project into sequential phases or cycles in which key stakeholders are excluded. These omissions are likely to cause critical risks to go undetected. Examples are excluding developers from system definition, excluding users from system construction, or excluding system maintainers from either definition or construction. Excluding developer participation in early cycles can lead to project commitments based on unrealistic assumptions about developer capabilities. Excluding users or maintainers from development cycles can lead to win/lose situations, which generally devolve into lose-lose situations.

Spiral Essential 3: Level of Effort Driven by Risk Considerations

Spiral Essential 3 dictates the use of risk considerations to answer the difficult questions concerning how much is enough of a given activity such as domain engineering, prototyping, testing, configuration management, and so on. The recommended approach is to evaluate Risk Exposure (RE), which is computed as Probability (Loss) • Size (Loss). There is risk of project error RE_{error} from doing too little effort and project delay RE_{delay} from doing too much. Ideally, the effort expended will be that which minimizes the sum $RE_{error} + RE_{delay}$. This approach applies to most activities that are undertaken in a spiral development.

Variants: The variants to be considered include the choice of methods used to pursue activities (e.g., MBASE/WinWin, Rational RUP, JAD, QFD, ESP) and the degree of detail of artifacts produced in each cycle. Another variant is an organization's choice of particular methods for risk assessment and management.

Example: Pre-Ship Testing

Risk considerations can help determine "how much testing is enough" before shipping a product. The more testing that is

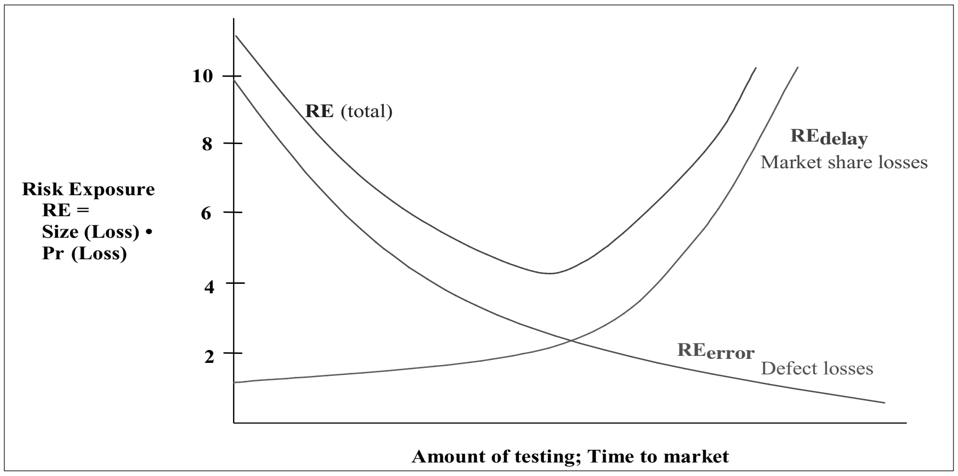


Figure 4: *Pre-Ship Test Risk Exposure*

done, the lower becomes RE_{error} due to defects, as discovered defects reduce both the size of loss due to defects and the probability that undiscovered defects still remain. However, the more time spent testing, the higher is RE_{delay} from losses due to both competitors entering the market and decreased profitability on the remaining market share. As shown in Figure 4, the sum of these risk exposures achieves a minimum at some intermediate level of testing. The location of this minimum-risk point in time will vary by type of organization. For example, it will be considerably shorter for a "dot.com" company than it will for a safety-critical product such as a nuclear power plant. Calculating the risk exposures also requires an organization to accumulate a fair amount of calibrated experience on the probabilities and size of losses as functions of test duration and delay in market entry.

Hazardous Spiral Look-Alikes: Risk Insensitivity

Hazardous spiral model look-alikes excluded by Essential 3 are:

- Risk-insensitive evolutionary development (e.g., neglecting scalability risks).
- Risk-insensitive incremental development (e.g., sub-optimizing during increment 1 with an ad hoc architecture that must be dropped or heavily reworked to accommodate future increments).
- Impeccable spiral plans with no commitment to managing the risks identified.

Spiral Essential 4: Degree of Detail Driven by Risk Considerations

Where Essential 3 circumscribes efforts, Essential 4 circumscribes the results of those efforts; it dictates that risk considerations determine the degree of detail of artifacts. This means, for example, that the traditional ideal of a complete, consistent, traceable, testable requirements specification is not a good idea for certain product components, such as a graphic user interface (GUI) or COTS interface. Here, the risk of precisely specifying screen layouts in advance of development involves a high probability of locking an awkward user interface into the development contract, while the risk of not specifying screen layouts is low, given the general availability of flexible GUI-builder tools. Even aiming for full consistency and testability can be risky, as it creates a pressure to prematurely specify decisions that would better be deferred (e.g., the form and content of exception reports). However, some risk patterns make it very important to have precise specifications, such as the risks of safety-critical interface mismatches between hardware and software components, or between a prime contractor's and a subcontractor's software.

This guideline shows when it is risky to over specify and under specify software features:

- If it's risky to *not* specify precisely, *DO specify* (e.g., hardware-software interface, prime-subcontractor interface).
- If it's risky to specify precisely, *DO NOT specify* (e.g., GUI layout, volatile COTS behavior).

Variants: Unconstrained by Essential 4 are the choices of representations for artifacts (SA/SD, UML, MBASE, formal specs, programming languages, …).

Example: Risk of Precise Specification

One editor specification required that every operation be available through a button on the window. As a result, the space available for viewing and editing became unusably small. The developer was precluded from moving some operations to menus because the GUI layout had been specified precisely at an early step. (Of course, given too much freedom programmers can develop very bad GUIs. Stakeholder review and prototype exercises are necessary to avoid such problems.)

Hazardous Spiral Look-Alikes: Insistence on Complete Specifications

It is often risky to undertake a spiral development project wherein complete specifications are pre-specified for *all* aspects. Aspects such as those described in the example should be left to be further defined during project exploratory phases.

Spiral Essential 5: Use Anchor Point Milestones LCO, LCA, IOC

A major difficulty of the original spiral model was its lack of intermediate milestones to serve as commitment points and progress checkpoints. This difficulty has been remedied by the development of a set of anchor point milestones:

LCO - Life Cycle Objectives - what should the system accomplish?

LCA - Life Cycle Architecture - what is the structure of the system?

IOC - Initial Operating Capability - the first released version.

(The artifacts for each are provided by an electronic process guide [10] and are also used by the Rational Unified Process.)

The focus of the LCO review is to ensure that at least one architecture choice is viable from a business perspective. The focus of the LCA review is to commit to a single detailed definition of the project. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan. The LCA milestone is particularly important, as its pass/fail criteria enables stakeholders to hold up projects attempting to proceed into evolutionary or incremental development without a life cycle architecture. Each milestone is a stakeholder commitment point: at LCO the stakeholders commit to support building architecture; at LCA they commit to support initial deployment; at IOC they commit to support operations. Together the anchor point milestones avoid analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, and useless systems.

Variants: One appropriate variant of Essential 5 is the number of spiral cycles between anchor points. Another possible variant is to merge anchor points. In particular, a project using a mature and appropriately scalable fourth generation language (4GL) or product line framework will have already determined its architecture by its LCO milestone, enabling the LCO and LCA milestones to be merged.

Example: Stud Poker Analogy

A valuable aspect of the spiral model is its ability to support incremental commitment of corporate resources rather than requiring a large outlay of resources to the project before its success prospects are well understood. Funding a spiral development can thus be likened to the game of stud poker. In that game, you put a couple of chips in the pot and receive two cards, one hidden and one exposed. If your cards don't promise a winning outcome, you can drop out without a great loss. This corresponds to cancelling a project at or before LCO. If your two cards are both aces, you will probably bet on your prospects aggressively (or less so if you see aces among other players' exposed cards). Dropping out of the second or third round of betting corresponds to cancelling at or before LCA. In any case, based on information available, you can decide during each round whether it's worth putting more chips in the pot to buy more information or whether it's better not to pursue this particular deal or project.

