



University of Southern California  
Center for Software Engineering

# Spiral Development: Experience, Principles, and Refinements

**Barry Boehm, USC**  
**Spiral Experience Workshop**  
**February 9, 2000**

**boehm@sunset.usc.edu**  
**<http://sunset.usc.edu/MBASE>**

2/9/00

©USC-CSE


1

This presentation opened the USC-SEI Workshop on Spiral Development\* Experience and Implementation Challenges held at USC February 9-11, 2000. The workshop brought together leading executives and practitioners with experience in transitioning to spiral development of software-intensive systems in the commercial, aerospace, and government sectors. Its objectives were to distill the participants' experiences into a set of critical success factors for transitioning to and successfully implementing spiral development, and to identify the most important needs, opportunities, and actions to expedite organizations' transition to successful spiral development.

To provide a starting point for addressing these objectives, I tried in this talk to distill my experiences in developing and transitioning the spiral model at TRW; in using it in system acquisitions at DARPA; in trying to refine it to address problems that people have had in applying it in numerous commercial, aerospace, and government contexts; and in working with the developers of major elaborations and refinements of the spiral model such that the Software Productivity Consortium's Evolutionary Spiral Process [SPC, 1994] and Rational, Inc's Rational Unified Process [Royce, 1998; Kruchten 1999; Jacobson et al., 1999]. I've modified the presentation somewhat to reflect the experience and discussions at the Workshop.

\*For the workshop, "development" was defined to include life cycle evolution of software-intensive systems and such related practices as legacy system replacement and integration of commercial-off-the-shelf (COTS) components.





University of Southern California  
Center for Software Engineering

---

## **“Spiral Development Model:” Candidate Definition**

**The spiral development model is a risk-driven process model generator. It 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. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.**

2/9/00 ©USC-CSE 3

A process model answers two main questions:

- What should the project do next?
- How long should the project continue doing it?

The spiral model holds that the answers to these questions vary from project to project, and that the variation is driven by risk considerations. It emphasizes the importance of having all of the project’s success-critical stakeholders participate concurrently in defining and executing the project’s processes (it uses risk considerations to ensure that progress is not overly slowed down by stakeholder overparticipation). It can be used to integrate software, hardware, and systems considerations, but is most important to use for software-intensive systems.

The cyclic nature of the spiral model was illustrated in Chart 2. The anchor-point stakeholder-commitment milestones are discussed in Charts 17-19.

## Spiral Invariants and Variants - 1

- Critical success factor examples

Invariants	Why Invariant	Variants
1. Concurrent rather than sequential determination of artifacts (OCD, Rqts, Design, Code, Plans) in each spiral cycle.	<ul style="list-style-type: none"> <li>Avoids premature sequential commitments to Rqts, Design, COTS, combination of cost/schedule performance                             <ul style="list-style-type: none"> <li>- 1 sec. response time</li> </ul> </li> </ul>	1a. Relative amount of each artifact developed in each cycle.  1b. Number of concurrent mini-cycles in each cycle.
2. Consideration in each cycle of critical-stakeholder objectives and constraints, product and process alternatives, risk identification and resolution, stakeholder review and commitment to proceed.	<ul style="list-style-type: none"> <li>Avoids commitment to stakeholder-unacceptable or overly risky alternatives.</li> <li>Avoids wasted effort in elaborating unsatisfactory alternatives.                             <ul style="list-style-type: none"> <li>- Mac-based COTS</li> </ul> </li> </ul>	2a. Choice of risk resolution techniques: prototyping, simulation, modeling, benchmarking, reference checking, etc.  2b. Level of effort on each activity within each cycle.
3. Level of effort on each activity within each cycle driven by risk considerations.	<ul style="list-style-type: none"> <li>Determines "how much is enough" of each activity: domain engr., prototyping, testing, CM, etc.                             <ul style="list-style-type: none"> <li>- Pre-ship testing</li> </ul> </li> <li>Avoids overkill or belated risk resolution.</li> </ul>	3a. Choice of methods used to pursue activities: MBASE/ WinWin, Rational USDP, JAD, QFD, ESP, ...  3b. Degree of detail of artifacts produced in each cycle.

2/9/00

©USC-CSE

4

This chart summarizes the first three of the six identified spiral invariants:

1. Concurrent rather than sequential determination of artifacts.
2. Consideration in each spiral cycle of the main spiral elements: critical-stakeholder objectives and constraints; product and process alternatives; risk identification and resolution; stakeholder review and commitment to proceed.
3. Using risk considerations to determine the level of effort to be devoted on each activity within each spiral cycle.

Chart 13 summarizes the second three spiral invariants:

4. Using risk considerations to determine the degree of detail of each artifact produced in each spiral cycle.
5. Managing stakeholder life-cycle commitments via three Anchor Point milestones: Life Cycle Objectives (LCO), Life Cycle Architecture(LCA), and Initial Operational Capability(IOC).
6. Emphasis on system and life cycle activities and artifacts rather than software and initial development activities and artifacts.

Both this chart and Chart 13 summarize the invariants; critical-success-factor reasons why they are essential invariants, and associated optional variants. The next few charts elaborate each individual invariant.



## **Spiral Invariant 1: Concurrent Determination of Key Artifacts (Ops Concept, Rqts, Design, Code, Plans)**

- **Why invariant**
  - **Avoids premature sequential commitments to system requirements, design, COTS, combination of cost/schedule/ performance**
  - **1 sec response time**
- **Variants**
  - 1a. Relative amount of each artifact developed in each cycle.**
  - 1b. Number of concurrent mini-cycles in each cycle.**
- **Models excluded**
  - **Incremental sequential waterfalls with high risk of violating waterfall model assumptions**

2/9/00

©USC-CSE

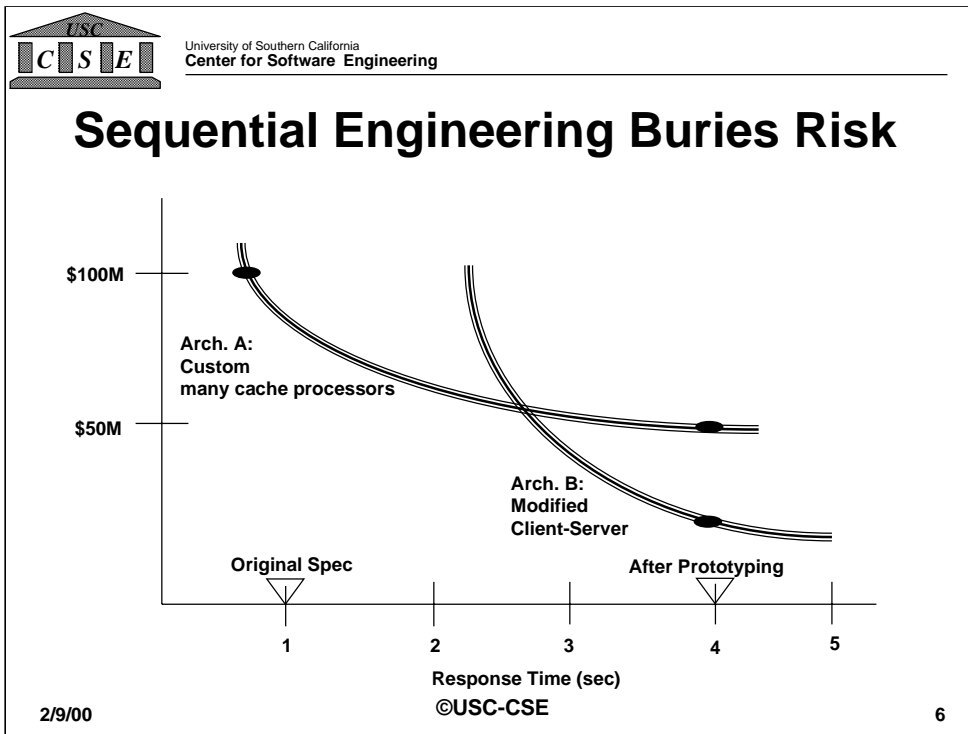
5

Spiral invariant 1 states that it is success-critical to concurrently determine a compatible and feasible combination of key artifacts: the operational concept, the system and software requirements, the system and software architecture and design, key elements of code (COTS, reused components, prototypes, success-critical components or algorithms), and plans.

