The Second Principle: Incremental Commitment and Accountability

Trust is built through effective commitments. Trust is lost through inconsistent behavior on matters of importance. If trust has been cultivated and grown over time ... the project will be highly resilient to problems.

Scott Berkun, The Art of Project Management, Chapter 12, O’Reilly, 2005

Don’t promise more than you can deliver.
Anonymous proverb

Successful system development projects are built on a bedrock of trust. If people don’t trust each other, they will cut down on sharing information and helping each other out. They will gird themselves in contractual overhead and bureaucracy, and build Pearl Harbor files to be able to blame others if the project gets into trouble – and such behavior is often what gets the project into trouble. And as stated above, trust is built through effective commitments.

Watts Humphrey, in Managing the Software Process, (Addison Wesley, 1989), has identified six critical elements of effective commitments:

1. The person making the commitment does so willingly.
2. The commitment is not made lightly; that is, the work involved, the resources, and the schedule are carefully considered.
3. There is agreement between the parties as to what is to be done, by whom, and when.
4. The commitment is openly and publicly stated.
5. The person responsible tries to meet the commitment, even if help is needed.
6. Prior to the committed date, if something changes that impacts either party relative to the commitment, advance notice is given and a new commitment is negotiated.

This may look like a heavyweight set of conditions, but if trust is built up among people initially unfamiliar with each other, the content of the commitments may be based on stories or capability descriptions rather than complete, consistent, traceable, testable requirements. And if meeting a fixed schedule is critical, the agreement may include the option of dropping lower-priority features to meet the schedule. Another definition of trust is that each party has confidence that the other parties feel accountable for the commitments they have made.

If you are going to be accountable for a commitment, it is important not to promise more than you can deliver. In a world of rapid change in mission priorities, new technologies, competitive challenges, and
organizational relationships, this means that total commitments to the full details of a product to be delivered five years later is a very risky bet. This is why incremental commitments are becoming more satisfactory than total up-front commitments.

An important concern in this regard is the Cone of Uncertainty shown in Figure 2-1 (Bauman. Early in a system’s life cycle, there are many possible system capabilities and solutions to consider, leading to a wide range of system costs. Particularly in competitive procurements, it is tempting to make bids and proposals at the lower edge of the Cone of Uncertainty. This is often rationalized by making optimistic assumptions (the A team will perform on the project; the COTS products will do everything right; the prototypes will just take a little effort to make into products). Often, such temptations will be met by similar behavior by acquirers who are trying to sell their programs, in a “conspiracy of optimism.” The usual result is that the project’s actual costs climb up the lower edge of the Cone, creating a large overrun.

Even with less optimistic assumptions, committing to a fixed-price contract at an early date would run a large risk of promising more than you can deliver and overrunning your agreed-upon budget. Instead, it is preferable to do some prototyping, architecting, or COTS evaluations before committing to a fixed price (or, as discussed in the Introduction, to use a poker or blackjack approach rather than a roulette approach).

![Figure 2-1. The Cone of Uncertainty](perhaps box explaining it)

We next present an example of a total-commitment project that failed and an example of an incremental-commitment project that was highly successful.
2.1 A Failed Total-Commitment Project: Bank of America’s MasterNet

In the 1950s and 1960s, Bank of America (BofA) was the leading pioneer in banking automation with its electronic check processing capability. Subsequent BofA leaders had other interests, allowing BofA’s banking automation capabilities to degrade. In 1981, BofA’s new president, Sam Armacost, had an agenda to regain its automation leadership by “leapfrogging into the 1990s.” After an in-house effort that spent $6 million and failed to develop a workable trust management system, Armacost appointed a new Executive VP of the trust management department, Clyde Claus, with the charge of either modernizing the department or discontinuing it.

2.1.1 Case Study History

In a partial implementation of ICSM Principle 1, Claus set up a Voice of the Customer approach to interview representatives of the various banks and trust management organizations that would be employing MasterNet’s services to determine what services the system should provide them. This resulted in a very large set of requirements, covering over 100 types of assets and the need to comply with numerous Federal and State requirements. The resulting software comprised 3.5 million lines of code.

Given the 1981 BofA in-house development failure, Claus began looking for an external company with a successful trust management system development record, and reusable software assets that could enable a large amount of trust management software to be built quickly. This search ended with Claus entering negotiations with Steven Katz, the CEO of Premier Systems, which had developed several successful trust management systems for smaller banks. Katz indicated that if he could reuse his current trust management software, he could develop the full set of desired capabilities in 9 months and $22 million.

Unfortunately, Claus’ systems engineering personnel had been transferred to a corporate systems engineering group, and he was unable to perform a detailed analysis of Katz’ proposal. Since BofA didn’t see any superior options, Claus and his management went ahead and signed a contract for Katz to proceed in March 1984, with the full system delivery due in December 1984. By then, Katz had 100 programmers working on the project, but it was far from complete, as the software was not as reusable as expected, it did not scale up well to BoFA’s large-bank needs, and there were numerous “devils in the details” that slowed the development down.

BoFA’s management did not wish to announce another failure. Instead, they agreed to add more budget and schedule to build a critical-mass trust management capability. After a large number of features had been developed in 1985-86, Claus decided to showcase the new MasterNet system in a major public relations event to demonstrate “the industry’s most sophisticated technology for handling trust accounts.” The event went moderately well, but it did not demonstrate scalability because the Prime computer hardware and software supporting Premier Systems’ trust management capabilities could not scale up to BoFA’s trust management workload.