Hazardous Look-Alike:
Evolutionary Development Without Life Cycle Architecture

The LCO and LCA milestones' pass-fail criteria emphasize that the system's architecture must support not just the initial increment's requirements, but also the system's evolutionary life-cycle requirements. This avoids the hazardous spiral look-alike of an initial increment optimized to provide an impressive early system demonstration or limited release, but without the architecture to support full-system requirements for security, fault-tolerance, or scalability to large workloads. Other important considerations for LCA are that the initial release ensure continued key stakeholder participation, that the user organizations are flexible enough to adapt to the pace of system evolution, and that legacy-system replacement be well thought out. Ignoring these aspects lead to other hazardous spiral look-alike processes.

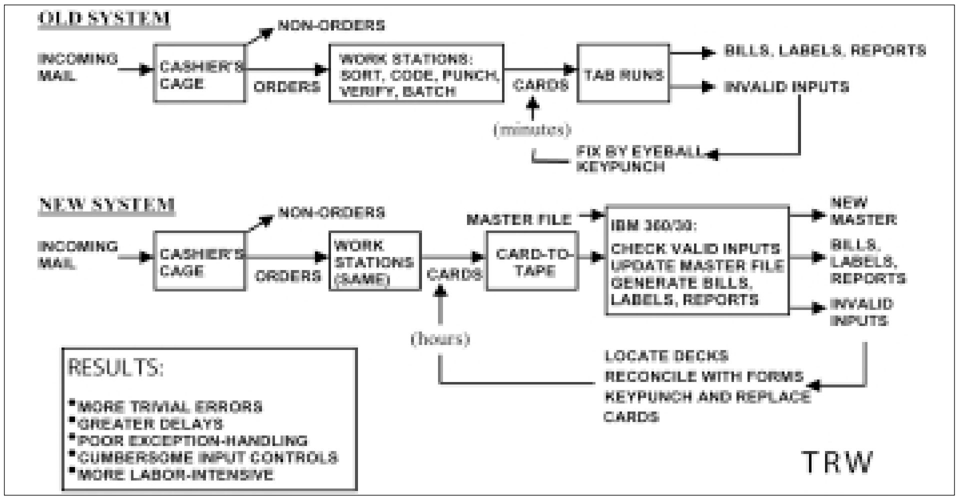


Figure 5: *Scientific American Order Processing*

Spiral Essential 6: Emphasis on System and Life Cycle Activities and Artifacts

Spiral Essential 6 emphasizes that spiral development of software-intensive systems needs to focus not just on software construction aspects, but also on overall system and life-cycle concerns. Will the product satisfy stakeholders? Will it meet cost and performance goals? Will it integrate with existing business practices? Will it adapt to organizational changes? Software developers are apt to fall into the oft-cited trap: "If your best tool is a hammer, the world you see is a collection of nails." Writing code may be a developer's forte, but it is as important to the project as are nails to a house.

Variants: The model's use of risk considerations to drive solutions makes it possible to tailor each spiral cycle to whatever mix of software and hardware, choice of capabilities, or degree of productization is appropriate.

Example: Order Processing

An example of failure to consider the whole system occurred with the Scientific American order processing system in Figure 5 [11]. Scientific American had hoped that computerizing the functions being performed on tabulator machines would reduce its subscription processing costs, errors, and delays. Rather than analyze the sources of these problems, the software house focused on the part of the problem having a software solution. The result was a batch-processing computer system whose long delays put extra strain on the clerical portion of the system that had been the major source of costs, errors, and delays in the first place. The software people looked for the part of the problem with a software solution (their "nail"), pounded it in with their software hammer, and left Scientific American worse off than when they started.

This kind of outcome would have resulted even if the software automating the tabulator-machine functions had been developed in a risk-driven cyclic approach. However, its Life Cycle Objectives milestone package would have failed its feasibility review, as it had no system-level business case demonstrating that the development of the software would lead to the desired reduction in costs, errors, and delays. Had a thorough business case analysis been done, it would have identified the need to reengineer the clerical business processes as well as to automate the manual tab runs.

Hazardous Spiral Look-Alikes:
Logic-Only OO Designs

Models excluded by Essential 6 include most published object-oriented analysis and design (OOA&D) methods, which are usually presented as abstract logical exercises independent of system performance or economic concerns. For example, in a recent survey of 16 OOA&D books, only six listed the word "performance" in their index, and only two listed "cost."

Using the Spiral Model for Evolutionary Acquisition

Both the February and September workshops had working groups on the relationships between spiral development and evolutionary acquisition. A primary conclusion was that the relationships differ across two major DoD acquisition sectors:

- Information systems, such as C4ISR systems, logistics systems, and management systems, in which spiral and evolutionary models coincide well.
- Software-intensive embedded hardware/software systems, in which the software aspects best follow a spiral approach, but the hardware aspects need to follow a more sequential approach to accommodate lead times for production facilities, production subcontracts, and long-lead critical component orders.

Even for embedded systems, however, spiral approaches can be helpful for synchronizing hardware and software processes, and for determining when to apply an evolutionary, incremental, or single-step acquisition strategy. For example, Xerox's time-to-market process uses the spiral anchor point milestones as hardware/software synchronization points for its printer business line [12]. Rechtin and Maier adopt a similar approach in their book, the Art of Systems Architecting [13].

Development strategy	Define all requirements first?	Multiple development cycles?	Distribute interim software?
Once-Through (Waterfall)	Yes	No	No
Incremental (Preplanned Product Improvement)	Yes	Yes	Maybe
Evolutionary	No	Yes	Yes

Table 1: *Evolutionary Acquisition Distinctions [14]*

A good example of the use of a risk-driven spiral approach to determine a preferred software/system acquisition strategy was originally developed for DoD's MIL-STD-498, and subsequently incorporated in IEEE/EIA 12207 [14]. This approach distinguishes among single-step (once-through or waterfall), incremental, and evolutionary acquisition processes as shown in Table 1. Thus, evolutionary acquisition avoids defining all requirements first, and proceeds in multiple development cycles, some of which involve distribution and usage of initial and intermediate operational capabilities.

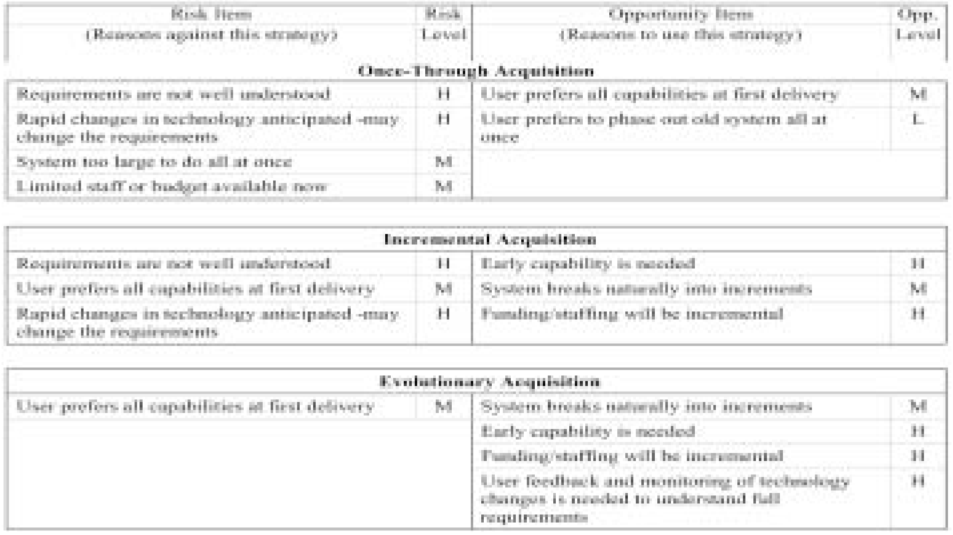
Table 2 shows how a spiral risk analysis can be performed during early integrated product and process definition cycles to select the most appropriate acquisition process for a system. The example shown in Table 2 is for a fairly large, high-technology C4ISR system. For such a system, the high risks of poorly-understood requirements and rapid technology changes push the decision away from once-through or incremental acquisition, while the needs for an early capability and for user feedback on full requirements push the decision toward evolutionary acquisition. If the system had been a small, lower-technology embedded system where all capabilities are needed at first delivery, the risk and opportunity factors would push the decision away from evolutionary and incremental acquisition toward once-through or single-step acquisition.