Why is this a success-critical invariant? Because sequential determination of the key artifacts will prematurely overconstrain, and often extinguish, the project's ability to develop a system which satisfies the stakeholders' essential success conditions. Examples are premature commitments to hardware platforms, to incompatible combinations of COTS components [Garlan et al., 1995], and to requirements whose achievability has not been validated. Chart 6 provides an example of the kinds of problems that occur when high-risk requirements are prematurely frozen.

The variants 1a and 1b indicate that the product and process internals of the concurrent engineering activity are not invariant. 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, there is no invariant number of mini-cycles (e.g., individual prototypes for COTS, algorithm, or user-interface risks) within a given spiral cycle.

This invariant excludes one model often labeled as a spiral process, but which is actually a "hazardous spiral look-alike." This is the use of a sequence of incremental waterfall developments with a high risk of violating the underlying assumptions of the waterfall model described in Chart 7.



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 and an oversimplified high-level performance analysis, they 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 anticipate query patterns and cache copies of data so that each user's likely data would be within one second's reach. The resulting hardware architecture had more than 25 super-midcomputers 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, the customer and developer decided to develop a prototype of the system's user interface and representative capabilities to test. The results showed that a four-second response time would satisfy users 90 percent of the time. A four-second response time dropped development costs closer to \$30 million [Boehm, 2000]. Thus, the premature specification of a 1-second response time buried the risk of creating an overexpensive and time-consuming system development.

## Waterfall Model Assumptions

1. The requirements are knowable in advance of implementation.
2. The requirements have no unresolved, high-risk implications
  - e.g., risks due to COTS choices, cost, schedule, performance, safety, security, user interfaces, organizational impacts
3. The nature of the requirements will not change very much
  - During development; during evolution
4. The requirements are compatible with all the key system stakeholders' expectations
  - e.g., users, customer, developers, maintainers, investors
5. The right architecture for implementing the requirements is well understood.
6. There is enough calendar time to proceed sequentially.

2/9/00

©USC-CSE

7

This chart summarizes the assumptions about a software project's state of nature that need to be true for the waterfall model to succeed. If all of these are true, then it is a project risk not to specify the requirements, and the waterfall model becomes a risk-driven special case of the spiral model. If any of the assumptions are untrue, then specifying a complete set of requirements in advance of risk resolution will commit a project to assumptions/requirements mismatches that will lead the project into trouble.

Assumption 1 -- the requirements are knowable in advance of implementation -- is generally untrue for new user-interactive systems, because of the IKIWISI syndrome. When asked for their required screen layout for a new decision-support systems, 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 needed.

The effects of invalidity in assumptions 2, 4, and 5 are well illustrated by the example in Chart 5. The 1-second response time requirement was unresolved and high-risk. It was compatible with the users' expectations, but not with the customer's budget expectations. And the need for an expensive custom architecture was not understood in advance.

The effects of invalidity in assumptions 3 and 6 are well illustrated by electronic commerce projects. There, the volatility of technology and the marketplace is so high that requirements and traceability updates will swamp the project in overhead. And the amount of initial calendar time it takes to work out a complete set of detailed requirements that are likely to change several times downstream is not a good investment of the scarce time to market available to develop an initial operational capability.



## **Spiral Invariant 2: Each cycle does objectives, constraints, alternatives, risks, review, commitment to proceed**

- **Why invariant**
  - Avoids commitment to stakeholder-unacceptable or overly risky alternatives.
  - Avoids wasted effort in elaborating unsatisfactory alternatives.
    - Windows-only COTS
- **Variants**
  - 2a. **Choice of risk resolution techniques: prototyping, simulation, modeling, benchmarking, reference checking, etc.**
  - 2b. **Level of effort on each activity within each cycle.**
- **Models excluded**
  - **Sequential phases with key stakeholders excluded**

Spiral invariant 2 identifies the activities in each quadrant of the original spiral diagram which 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 stakeholder's review and commitment to proceed based on satisfaction of their critical objectives and constraints.

If all of these are not considered, the project may prematurely commit itself to alternatives that are either unacceptable to key stakeholders or overly risky. Or it can waste a good deal of effort in elaborating an alternative that could have been shown earlier to be unsatisfactory. Chart 10 provides a representative example.

Spiral invariant 2 does not mandate particular generic choices of risk resolution techniques. However, there are risk management guidelines, e.g., [Boehm, 1989], that suggest best-candidate risk resolution techniques for the major sources of project risk. Invariant 2 also does not mandate particular levels of effort for the activities performed during each cycle. This means, for example, that software cost estimation models cannot be precise about the amount of effort and cost required for each cycle.

This invariant excludes another "hazardous spiral look-alike": organizing the project into sequential phases or cycles in which key stakeholders are excluded. Examples are excluding developers from system definition, excluding users from system construction, or excluding system maintainers from either definition or construction (see Chart 11).



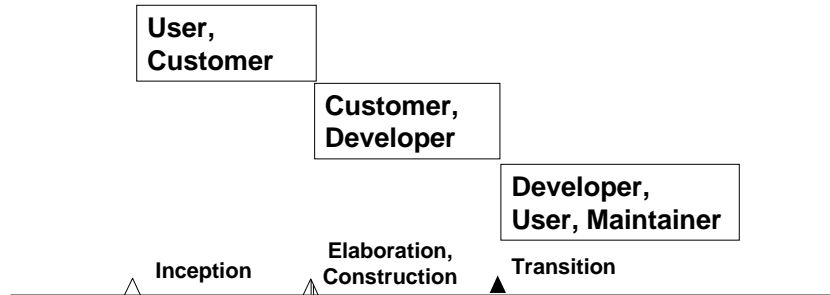
## Windows-Only COTS Example: Digital Library Artifact Viewer

- **Great prototype using ER Mapper**
  - Tremendous resolution
  - Incremental-resolution artifact display
  - Powerful zoom and navigation features
- **Only runs well on Windows**
  - Mac, Unix user communities forced to wait
  - Overoptimistic assumptions on length of wait
- **Eventual decision to drop ER Mapper**

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 indicators that Mac and Unix versions of ER Mapper would be available soon.

However, subsequent investigations indicated that it would be a long time before such Mac and Unix capabilities would become available. At a subsequent review, ER Mapper was dropped in favor of a less powerful but fully portable COTS product, Mr. SID, but only after a good deal of wasted effort was devoted to elaborating the ER Mapper solution. If a representative of the Mac or UNIX user community had been involved in the early project decisions, the homework leading to choosing Mr. SID would have been done earlier, and the wasted effort in elaborating the ER Mapper solution would have been avoided.