Conclusion

This paper has defined the spiral development model as a risk-driven process model generator with cyclic process execution and a set of three anchor point milestones. The definition was sharpened by presenting a set of six "essential" attributes; that is, six attributes which every spiral development process must incorporate. These essentials are summarized in Table 3 (See page 10). Omission of each of these gives rise to process models that are cyclic or iterative, but are not examples of spiral development. These are called "hazardous spiral look-alikes." Each was described and pilloried as part of describing the Essential it violates.

Spiral development works fairly seamlessly with evolutionary acquisition of information systems. For evolutionary acquisition of software-intensive embedded hardware-software systems, a mix of spiral and sequential processes is often needed. Even here, however, the spiral anchor points and risk-driven process selection approach are useful for determining how best to synchronize the hardware and software processes.◆

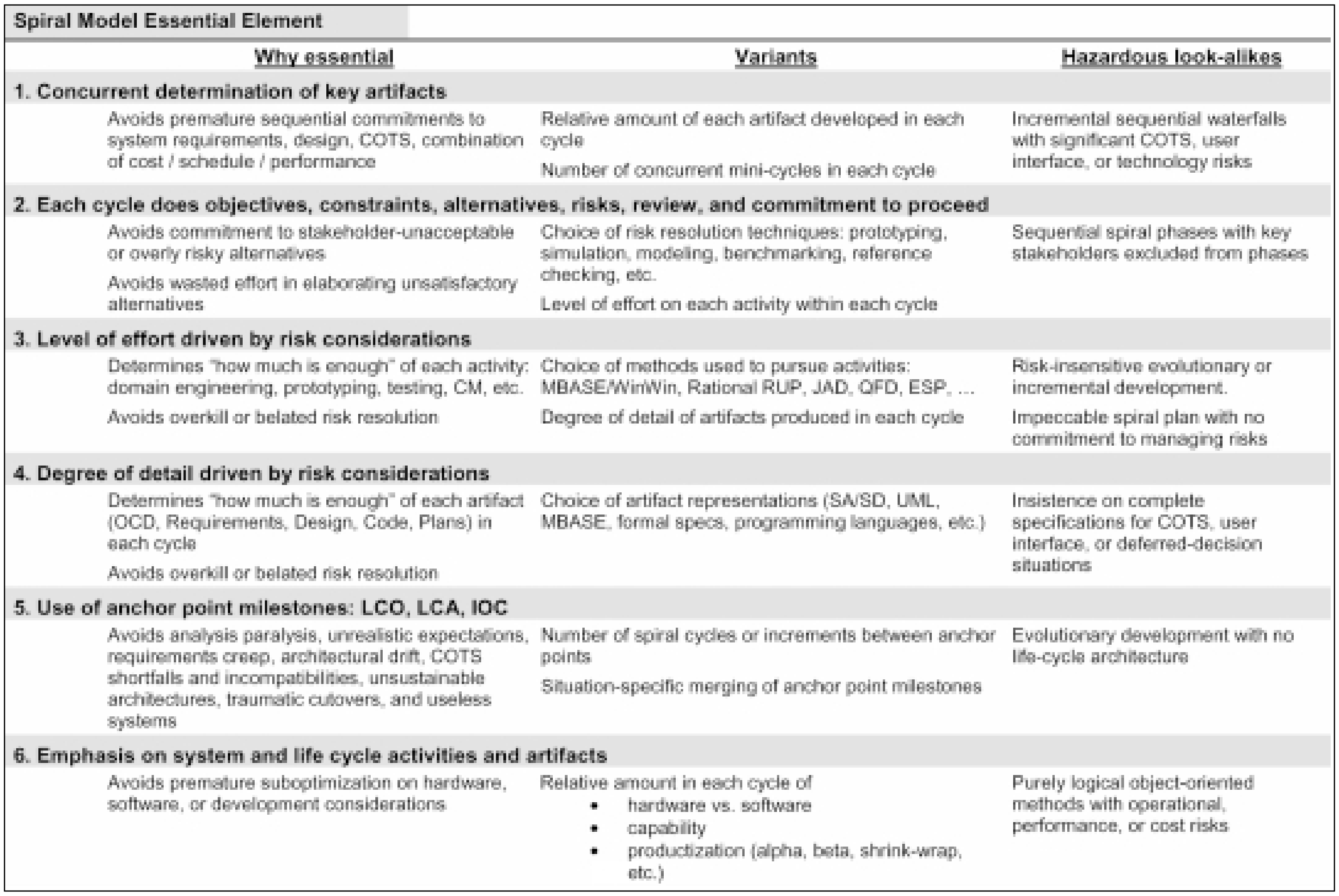
Table 2: *Spiral Risk Analysis for Process Selection [14]*



Risk Item (Reasons against this strategy)	Risk Level	Opportunity Item (Reasons to use this strategy)	Opp. Level
Once-Through Acquisition			
Requirements are not well understood	H	User prefers all capabilities at first delivery	M
Rapid changes in technology anticipated -may change the requirements	H	User prefers to phase out old system all at once	L
System too large to do all at once	M		
Limited staff or budget available now	M		
Incremental Acquisition			
Requirements are not well understood	H	Early capability is needed	H
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
Rapid changes in technology anticipated -may change the requirements	H	Funding/staffing will be incremental	H
Evolutionary Acquisition			
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
		Early capability is needed	H
		Funding/staffing will be incremental	H
		User feedback and monitoring of technology changes is needed to understand full requirements	H

References

1. Boehm, B., A Spiral Model of Software Development and Enhancement, *Computer,* May 1988, pp. 61-72.
2. Department of Defense Instruction 5000.2, Operation of the Defense



Spiral Model Essential Element	Why essential	Variants	Hazardous look-alikes
1. Concurrent determination of key artifacts	Avoids premature sequential commitments to system requirements, design, COTS, combination of cost / schedule / performance	Relative amount of each artifact developed in each cycle Number of concurrent mini-cycles in each cycle	Incremental sequential waterfalls with significant COTS, user interface, or technology risks
2. Each cycle does objectives, constraints, alternatives, risks, review, and commitment to proceed	Avoids commitment to stakeholder-unacceptable or overly risky alternatives Avoids wasted effort in elaborating unsatisfactory alternatives	Choice of risk resolution techniques: prototyping, simulation, modeling, benchmarking, reference checking, etc. Level of effort on each activity within each cycle	Sequential spiral phases with key stakeholders excluded from phases
3. Level of effort driven by risk considerations	Determines "how much is enough" of each activity: domain engineering, prototyping, testing, CM, etc. Avoids overkill or belated risk resolution	Choice of methods used to pursue activities: MBASE/WinWin, Rational RUP, JAD, QFD, ESP, … Degree of detail of artifacts produced in each cycle	Risk-insensitive evolutionary or incremental development. Impeccable spiral plan with no commitment to managing risks
4. Degree of detail driven by risk considerations	Determines "how much is enough" of each artifact (OCD, Requirements, Design, Code, Plans) in each cycle Avoids overkill or belated risk resolution	Choice of artifact representations (SA/SD, UML, MBASE, formal specs, programming languages, etc.)	Insistence on complete specifications for COTS, user interface, or deferred-decision situations
5. Use of anchor point milestones: LCO, LCA, IOC	Avoids analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, and useless systems	Number of spiral cycles or increments between anchor points Situation-specific merging of anchor point milestones	Evolutionary development with no life-cycle architecture
6. Emphasis on system and life cycle activities and artifacts	Avoids premature suboptimization on hardware, software, or development considerations	Relative amount in each cycle of • hardware vs. software • capability • productization (alpha, beta, shrink-wrap, etc.)	Purely logical object-oriented methods with operational, performance, or cost risks