## Models Excluded: Sequential Phases Without Key Stakeholders



- **High risk of win-lose even with spiral phases**
  - Win-lose evolves into lose-lose
- **Key criteria for IPT members (AFI 63-123)**
  - Representative, empowered, knowledgeable, collaborative, committed

2/9/00

©USC-CSE

10

Even though the phases shown in this chart may look like risk-driven spiral cycles, this spiral look-alike will be hazardous because its exclusion of key stakeholders is likely to cause critical risks to go undetected. 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 evolve into lose-lose situations [Boehm-Ross, 1989].

Projects must also guard against having the appearance but not the reality of stakeholder participation by accepting an unqualified member of an integrated product team (IPT). A good set of criteria for qualified IPT members described in [Boehm et al., 1998] and adopted in [USAF, 2000] is to ensure that IPT members are representative (of organizational rather than personal positions); empowered (to make commitments which will be honored by their organizations); knowledgeable (of their organization's critical success factors); collaborative, and committed.

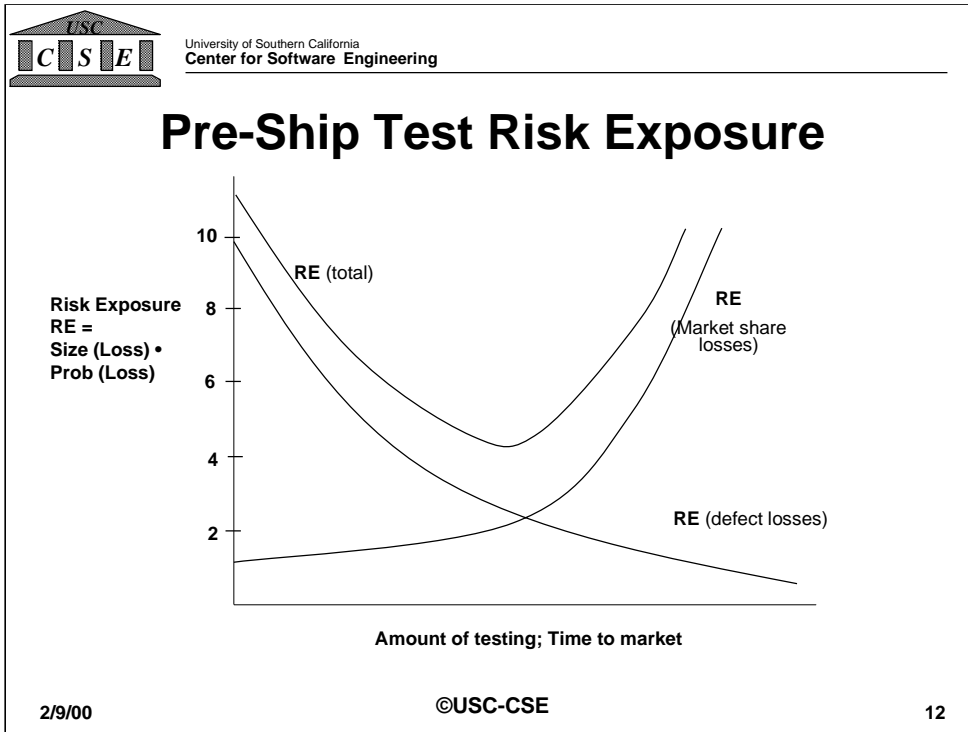
## Spiral Invariant 3: Level of Effort Driven by Risk Considerations

- **Why invariant**
  - Determines ‘how much is enough’ of each activity: domain engr., prototyping, testing, CM, etc.
    - Pre-ship testing
  - Avoids overkill or belated risk resolution.
- **Variants**
  - 3a. Choice of methods used to pursue activities: MBASE/WinWin, Rational RUP, JAD, QFD, ESP, . . .
  - 3b. Degree of detail of artifacts produced in each cycle.
- **Models excluded**
  - Risk-insensitive evolutionary or incremental development

Spiral invariant 3 uses risk considerations to provide answers to one of the most difficult questions for a project to answer: how much of a given activity (domain engineering, prototyping, testing, configuration management, etc.) is enough? An example of how this works for testing is provided in Chart 12. It shows that if you plot a project’s risk exposure as a function of time spent testing, there is a point at which risk exposure is minimized. Spending significantly more testing time than this is an overkill leading to late market entry and decreased market capitalization. Spending significantly less testing time than this is an underkill leading to early market entry with products that are so unreliable that the company loses market share and market capitalization.

Given that risk profiles vary from project to project, this means that the risk-minimizing level of testing effort will vary from project to project. The amount of effort devoted to other activities will also vary as a function of a project’s risk profile, again presenting a challenge for software cost models’ ability to estimate a project’s effort distribution by activity and phase. Another variant is an organization’s choice of particular methods for risk assessment and management.

Hazardous spiral model look-alikes excluded by invariant 3 are risk-insensitive evolutionary development (e.g., neglecting scalability risks) or risk-insensitive incremental development (e.g., suboptimizing on increment 1 with a point-solution architecture which must be dropped or heavily reworked to accommodate future increments); or impeccable spiral plans with no commitment to managing the risks identified.



This chart shows how risk considerations can help determine “how much testing is enough” before shipping a product. This can be determined by adding up the two main sources of risk Exposure,  $RE = \text{Probability (Loss)} \cdot \text{Size (Loss)}$ , incurred by two sources of loss: loss of profitability due to product defects, and loss of profitability due to delays in capturing market share. The more testing the project does, the lower becomes the risk exposure 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 the project spends testing, the higher are both the probability of loss due to competitors entering market and the size of loss due to decreased profitability on the remaining market share.

As shown in Chart 12, 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 “.com” company than it will for a safety-critical product such as a nuclear powerplant. 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.

## Spiral Invariants and Variants - 2

Invariants	Why Invariant	Variants
4. Degree of detail of artifacts produced in each cycle driven by risk considerations.	<ul style="list-style-type: none"> <li>Determines "how much is enough" of each artifact (OCD, Rqts, Design, Code, Plans) in each cycle.</li> <li>Avoids overkill or belated risk resolution</li> </ul>	4a. Choice of artifact representations (SA/SD, UML, MBASE, formal specs, programming languages, etc.)
5. Managing stakeholder life-cycle commitments via the LCO, LCA, and IOC Anchor Point milestones (getting engaged, getting married, having your first child),	<ul style="list-style-type: none"> <li>Avoids analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, useless systems.</li> </ul>	5a. Number of spiral cycles or increments between anchor points. 5b. Situation-specific merging of anchor point milestones.
6. Emphasis on system and life cycle activities and artifacts rather than software and initial development activities and artifacts.	<ul style="list-style-type: none"> <li>Avoids premature suboptimization on hardware, software, or initial development considerations.</li> </ul>	6a. Relative amount of hardware and software determined in each cycle. 6b. Relative amount of capability in each life cycle increment. 6c. Degree of productization (alpha, beta, shrink-wrap, etc.) of each life cycle increment.

2/9/00

©USC-CSE

13

This chart summarizes the second three spiral invariants:

4. Using risk considerations to determine the degree of detail of each artifact produced in each spiral cycle.
5. Managing stakeholder life-cycle commitments via three Anchor Point milestones: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC).
6. Emphasis on system and life cycle activities and artifacts rather than software and initial development activities and artifacts.

Both this chart and Chart 13 summarize the invariants, critical-success-factor reasons why they are essential invariants, and associated optional variants. The next few charts elaborate the second three invariants.



## Spiral Invariant 4:

### Degree of Detail Driven by Risk Considerations

- **Why invariant**
  - Determines “how much is enough” of each artifact (OCD, Rqts, Design, Code, Plans) in each cycle.
    - Screen layout rqts.
  - Avoids overkill or belated risk resolution.
- **Variants**
  - 4a. Choice of artifact representations (SA/SD, UML, MBASE, formal specs, programming languages, etc.)
- **Models excluded**
  - Complete, consistent, traceable, testable requirements specification for systems involving significant levels of GUI, COTS, or deferred decisions

Spiral invariant 4 is the product counterpart of invariant 3: that risk considerations determine the degree of detail of products as well as processes. This means, for example, that the traditional ideal of a complete, consistent, traceable, testable requirements specification is not a good idea for a number of product components, such as a graphic user interface (GUI). 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 (see Chart 15). 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, as indicated in Chart 15, some risk patterns make it very important to have precise specifications.

Related spiral variants are the project’s choices of representations for product artifacts.

## Risk-Driven Specifications

- If it's risky not to specify precisely, Do
  - Hardware-software interface
  - Prime-subcontractor interface
- If it's risky to specify precisely, Don't
  - GUI layout
  - COTS behavior

This chart gives further examples of when it is risky to overspecify software features (GUI layouts, COTS behavior) and when it is risky to underspecify them (critical interfaces with hardware or with externally developed software).



## Spiral Invariant 5: Use of LCO, LCA, IOC, Anchor Point Milestones

- **Why invariant**
  - **Avoids analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, useless systems.**
- **Variants**
  - 5a. Number of spiral cycles or increments between anchor points.**
  - 5b. Situation-specific merging of anchor point milestones**
    - **Can merge LCO and LCA when adopting an architecture from mature 4GL, product line**
- **Models excluded**
  - **Evolutionary or incremental development with no life cycle architecture**

2/9/00

©USC-CSE

16

A major difficulty of the original spiral model was its lack of intermediate milestones to serve as commitment points and progress checkpoints [Forsberg et al., 1996]. This difficulty has been remedied by the development of a set of anchor point milestones: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC) [Boehm, 1996].

Chart 17 describes the role of the LCO, LCA, and IOC milestones as stakeholder commitment points in the software life cycle. Chart 18 provides details on the content and pass/fail criteria for the LCO and LCA milestones. Chart 19 summarizes the content of the IOC milestone.

Appropriate variants include the number of spiral cycles of development increments between the anchor points. In some cases, anchor point milestones can be merged. In particular, a project deciding to use a mature and appropriately scalable fourth generation language (4GL) or product line framework will have already determined its choice of life cycle architecture by its LCO milestone, enabling the LCO and LCA milestones to be merged.

The LCA milestone is particularly important, as its pass/fail criteria enable stakeholders to hold up projects attempting to proceed into evolutionary or incremental development without a life cycle architecture. Chart 20 summarizes other evolutionary development assumptions whose validity should be verified at the LCA milestone.

Charts 21-25 summarize other aspects of the spiral model relevant to the anchor point milestones, such as their support of incremental commitment and their relation to the Rational Unified Process [Royce, 1998; Kruchten, 1998; Jacobson et al., 1999] and the USC MBASE approach [Boehm-Port, 1999a; Boehm-Port, 1999b; Boehm et al., 2000].



## Life Cycle Anchor Points

- **Common System/Software stakeholder commitment points**
  - Defined in concert with Government, industry affiliates
  - Coordinated with the Rational Unified Process
- **Life Cycle Objectives (LCO)**
  - Stakeholders' commitment to support architecting
  - Like getting engaged
- **Life Cycle Architecture (LCA)**
  - Stakeholders' commitment to support full life cycle
  - Like getting married
- **Initial Operational Capability (IOC)**
  - Stakeholders' commitment to support operations
  - Like having first child

The anchor point milestones were defined in a pair of USC Center for Software Engineering Affiliates' workshops, originally for the purpose of defining a set of common reference points for COCOMO II cost model estimates of spiral model projects' cost and schedule. One of the Affiliates, Rational, Inc., had been defining the phases of its Rational Unified Process, and adopted the anchor point milestones as its phase gates.

The first two anchor points are the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA). At each of these anchor points the key stakeholders review six artifacts: *operational concept description*, *prototyping results*, *requirements description*, *architecture description*, *life cycle plan*, and *feasibility rationale* (see next chart for details).

The feasibility rationale covers the key pass/fail question: "If I build this product using the specified architecture and processes, will it support the operational concept, realize the prototyping results, satisfy the requirements, and finish within the budgets and schedules in the plan?" If not, the package should be reworked.

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 review artifacts. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan.

The LCO milestone is the equivalent of getting engaged, and the LCA milestone is the equivalent of getting married. As in life, if you marry your architecture in haste, you and your stakeholders will repent at leisure. The third anchor point milestone, the Initial Operational Capability (IOC), constitutes an even larger commitment: It is the equivalent of having your first child.



## Win Win Spiral Anchor Points

(Risk-driven level of detail for each element)

Milestone Element	Life Cycle Objectives (LCO)	Life Cycle Architecture (LCA)
<b>Definition of Operational Concept</b>	<ul style="list-style-type: none"> <li>• Top-level system objectives and scope</li> <li>• System boundary</li> <li>• Environment parameters and assumptions</li> <li>• Evolution parameters</li> <li>• Operational concept</li> <li>• Operations and maintenance scenarios and parameters</li> <li>• Organizational life-cycle responsibilities (stakeholders)</li> </ul>	<ul style="list-style-type: none"> <li>• Elaboration of system objectives and scope of increment</li> <li>• Elaboration of operational concept by increment</li> </ul>
<b>System Prototype(s)</b>	<ul style="list-style-type: none"> <li>• Exercise key usage scenarios</li> <li>• Resolve critical risks</li> </ul>	<ul style="list-style-type: none"> <li>• Exercise range of usage scenarios</li> <li>• Resolve major outstanding risks</li> </ul>
<b>Definition of System Requirements</b>	<ul style="list-style-type: none"> <li>• Top-level functions, interfaces, quality attribute levels, including:               <ul style="list-style-type: none"> <li>- Growth vectors and priorities</li> <li>- Prototypes</li> <li>- Stakeholders' concurrence on essentials</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Elaboration of functions, interfaces, quality attributes, and prototypes by increment</li> <li>• Identification of TBD's (to-be-determined items)</li> <li>• Stakeholders' concurrence on their priority concerns</li> </ul>
<b>Definition of System and Software Architecture</b>	<ul style="list-style-type: none"> <li>• Top-level definition of at least one feasible architecture</li> <li>• Physical and logical elements and relationships</li> <li>• Choices of COTS and reusable software elements</li> <li>• Identification of infeasible architecture options</li> </ul>	<ul style="list-style-type: none"> <li>• Choice of architecture and elaboration by increment               <ul style="list-style-type: none"> <li>- Physical and logical components, connectors, configurations, constraints</li> <li>- COTS, reuse choices</li> <li>- Domain-architecture and architectural style choices</li> </ul> </li> <li>• Architecture evolution parameters</li> </ul>
<b>Definition of Life-Cycle Plan</b>	<ul style="list-style-type: none"> <li>• Identification of life-cycle stakeholders               <ul style="list-style-type: none"> <li>- Users, customers, developers, maintainers, interoperators, general public, others</li> </ul> </li> <li>• Identification of life-cycle process model               <ul style="list-style-type: none"> <li>- Top-level stages, increments</li> <li>- Top-level WWWWWHH* by stage</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Elaboration of WWWWWHH* for Initial Operational Capability (IOC)               <ul style="list-style-type: none"> <li>- Partial elaboration, identification of key TBD's for later increments</li> </ul> </li> </ul>
<b>Feasibility Rationale</b>	<ul style="list-style-type: none"> <li>• Assurance of consistency among elements above               <ul style="list-style-type: none"> <li>- via analysis, measurement, prototyping, simulation, etc.</li> <li>- Business case analysis for requirements, feasible architectures</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Assurance of consistency among elements above               <ul style="list-style-type: none"> <li>- All major risks resolved or covered by risk management plan</li> </ul> </li> </ul>