Table 3: *Summary*

Acquisition System, September 2000, www.acq.osd.mil/ap/i50002p.doc

3. Hansen, W.; Foreman, J.; Carney, D; Forrester, E.; Graettinger, C.; Peterson, W.; and Place, P., Spiral Development Building the Culture: A Report on the CSE-SEI Workshop, February 2000, Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-2000-SR-006, July 2000, www/cbs/spiral2000/february2000/finalreport.html

4. Hansen, W.; Foreman, J.; Albert, C.; Axelband, E.; Brownsword, L.; Forrester, E. ; and Place, P., Spiral Development and Evolutionary Acquisition: The SEI-CSE Workshop, September, 2000, Software Engineering Institute, Carnegie Mellon University, Special Report, in preparation.

5. Boehm, Barry, edited by Hansen, Wilfred J., Spiral Development: Experience, Principles, and Refinements, Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-00-SR-08, ESC-SR-00-08, June, 2000, www/cbs/ spiral2000/february2000/BoehmSR.html.

6. Carr, M. J.; Konda, S. L.; Monarch, I.; Ulrich, F. C., and Walker, C. F., Taxonomy-Based Risk Identification, Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-93-TR-6, ESC-TR-93-183, June,1993,www.sei.cmu.edu/legacy/risk/kit/tr06.93.pdf

7. Williams, R. C.; Pandelios, G. J.; and Behrens, S.G., Software Risk Evaluation (SRE) Method Description (Version 2.0), Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-99-TR-029, ESC-TR-99-029, December, 1999, www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr029body.pdf

8. Boehm, B., Unifying Software Engineering and Systems Engineering, *IEEE Computer*, March 2000, pp. 114-116.

9. Boehm, B., *Software Risk Management*, IEEE Computer Society Press, 1989.

10. Mehta, N., *MBASE Electronic Process Guide*, USC-CSE, Los Angeles, Calif., October 1999, sunset.usc.edu/research/MBASE/EPG

11. Boehm, B., *Software Engineering Economics*, New York, N.Y., Prentice Hall, 1981.

12. Hantos, P., From Spiral to Anchored Processes: A Wild Ride in Lifecycle Building architecture, Proceedings, USC-SEI Spiral Experience Workshop, Los Angeles, Calif., Febuary 2000, www.sei.cmu.edu/cbs/spiral2000/Hantos

13. Rechtin, E., and Maier, M., *The Art of Systems Architecting*, CRC Press, 1997.

14. IEEE and EIA, Industry Implementation of ISO/IEC 12207: Software Life Cycle Processes-Implementation Considerations, IEEE/EIA 12207.2 - 1997, April 1998.

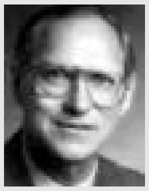
Article Web Site Location

http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00sr008.pdf

"It is impossible to make a program foolproof because fools are so ingenious."

Anonymous

About the Authors



Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Boehm orginiated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Voice: 213-740-8163
Fax: 213-740-4927
E-mail: boehm@sunset.usc.edu

Wilfred J.Hansen has been consulting on the tools and processes for utilizing commercial-off-the-shelf (COTS) software within larger systems at the Software Engineering Institute. Previously he was director of the Andrew Consortium where he led the development and maintenance of the Andrew User Interface System, which was the first modern word-processor in that it originated the embedding of arbitrary objects into text and into other objects. One of his contributions to Andrew was a scripting language having a unique integer-free sub-language for processing strings. Earlier in his career, Hansen taught data structures and programming languages.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-3890
Voice: 412-268-8247
E-mail: wjh@sei.cmu.edu

Web sites

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center (STSC) was established in 1987 as the command focus for proactive application of software technology in weapon, command and control, intelligence, and mission-critical systems. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. It helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. STSC uses the term technology in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. Its focus is on field-proven technologies that will benefit the Department of Defense mission.

Software Engineering Institute

www.sei.cmu.edu

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the Department of Defense to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. SEI helps organizations and individuals to improve their software engineering management practices. The site features a software engineering management practices work area that focuses on the ability of organizations to predict and control quality, schedule, cost, cycle time, and productivity when acquiring, building, or enhancing software systems.

Data & Analysis Center for Software

www.dacs.dtic.mil

The Data and Analysis Center for Software (DACS) is a Department of Defense (DoD) Information Analysis Center. The DACS is the DoD software information clearinghouse serving as an authoritative source for state of the art software information and providing technical support to the software community. The center's technical area of focus is software technology and software engineering, in its broadest sense. The DACS offers a wide variety of technical services designed to support the development, testing, validation, and transitioning of software engineering technology. The DACS is technically managed by the Air Force Research Laboratory - Information Directorate.