\*WWWWHH: Why, What, When, Who, Where, How, How Much

2/9/00

©USC-CSE

18

Here are the major features of the LCO and LCA milestones which distinguish them from most current software milestones, which provide a rationale for their success-criticality on projects, and which enable them to function successfully as anchor points across many types of software development.

- Their focus is not on requirements snapshots or architecture point solutions, but on requirements and architectural specifications which anticipate and accommodate system evolution. This is the reason for calling them the “Life Cycle” Objectives and Architecture milestones.
- Elements can be either specifications or executing programs with data (e.g., prototypes, COTS products).
- The Feasibility Rationale is an essential element rather than an optional add-on.
- Stakeholder concurrence on the milestone elements is essential. This establishes mutual stakeholder buy-in to the plans and specifications, and enables a collaborative team approach to unanticipated setbacks rather than an adversarial approach as in most contract models.

A feature distinguishing the LCA milestone from the LCO milestone is the need to have all of the system’s major risks resolved, or at least covered by an element of the system’s risk management plan. For large systems, passing the LCA milestone is the point at which the project will significantly escalate at its staff level and resource commitments. Proceeding into this stage with major risks unaddressed has led to disasters for many large projects. Some good guidelines for software risk assessment can be found in [Boehm, 1989; Charette, 1989; Carr et al., 1993; and Hall, 1998].

A key feature of the LCO milestone is the need for the Feasibility Rationale to demonstrate a viable business case for the proposed system. Not only should this business case be kept up to date, but also it should be used as a basis for verifying that expected benefits will actually be realized (see chart 27).



## Initial Operational Capability (IOC)

- **Software preparation**
  - Operational and support software
  - Data preparation, COTS licenses
  - Operational readiness testing
- **Site preparation**
  - Facilities, equipment, supplies, vendor support
- **User, operator, and maintainer preparation**
  - Selection, teambuilding, training

Another distinguishing feature of the LCO and LCA milestones is that they are the milestones with the most serious consequences if one gets any parts of them wrong. At the other end of the development cycle, the milestone with the most serious consequences of getting things wrong is the Initial Operational Capability (IOC). Greeting users with a new system having ill-matched software, poor site preparation, or poor users preparation has been a frequent source of user alienation and killed projects.

The key elements of the IOC milestone are:

- Software preparation, including both operational and support software with appropriate commentary and documentation; data preparation or conversion; the necessary licenses and rights for COTS and reused software, and appropriate operational readiness testing.
- Site preparation, including facilities, equipment, supplies, and COTS vendor support arrangements.
- User, operator and maintainer preparation, including selection, teambuilding, training and other qualification for familiarization, usage, operations, or maintenance.

As discussed on Chart 12, the nature of the IOC milestone is also risk-driven with respect to the system objectives determined in the LCO and LCA milestones. Thus, for example, these objectives drive the tradeoff between IOC date and quality of the product (e.g. between the safety-critical Space Shuttle Software and a market window-critical commercial software product). The difference between these two cases is narrowing as commercial vendors and users increasingly appreciate the market risks involved in buggy products [Cusumano-Selby, 1995].

## Evolutionary Development Assumptions

- 1. The initial release is sufficiently satisfactory to key system stakeholders that they will continue to participate in its evolution.**
- 2. The architecture of the initial release is scalable to accommodate the full set of system life cycle requirements (e.g., performance, safety, security, distribution, localization).**
- 3. The operational user organizations are sufficiently flexible to adapt to the pace of system evolution**
- 4. The dimensions of system evolution are compatible with the dimensions of evolving-out the legacy systems it is replacing.**

All too often, an evolutionary development will start off with a statement such as, “We’re not sure what to build, so let’s throw together a prototype and evolve it until the users are satisfied.” This approach is insensitive to several risks corresponding to the set of assumptions for a successful evolutionary development summarized in Chart 20.

Without some initial attention to user needs, the prototype may be so far from the users’ needs that they consider it a waste of time to continue. As discussed on Chart 16, it will be risky to proceed without a life cycle architecture to support evolution. Another risk is “information sclerosis”: the propensity for organizations to lock into operational procedures making it difficult to evolve toward better capabilities [Boehm, 1988]. A final frequent risk is that legacy systems are often too inflexible to adapt to desired directions of evolution. In such cases, a preferable process model is incremental development, with the increments determined by the ease of evolving-out portions of the legacy system to be replaced.



## **Spiral Model and Incremental Commitment: Stud Poker Analogy**

- **Evaluate alternative courses of action**
  - **Fold:** save resources for other deals
  - **Ante:** buy at least one more round
- **Using incomplete information**
  - **Hole cards:** competitive situation
  - **Rest of deck:** chance of getting winner
- **Anticipating future possibilities**
  - **Likelihood that next round will clarify outcome**
- **Commit incrementally rather than all at once**
  - **Challenge: DoD POM process makes this hard to do**

2/9/00

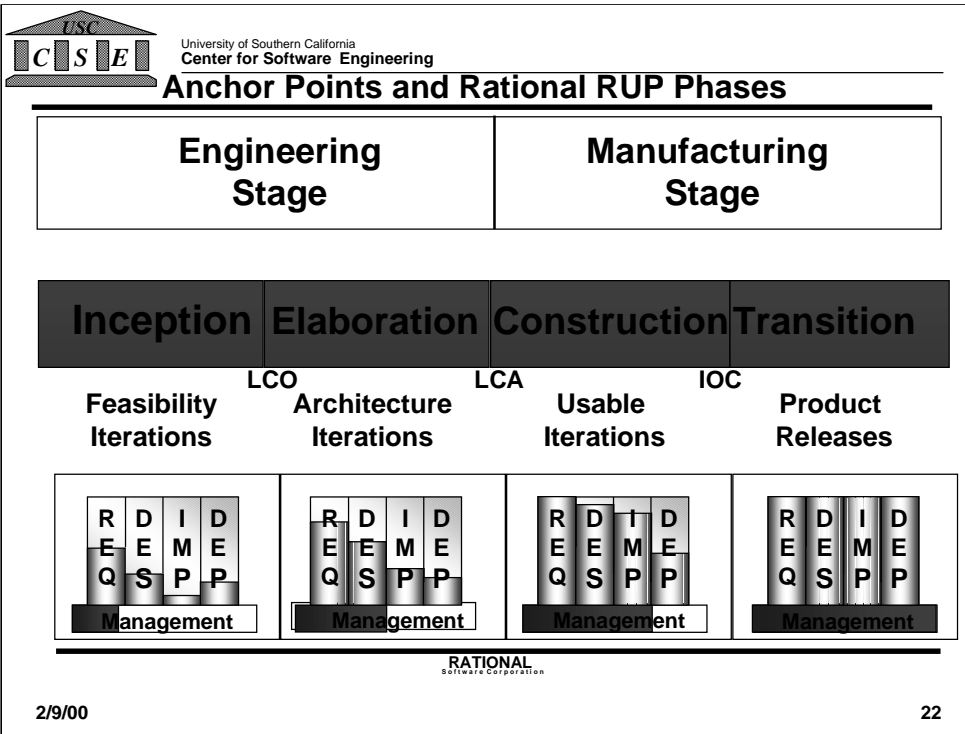
©USC-CSE

21

A valuable aspect of the original application of the spiral model to the TRW Software Productivity System was its ability to support incremental commitment of corporate resources to the exploration, definition, and development of the system, rather than requiring a large outlay of resources to the project before its success prospects were well understood [Boehm, 1988].

Funding a spiral development can thus be envisioned in a way similar to the game of stud poker. You can put a couple of chips in the pot and receive two hidden cards and one exposed card, along with the other players in the game. If your cards don't promise a winning outcome, you can drop out without a great loss. If your two hidden cards are both aces, you will probably bet on your prospects aggressively (although perhaps less so if you can see the other two aces as other players' exposed cards). In any case, you can decide during each round whether it's worth putting more chips in the pot to buy more information about your prospects for a win or whether it's better not to pursue this particular deal, based on the information available.

One of the main challenges for organizations such as the Department of Defense (DoD), is to find incremental commitment alternatives to its current Program Objectives Memorandum (POM) process which involves committing to the full funding of a program based on very incomplete early information.



Versions of this chart are in the three main books on the Rational Unified Process (RUP) [Royce, 1998; Kruchten, 1998; and Jacobson, et al., 1999]. It shows the relations between LCO, LCA, and IOC milestones and the RUP Inception, Elaboration, Construction, and Transition phases. It also illustrates that the requirements, design, implementation, and deployment artifacts are incrementally grown throughout the phases. As indicated in Variant 3b on Chart 11, the size of the shaded bars will vary from project to project.

# Spiral Model Refinements

•Where do objectives, constraints, alternatives come from?

–Win Win extensions

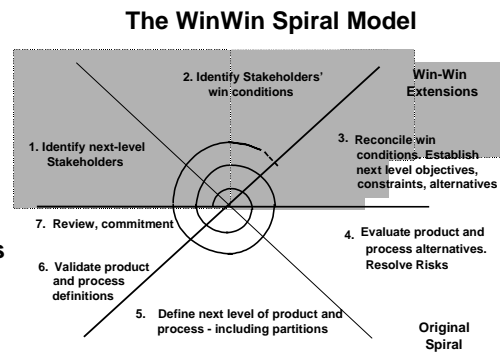
•Lack of intermediate milestones

–Anchor Points: LCO, LCA, IOC

–Concurrent-engineering spirals between anchor points

•Need to avoid model clashes, provide more specific guidance

–MBASE



2/9/00

©USC-CSE

23

The original spiral model [Boehm, 1988] began each cycle of the spiral by performing the next level of elaboration of the prospective system's objectives, constraints and alternatives. A primary difficulty in applying the spiral model has been the lack of explicit process guidance in determining these objectives, constraints, and alternatives. The Win-Win Spiral Model [Boehm-Bose, 1994] uses the Theory W (win-win) approach [Boehm-Ross, 1989] to converge on a system's next-level objectives, constraints, and alternatives. This Theory W approach involves identifying the system's stakeholders and their win conditions, and using negotiation processes to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders.

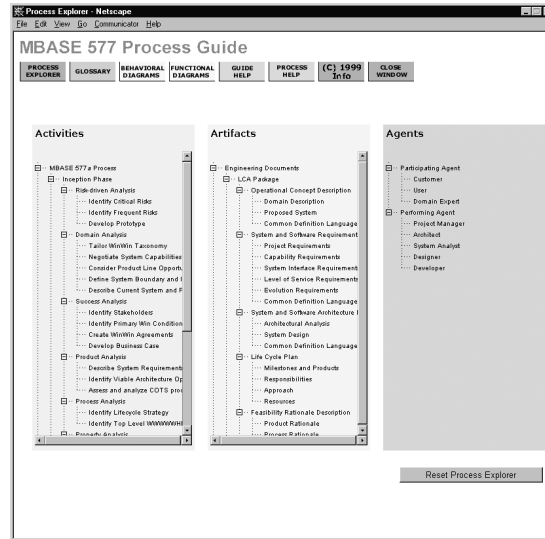
In particular, the nine-step Theory W process translates into the following Spiral Model extensions:

- Determine Objectives. Identify the system life-cycle stakeholders and their win conditions. Establish initial system boundaries and external interfaces.
- Determine Constraints. Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.
- Identify and Evaluate Alternatives. Solicit suggestions from stakeholders. Evaluate them with respect to stakeholders' win conditions. Synthesize and negotiate candidate win-win alternatives. Analyze, assess, and resolve win-lose or lose-lose risks.
- Record Commitments, and areas to be left flexible, in the project's design record and life cycle plans.

Cycle Through the Spiral. Elaborate win conditions, evaluate and screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans.

A further extension, the Model-Based (System) Architecting and Software Engineering (MBASE) approach, [Boehm-Port, 1999a; Boehm-Port, 1999b], provides more detailed definitions of the anchor point milestone elements [Boehm et al., 2000], and a process guide for deriving them (see next charts).

## MBASE Electronic Process Guide (1)



2/9/00

©USC-CSE

24

The MBASE Electronic Process Guide [Mehta, 1999] was developed using the SEI's Electronic Process Guide support tool [Kellner et al., 1999]. It uses Microsoft Access to store the process elements, using an Activities-Artifacts-Agents model, and translates the results into hyperlinked html for web-based usage. Thus, for example, when a user clicks on the "MBASE 577a Process" activity and the "Operational Concept Definition" artifact, the tool displays the corresponding elements as shown in Chart 25.

These elements are also hyperlinked. Thus, for example, a user can access a template for composing a project's Operational Concept Description by clicking on the "Templates" entry in its left column.



# MBASE Electronic Process Guide (2)

Activity: MBASE 577a Process - Netcape

MBASE 577a Process Guide

PROCESSES | GLOSSARY | BEHAVIORAL EXAMINERS | FUNCTIONAL EXAMINERS | GUIDE HELP | PROCESS HELP | [C] 1999 | LOGIN | WINDOW

Activities:

- MBASE 577a Process
  - Inception Phase
  - Elaboration Phase
  - Recent Projects/Events
  - Netcape/MBASE

**MBASE 577a Process**

**Overview**

The process followed by students of CS 577a for Digital Library projects

**Purpose**

The objectives for the MBASE 577a process are:

- To define a life cycle process for the students of CS 577a for use in Digital Library and similar projects
- To provide guidance to process enactors about the inter-dependence of MBASE process elements

**Decomposition**

The activity MBASE 577a Process is decomposed into the following:

- Inception Phase
- Elaboration Phase
- Recent Project Effort

**Description**

Model-Based (System) Architecting and Software Engineering (MBASE) is an approach for developing software intensive systems. The MBASE approach integrates the four common development models (process, product, process and property) around the creation and use of a software architecture package using MBASE, model deploy can be recognized and recorded as a matter of self-maner mark step the file.