The Institute of Electrical and Electronics Engineers

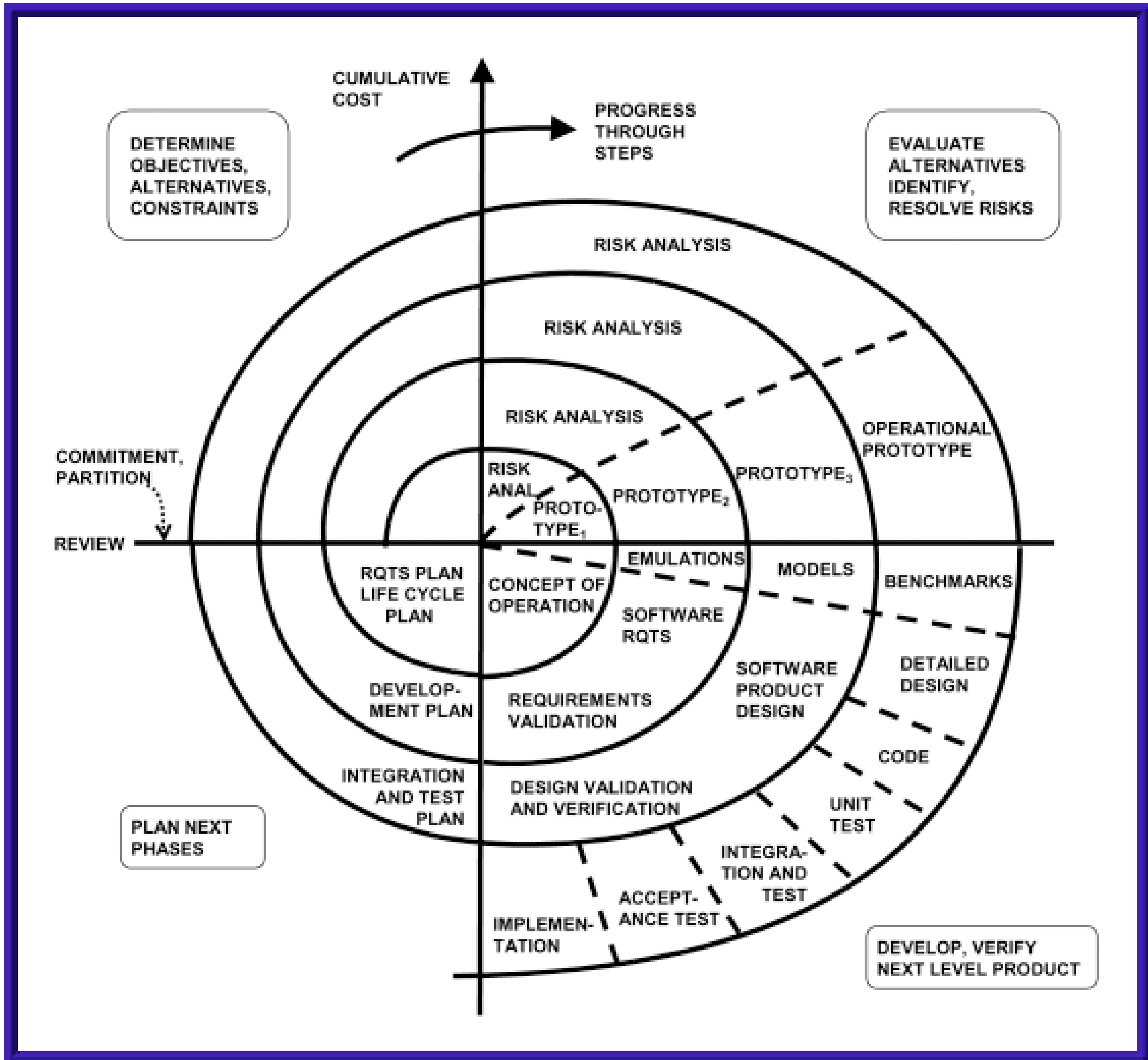
www.ieee.org

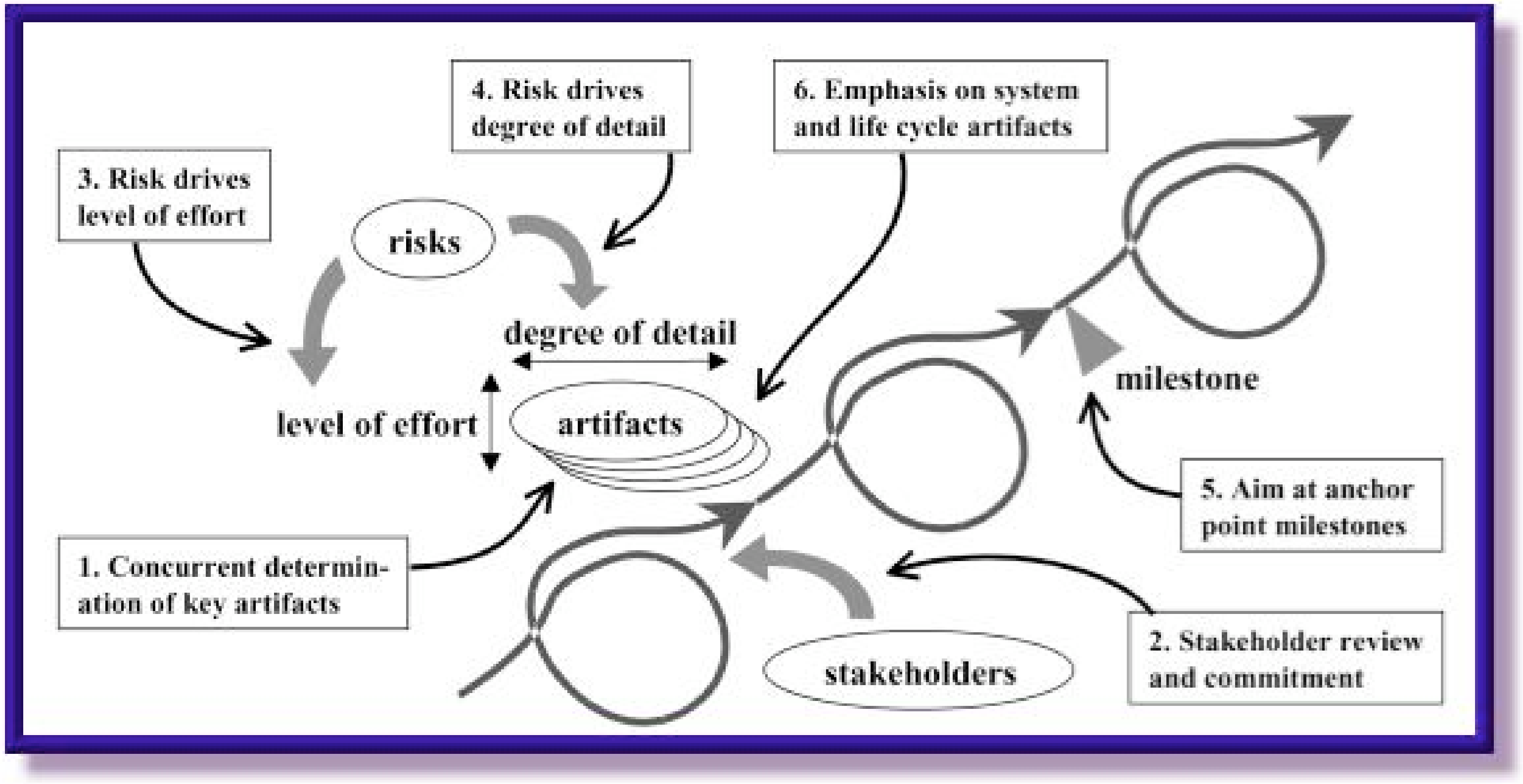
The Institute of Electrical and Electronics Engineers, Inc. (IEEE), helps advance global prosperity by promoting the engineering process of creating, developing, integrating, sharing, and applying knowledge about electrical and information technologies and sciences for the benefit of humanity and the profession. The IEEE is a nonprofit, technical professional association of more than 350,000 individual members in 150 countries. Through its members, the IEEE is a leading authority in technical areas ranging from computer engineering, biomedical technology and telecommunications, to electric power, aerospace and consumer electronics, among others.