The MBASE 577 process is a variant of the MBASE framework used by students in the course CS 577 offered at USC. The process consists of four distinct phases: Inception, Elaboration, Construction and Transition. Each phase is completed with a commitment from the stakeholders during a review.

The tasks to be performed during the MBASE 577 process are:

- Identifying and resolving the critical risks
- Analyzing the problem domain
- Identifying the feasible architecture
- Developing a prototype to test the validity of the architecture and satisfaction of stakeholder concerns
- Understanding the requirements of the life cycle and obtaining concurrence from the system stakeholders

**Tools and Techniques**

Electronic Process Guide for MBASE

Guidelines for Model-Based Architecting and Software Engineering (MBASE)

Derivables: Inception and Elaboration

**Pitfalls**

Artifact: Operational Concept Description - Netcape

MBASE 577 Process Guide

PROCESSES | GLOSSARY | BEHAVIORAL EXAMINERS | FUNCTIONAL EXAMINERS | GUIDE HELP | PROCESS HELP | [C] 1999 | LOGIN | WINDOW

Artifacts:

- ring Document
- Proposal
- Operational Concept Description
  - Domain Description
  - Proposed System
  - Common Definition Language for System and Software Architecture Data
  - System and Software Requirements Data
  - System and Software Architecture Data
  - Life Cycle Plan
  - Feasibility Statement Description
  - Postpone
  - Workshop Negotiation Report
  - System and Software Architecture Data
  - System and Software Requirements Data
  - and Other Reports
  - Use Patterns

**Operational Concept Description**

**Overview**

Provides the overall context of the proposed system and its operational concept

**Purpose**

Describe the overall context of the system to be developed, why it's being built, what scope, how, and when the project is started from

- Describe to the stakeholders of the system to be developed (Developed) (intended to provide system terms as technology, "engineering")
- Remember! Ask the system will work in practice once it is deployed
- Enable the operational stakeholders to evolve knowledgeably from their current operational context to the new operational concept, and to collaboratively adapt the operational concept as developments arise, to make clear the value of developing the new system

**Owner**

The artifact Operational Concept Description is owned by the agent System Analyst

**Decomposition**

The artifact Operational Concept Description is decomposed into the following:

- Domain Description
- Proposed System
- Common Definition Language for Domain

**Description**

Operational Concept Description

- Overview
- Purpose
- Decomposition
- Periodicity
- Storage Location
- Files
- Behavior
- Behavior Index
- Transition Period
- Templates
- Examples

Operational Concept Description and Participants

- Audience
- Customer for Domain Description Domain
- Domain Expert for System Analysis
- Participants
- Use language and define CDE, appropriately for intended audience
- Gain stakeholders as WinWin negotiation
- Establish concept of operation agreed on by all stakeholders

Operational Concept Description High-Level Dependencies

- WinWin Negotiation: Done
- System Responsibilities
- Changes Considered But Not Included
- Domain Description Terms
- Project Goals, Quality Goals
- OCG Needs
- Project System and Quality Req for SSRD
- Domain Description and Initial Reports for SSAD
- Stakeholder and Organizational Responsibilities



## Spiral Invariant 6: Emphasis on System and Life Cycle Activities and Artifacts

- **Why invariant**
  - Avoids premature suboptimization on hardware, software, or development considerations.
    - Scientific American
- **Variants**
  - 6a. Relative amount of hardware and software determined in each cycle.
  - 6b. Relative amount of capability in each life cycle increment
  - 6c. Degree of productization (alpha, beta, shrink-wrap, etc.) of each life cycle increment.
- **Models excluded**
  - Purely logical object-oriented methods
    - Insensitive to operational, performance, cost risks

2/9/00

©USC-CSE

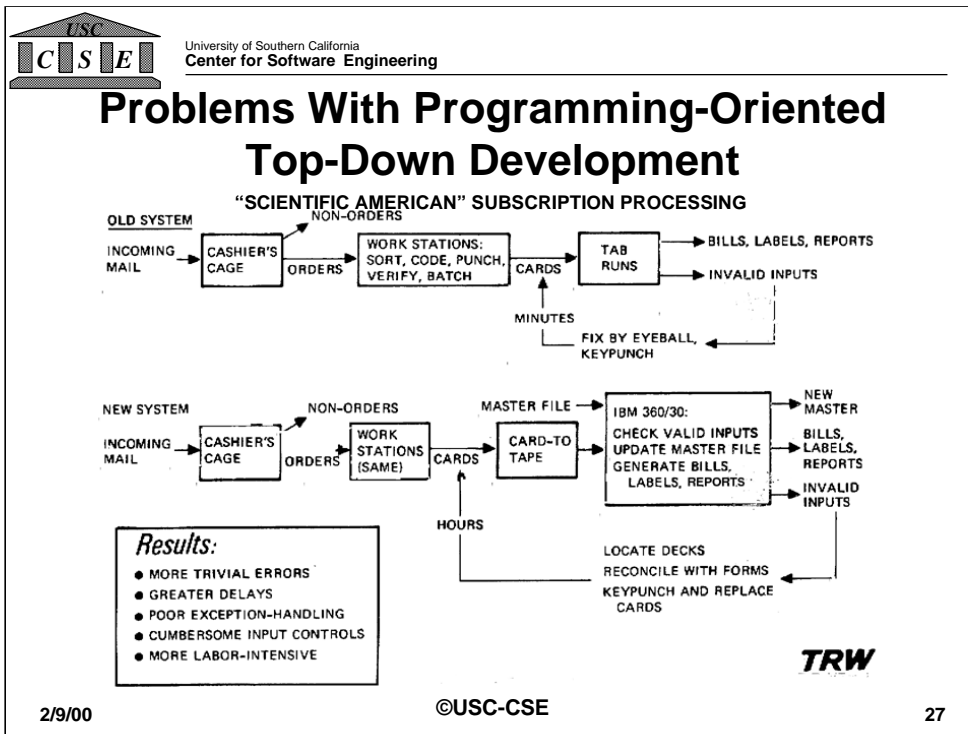
26

Spiral invariant 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. Software developers are particularly apt to fall into the oft-cited trap: “If your only tool is a hammer, the world begins to look like a collection of nails.”

A good example is the Scientific American case study shown in the next chart. 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.

The spiral model’s emphasis on using stakeholder objectives to drive system solutions, and on the life cycle anchor point milestones, guides projects to focus on system and life cycle concerns. Its use of risk considerations to drive solutions enables projects to tailor each spiral cycle to whatever mix of software and hardware, choice of capabilities, or degree of productization is appropriate.

Models excluded by invariant 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 6 had the word “performance” in its index, and only 2 had the word “cost” in its index.



Scientific American's objectives were to reduce their subscription processing system's costs, errors, and delays. Rather than analyze the system's sources of costs, errors, and delays, the software house jumped in and 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 which had been the major source of the costs, errors, and delays in the first place. As seen in the chart, the business outcome was a new system with more errors, greater delays, higher costs, and less attractive work than its predecessor [Boehm, 1981].

This kind of outcome would have happened 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 re-engineer the clerical business processes as well as to automate the manual tab runs. Further, as shown by recent methods such as the DMR Benefits Realization Approach [Thorp, 1998], the business case could have been used to monitor the actual realization of the expected benefits, and to apply corrective action to either the business process re-engineering or the software engineering portions of the solution (or both) as appropriate.