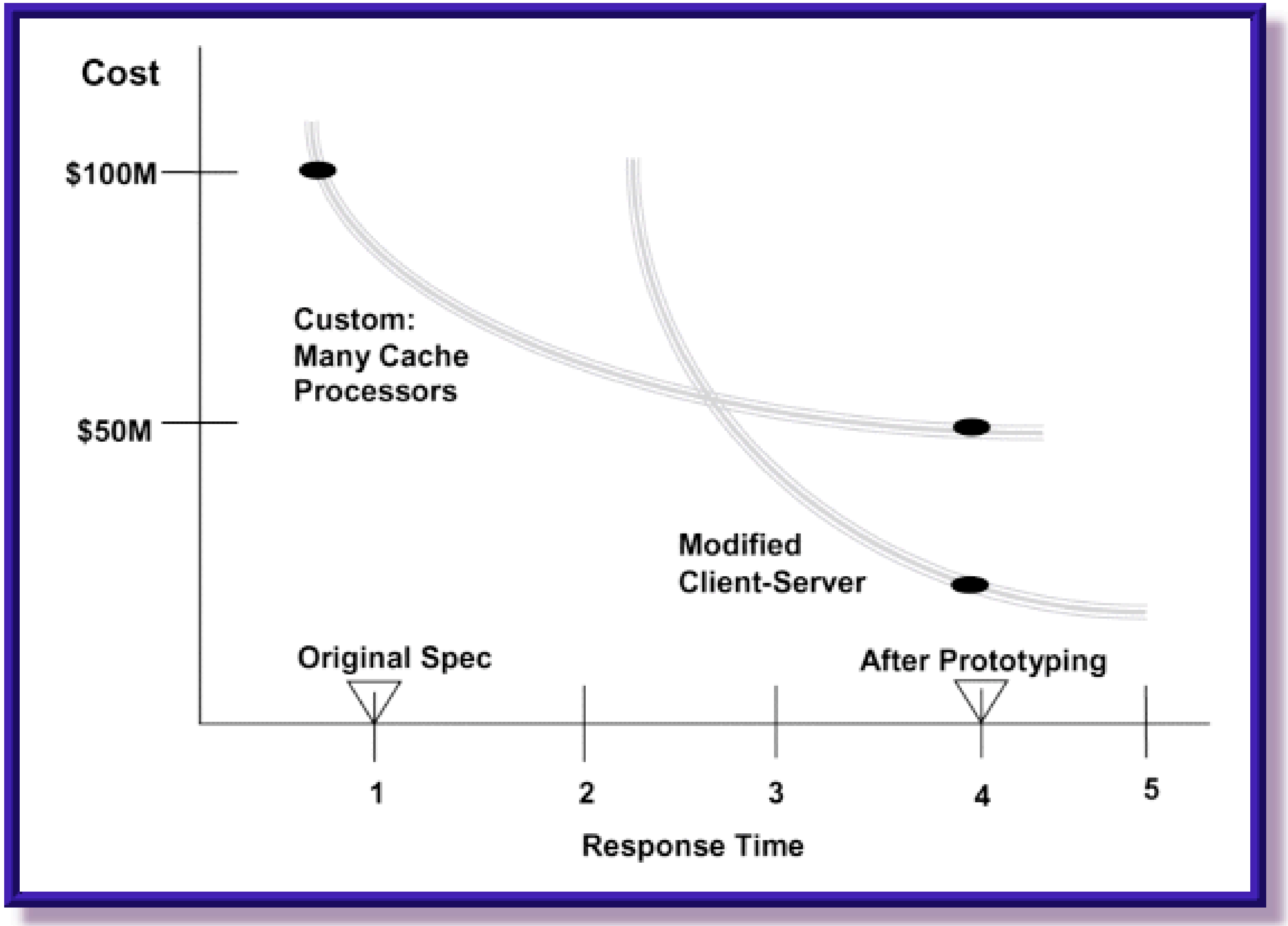
Software Productivity Consortium

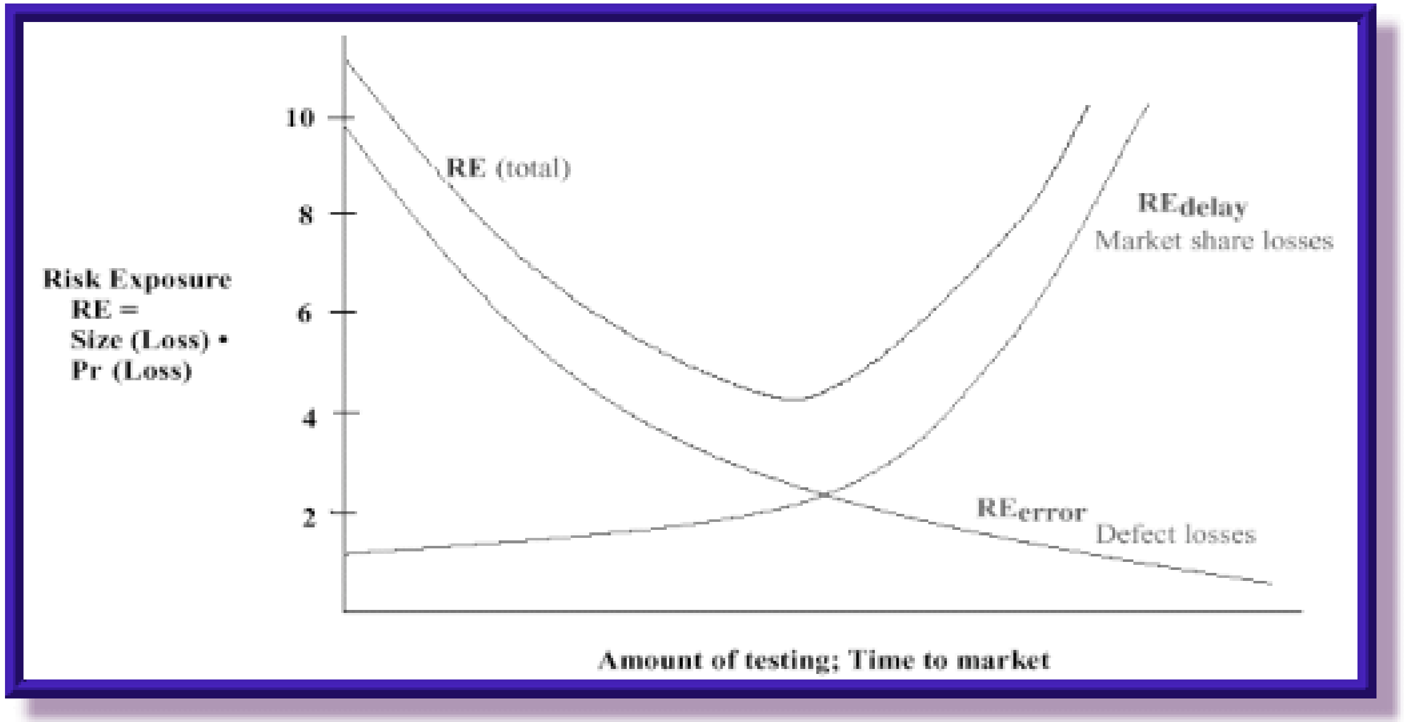
www.software.org

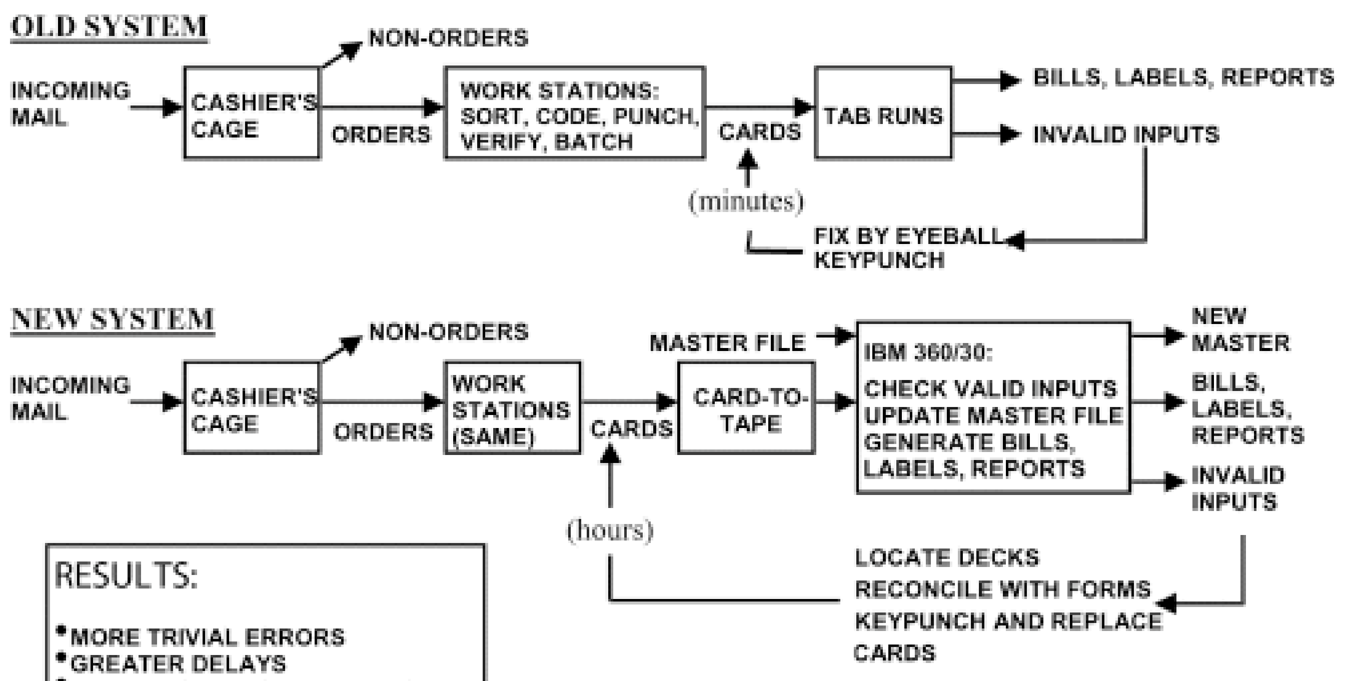
The Software Productivity Consortium is a unique, nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. The site features an interactive section to discuss new trends.