## Summary: Hazardous Spiral Look-Alikes

- **Incremental sequential waterfalls with significant COTS, user interface, or technology risks**
- **Sequential spiral phases with key stakeholders excluded from phases**
- **Risk-insensitive evolutionary or incremental development**
- **Evolutionary development with no life-cycle architecture**
- **Insistence on complete specs for COTS, user interface, or deferred-decision situations**
- **Purely logical object-oriented methods with operational, performance, or cost risks**
- **Impeccable spiral plan with no commitment to managing risks**

As with many methods, the spiral model has a number of “hazardous look-alikes,” which have been discussed in the previous charts.

Incremental sequential waterfalls with significant COTS, user interface, or technology risks are discussed in Charts 6 and 7. Sequential spiral phases with key stakeholders excluded from phases are discussed in Charts 8, 9, and 10. Risk-insensitive evolutionary or incremental development is discussed in Charts 11, 16, and 20, as is evolutionary development with no life-cycle architecture. Insistence on complete specs for COTS, user interface, or deferred-decision situations is discussed in Charts 14 and 15. Purely logical object-oriented methods with operational, performance, or cost risks are discussed in Chart 26. Impeccable spiral plans with no commitment to managing risks are discussed in Charts 11 and 23.

## Summary: Successful Spiral Examples

- **Rapid commercial: C-Bridge's RAPID process**
- **Large commercial: AT&T/Lucent/Telcordia spiral extensions**
- **Commercial hardware-software: Xerox Time-to-Market process**
- **Large aerospace: TRW CCPDS-R**
- **Variety of projects: Rational Unified Process, SPC Evolutionary Spiral Process, USC MBASE approach**

A number of successful spiral approaches satisfying the spiral model invariants were presented at the workshop, often with supplementary material elsewhere. C-Bridge's RAPID approach has been used successfully to develop e-commerce applications in 12-24 weeks. Its Define, Design, Develop, and Deploy phases use the equivalent of the LCO, LCA and IOC anchor point milestones as phase gates [Leinbach, 2000]. The large spiral telecommunications applications discussed in [Bernstein, 2000] and [DeMillo, 2000] use a complementary best practice at their anchor point milestones: the AT&T/Lucent/Telcordia Architecture Review Board process [AT&T, 1993]. Xerox's Time-to-Market process uses the anchor point milestones as hardware-software synchronization points for its printer business line [Hantos, 2000].

Several successful large aerospace spiral projects were also discussed. The best documented of these is the CCPDS-R project discussed in [Royce, 1998]. Its Ada Process Model was the predecessor of the Rational Unified Process and USC MBASE approach, which have been used on a number of successful spiral projects [Jacobson et al., 1999; Boehm et al., 1998], as has the SPC Evolutionary Spiral Process [SPC, 1994].

## References

(MBASE material available at <http://sunset.usc.edu/MBASE>)

- [AT&T, 1993]. "Best Current Practices: Software Architecture Validation," AT&T, Murray Hill, NJ 1993.
- [Bernstein, 2000]. L. Bernstein, "Automation of Provisioning," Proceedings, USC-SEI Spiral Experience Workshop, February 2000.
- [Boehm, 1988]. "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.
- [Boehm, 1989]. "Software Risk Management", IEEE Computer Society Press, 1989.
- [Boehm, 2000]. B. Boehm, "Unifying Software Engineering and Systems Engineering," IEEE Computer, March 2000, pp. 114-116.
- [Boehm et al., 1997] "Developing Multimedia Applications with the WinWin Spiral Model," Proceedings, ESEC/FSE 97, Springer Verlag, 1997.
- [Boehm et al., 1998]. "Using the Win Win Spiral Model: A Case Study," IEEE Computer, July 1998, pp. 33-44.
- [Boehm et al., 2000]. B. Boehm, M. Abi-Antoun, A.W. Brown, N. Mehta, and D. [Port, 2000]. "Guidelines for the LCO and LCA Deliverables for MBASE," USC-CSE, March 2000, [http://sunset.usc.edu/classes/cs577b\\_2000/EP/07/MBASE\\_Guidelines\\_for\\_CS577v0.2.pdf](http://sunset.usc.edu/classes/cs577b_2000/EP/07/MBASE_Guidelines_for_CS577v0.2.pdf)
- [Boehm-Ross, 1989]. "Theory W Software Project Management: Principles and Examples" IEEE Trans. Software Engr., July 1989.
- [Boehm-Bose, 1994]. "A Collaborative Spiral Software Process Model Based on Theory W," Proceedings, ICSP 3, IEEE, Reston, Va. October 1994.
- [Boehm-Port, 1999a]. "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them," ACM Software Engineering Notes, January, 1999, pp. 36-48.
- [Boehm-Port, 1999b]. "When Models Collide: Lessons from Software Systems Analysis," IEEE IT Professional, January/February 1999, pp. 49-56.
- [Carr et al., 1993]. "Taxonomy-Based Risk Identification," CMU/SEI-93-TR-06, Software Engineering Institute, 1993.
- [Charette, 1989]. Software Engineering Risk Analysis and Management, McGraw Hill, 1989.
- [Cusumano-Selby, 1995] Microsoft Secrets, Free Press, 1995

- [DeMillo, 2000]. R. DeMillo, "Continual Improvement: Spiral Software Development", Proceedings, USC-SEI Spiral Experience Workshop, February 2000.
- [Hall, 1998] Managing Risk, Addison Wesley, 1998.
- [Hantos, 2000]. P. Hantos, "From Spiral to Anchored Processes: A Wild Ride in Lifecycle Architecting", Proceedings, USC-SEI Spiral Experience Workshop, February 2000.
- [Forsberg et al., 1996]. K. Forsberg, H. Mooz, and H. Cotterman, Visualizing Project Management, Wiley, 1996.
- [Garlan et al., 1995]. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse Is So Hard," IEEE Software, November 1995, pp. 17-26.
- [Jacobson et al., 1999]. The Unified Software Development Process, Addison-Wesley, 1999.
- [J. Thorp, 1998]. The Information Paradox, McGraw Hill, 1998.
- [Kellner et al., 1998]. "Process Guides: Effective Guidance for Process Participants," Proceedings of the 5th International Conference on the Software Process: Computer Supported Organizational Work, 1998.
- [Kitaoka, 2000]. B. Kitaoka, "Yesterday, Today & Tomorrow: Implementations of the Development Lifecycles", Proceedings, USC-SEI Spiral Experience Workshop, February 2000.
- [Kruchten, 1999]. The Rational Unified Process, Addison-Wesley, 1998.
- [Leinbach, 2000]. C. Leinbach, "E-Business and Spiral Development", Proceedings, USC-SEI Spiral Experience Workshop, February 2000.
- [Mehta, 1999]. N. Mehta, "MBASE Electronic Process Guide," USC-CSE, October 1999, [http://sunset.usc.edu/classes/cs577a\\_99/epg](http://sunset.usc.edu/classes/cs577a_99/epg)
- [Royce, 1998] Software Project Management: A Unified Framework, Addison Wesley, 1998.
- [SPC, 1994]. Software Productivity Consortium, "Process Engineering with the Evolutionary Spiral Process Model," SPC-93098-CMC, version 01.00.06, Herndon, Virginia, 1994.
- [USAF, 2000]. U.S. Air Force, "Evolutionary Acquisition for C2 Systems," Air Force Instruction 63-123, 1 January 2000.