CUMULATIVE COST
PROGRESS THROUGH STEPS
DETERMINE OBJECTIVES, ALTERNATIVES, CONSTRAINTS
EVALUATE ALTERNATIVES IDENTIFY, RESOLVE RISKS
RISK ANALYSIS
RISK ANALYSIS
RISK ANALYSIS
RISK ANAL.
PROTO-TYPE$_1$
PROTOTYPE$_2$
PROTOTYPE$_3$
OPERATIONAL PROTOTYPE
COMMITMENT, PARTITION
REVIEW
RQTS PLAN LIFE CYCLE PLAN
CONCEPT OF OPERATION
EMULATIONS
MODELS
BENCHMARKS
SOFTWARE RQTS
SOFTWARE PRODUCT DESIGN
DETAILED DESIGN
DEVELOP-MENT PLAN
REQUIREMENTS VALIDATION
CODE
UNIT TEST
INTEGRATION AND TEST PLAN
DESIGN VALIDATION AND VERIFICATION
INTEGRA-TION AND TEST
ACCEPT-ANCE TEST
IMPLEMEN-TATION
PLAN NEXT PHASES
DEVELOP, VERIFY NEXT LEVEL PRODUCT


3. Risk drives level of effort
4. Risk drives degree of detail
6. Emphasis on system and life cycle artifacts
risks
degree of detail
level of effort
artifacts
milestone
1. Concurrent determin-ation of key artifacts
5. Aim at anchor point milestones
stakeholders
2. Stakeholder review and commitment


Cost
$100M
$50M
Custom:
Many Cache
Processors
Modified
Client-Server
Original Spec
After Prototyping
1
2
3
4
5
Response Time

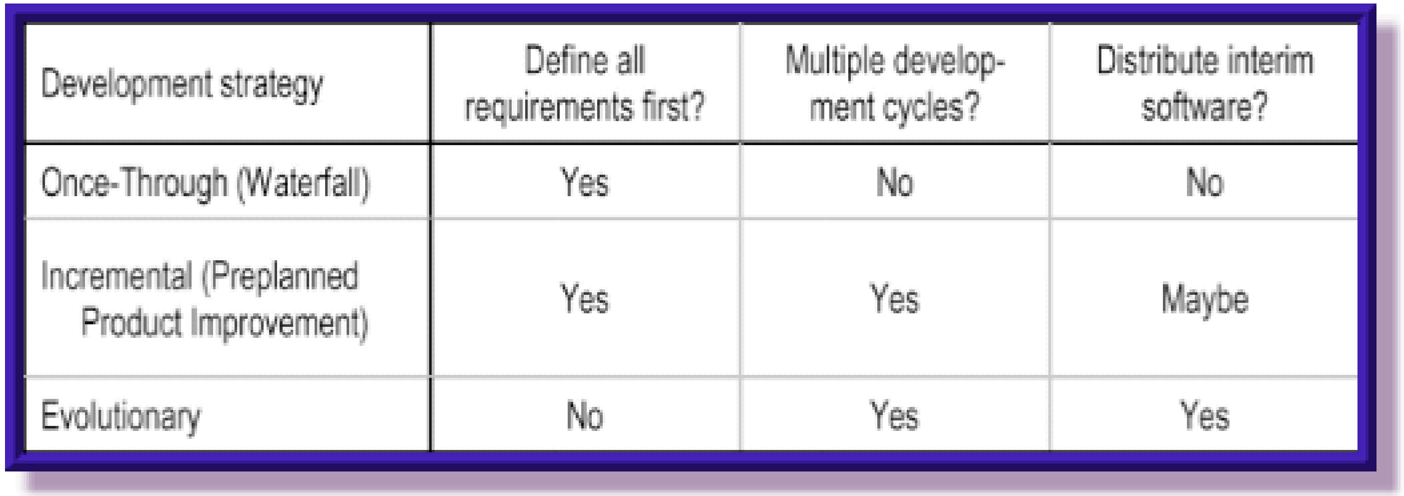

Risk Exposure
RE =
Size (Loss) •
Pr (Loss)
10
8
6
4
2
RE (total)
RE delay
Market share losses
RE error
Defect losses
Amount of testing; Time to market



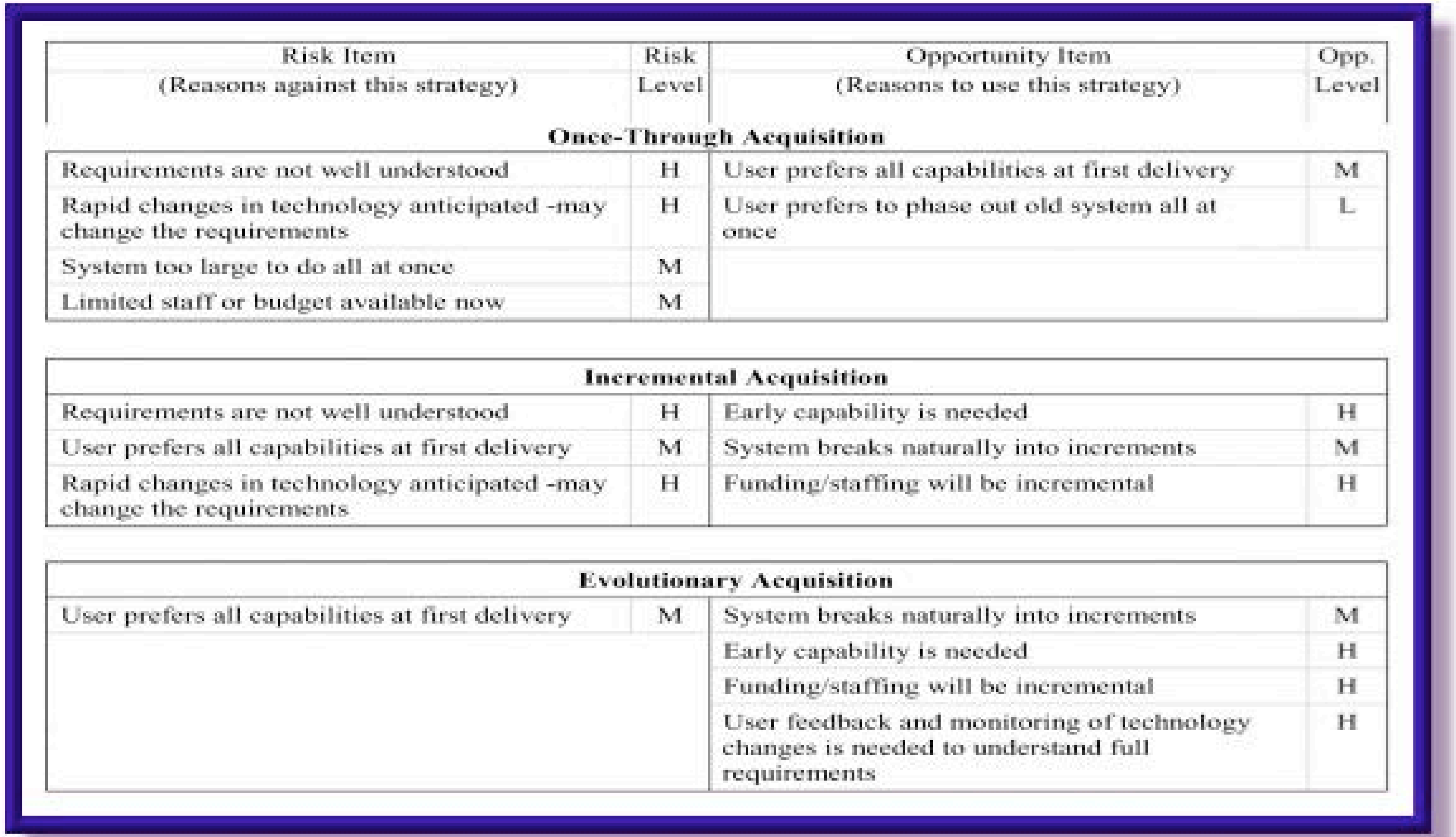
RESULTS:

- MORE TRIVIAL ERRORS
- GREATER DELAYS
- POOR EXCEPTION-HANDLING
- CUMBERSOME INPUT CONTROLS
- MORE LABOR-INTENSIVE

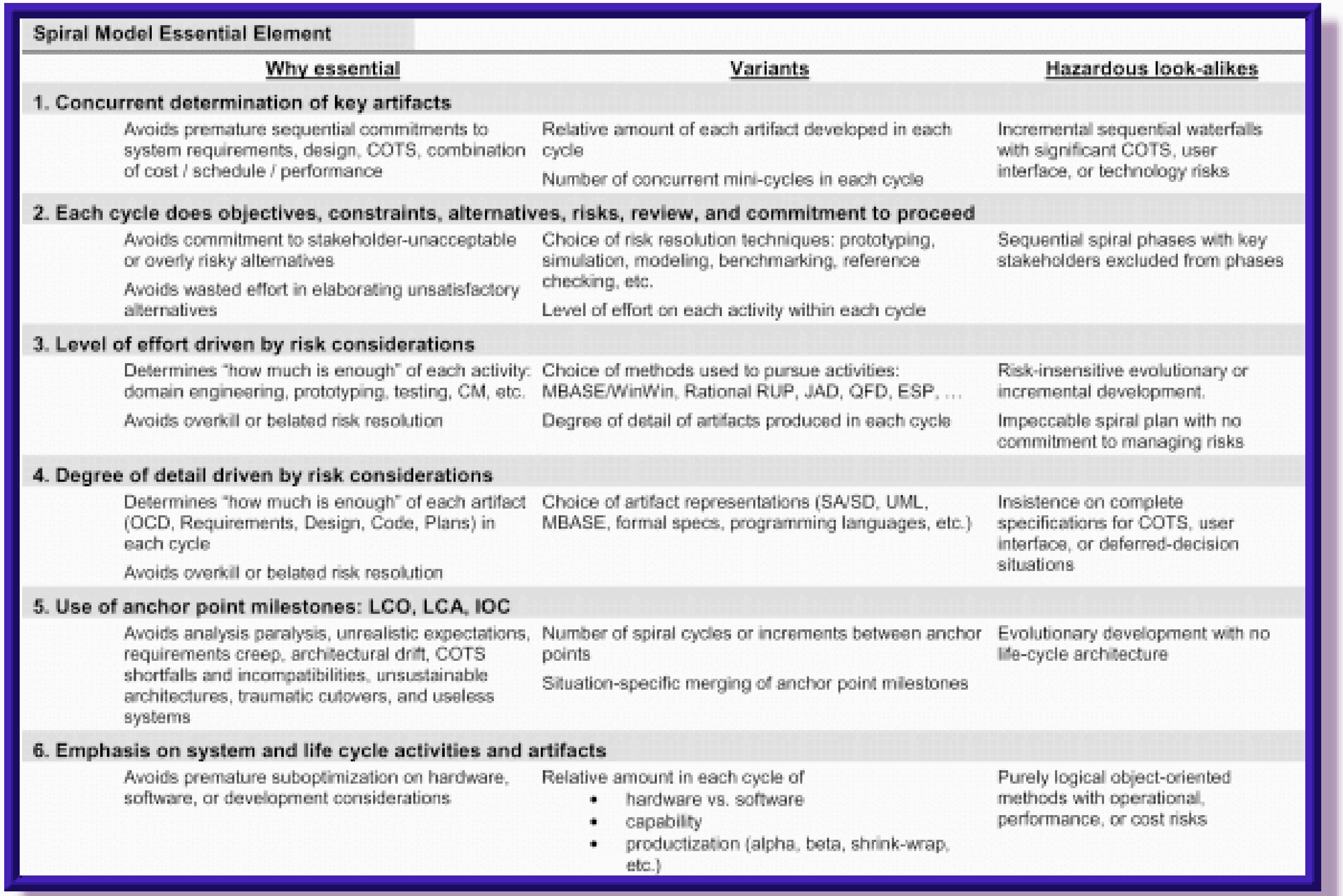
TRW



Development strategy	Define all requirements first?	Multiple develop- ment cycles?	Distribute interim software?
Once-Through (Waterfall)	Yes	No	No
Incremental (Preplanned Product Improvement)	Yes	Yes	Maybe
Evolutionary	No	Yes	Yes



Risk Item (Reasons against this strategy)	Risk Level	Opportunity Item (Reasons to use this strategy)	Opp. Level
Once-Through Acquisition			
Requirements are not well understood	H	User prefers all capabilities at first delivery	M
Rapid changes in technology anticipated -may change the requirements	H	User prefers to phase out old system all at once	L
System too large to do all at once	M		
Limited staff or budget available now	M		
Incremental Acquisition			
Requirements are not well understood	H	Early capability is needed	H
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
Rapid changes in technology anticipated -may change the requirements	H	Funding/staffing will be incremental	H
Evolutionary Acquisition			
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
		Early capability is needed	H
		Funding/staffing will be incremental	H
		User feedback and monitoring of technology changes is needed to understand full requirements	H



Spiral Model Essential Element	Why essential	Variants	Hazardous look-alikes
1. Concurrent determination of key artifacts			
	Avoids premature sequential commitments to system requirements, design, COTS, combination of cost / schedule / performance	Relative amount of each artifact developed in each cycle Number of concurrent mini-cycles in each cycle	Incremental sequential waterfalls with significant COTS, user interface, or technology risks
2. Each cycle does objectives, constraints, alternatives, risks, review, and commitment to proceed			
	Avoids commitment to stakeholder-unacceptable or overly risky alternatives Avoids wasted effort in elaborating unsatisfactory alternatives	Choice of risk resolution techniques: prototyping, simulation, modeling, benchmarking, reference checking, etc. Level of effort on each activity within each cycle	Sequential spiral phases with key stakeholders excluded from phases
3. Level of effort driven by risk considerations			
	Determines "how much is enough" of each activity: domain engineering, prototyping, testing, CM, etc. Avoids overkill or belated risk resolution	Choice of methods used to pursue activities: MBASE/WinWin, Rational RUP, JAD, QFD, ESP, … Degree of detail of artifacts produced in each cycle	Risk-insensitive evolutionary or incremental development. Impeccable spiral plan with no commitment to managing risks
4. Degree of detail driven by risk considerations			
	Determines "how much is enough" of each artifact (OCD, Requirements, Design, Code, Plans) in each cycle Avoids overkill or belated risk resolution	Choice of artifact representations (SA/SD, UML, MBASE, formal specs, programming languages, etc.)	Insistence on complete specifications for COTS, user interface, or deferred-decision situations
5. Use of anchor point milestones: LCO, LCA, IOC			
	Avoids analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, and useless systems	Number of spiral cycles or increments between anchor points Situation-specific merging of anchor point milestones	Evolutionary development with no life-cycle architecture
6. Emphasis on system and life cycle activities and artifacts			
	Avoids premature suboptimization on hardware, software, or development considerations	Relative amount in each cycle of • hardware vs. software • capability • productization (alpha, beta, shrink-wrap, etc.)	Purely logical object-oriented methods with operational, performance, or cost risks