During 1986, Claus began preparing for the conversion and cutover to Master Net, beginning with the transfer of the $38 billion worth of institutional trust customers. The initial platform was a 1 million instructions per second (MIPS), 8 megabytes of main memory Prime computer, which was seriously underpowered. Even with an upgrade to a three 16-megabyte, 8-MIPS processor configuration (along with a lot of operating system and data management system rework), the system could not process the workload. Subsequently, 21 of the 24 Prime disk drives failed and needed to be replaced.

System problems continued during 1987. Two more processors were added, but crashes continued, and monthly reports were up to two months late. Clients began dropping off, from 800 to 700 accounts and from $38 billion to $34 billion in institutional assets. The Prime-based trust system did not interoperate well with the rest of BofA’s IBM mainframe systems. Eventually, in May 1988, BoFA transferred its whole trust business to other banks, after an overall expenditure of $80 million and over four years of project effort.
Armacost had been replaced by the previous president, Tom Clausen, in late 1986, and Claus had resigned in October 1987. Further descriptions of MasteNet are in (Flowers, 1998) and (Glass, 1998).

2.1.2 Relation to ICSM Principles

There were numerous contributing factors to the failed MasterNet project, but the main one was the total-commitment violation of Principle 2 to entrust Katz and Premier Systems with the implementation of the full 3.5 million lines of software, without incrementally determining whether his solution was feasible and compatible with the rest of BofA’s IBM mainframe systems. By overfocusing on the Voice of the Customer in committing to their full wish list of capabilities, and neglecting the voices of the maintainers in choosing an incompatible Prime-based system, and the voices of BofA top management and investors in neglecting to perform a full risk analysis of the project’s feasibility, the project was also in violation of Principle 1 of considering the interests of all the success-critical stakeholders. And as we will see, it was also in violation of Principle 4 on evidence-based and risk-based decision making.

Figure 2-2 summarizes the results of several studies performed at USC on the root causes of failed projects [Boehm and Port 1999; Boehm-Port-Al-Said 2000; and Al-Said 2003]. The studies concluded that the main causes of project failure tended to be due to projects proceeding into a total-commitment development approach without considering issues of development feasibility with respect to the full range of success-critical stakeholders. The studies found that the key stakeholder classes common to virtually all projects (Users, Acquirers, Developers, Maintainers) consistently had serious incompatibilities among their top value propositions or success models.

The upper left part of Figure 2-2 shows the most frequent value propositions or win conditions of people in the User role (for MasterNet, the trust departments of banks for whom they were creating MasterNet, i.e., their “voice of the customer.” The “Many features” product model (PD) added up to 3.5 million lines of code. The “Applications compatibility” product model (PD) meant interoperability with their other IBM mainframe/OS-360/COBOL applications. The “High levels of service” property model (PP) meant near-instant response time, 24/7 availability, and ease of use, among others. The “Voice in acquisition” process model (PC) often meant, “I’m not sure I’ll need this, but since you’re asking, I’ll put it on the list.” The “Early availability” property model (PP) meant, “We need it all as soon as possible.”

The BofA Acquirers in the upper right part of Figure 2-2, bereft of much systems engineering support, interpreted the “Mission cost-effectiveness” success model (S) as committing to satisfy the full set of User wishes on a highly ambitious ($22 million) budget and (9 month) schedule. They tried to achieve this by choosing Premier Systems and their reusable models and software, but Premier Systems’ Product model (PD) value proposition of Developer “Freedom of choice: COTS/reuse” -- using Prime computer equipment -- clashed with the IBM mainframe/OS-360/COBOL applications compatibility Product model (PD) of the Users and Maintainers; the high levels of service (performance, reliability) Property (PP) model of the Users, and the ease of transition and maintenance Product models (PD) of the Maintainers. Further explanations are in [Boehm and Port 1999; Boehm-Port-Al-Said 2000; and Al-Said 2003].

The red or gray lines in Figure 2-2 show that the MasterNet project was caught in a “spider web” of the most frequent Product-Process-Property-Success “model clashes” among the most common stakeholder roles of User, Acquirer, Developer, and Maintainer, that can cause projects to fail. The black lines come from analyses of other failed projects.

Still, there was a great deal of uncertainty about the ability of Premier Systems to develop at least a partial face-saving trust management system, and the MasterNet management decided to go ahead with a total commitment to their approach. This was in violation of ICSM Principle 2, along with violations of Principle 1 in neglecting the Maintainer and User value propositions of application compatibility, ease of transition, and ease of maintenance. Most serious, though, was the decision to go forward with no evidence that the Premier solution was feasible with respect to the other value propositions, in violation of Principle 4.
2.2. A Successful Incremental-Commitment Project: The TRW Software Productivity System

The spiral model was originally developed at TRW in 1978 as a way to help the company evolve away from its overcommitment to the waterfall model in its corporate software development policies and standards. Its initial formulation showed how a project could use risk considerations to determine whether a project should be done as a pure waterfall process, a pure evolutionary prototyping process, or a mix of the two. But it was only first used for the full definition and development of a software-intensive system when TRW embarked on the development of a corporate Software Productivity System (SPS), as part of a corporate initiative to improve productivity in all of its divisions.

Since TRW was using its own money rather than the Government’s money to fund the project, it found the spiral model to be a good way to converge incrementally on the definition of the SPS, and to develop the system incrementally while being linked to an initial user project rather than being fully developed before being offered to user projects. This enabled the initial Spiral Model project to also be the initial Incremental Commitment Spiral Model project. The description below of the start of the project comes from [Boehm-Penedo 2009].

2.2.1 Getting Started: The Exploration Phase

Scene: Bob Williams’ office, late 1979.

Bob is the Vice President/General Manager of the 2000-person Software and Information Systems Division, one of six Divisions in the TRW Defense and Space Systems Group. Barry is his Chief Engineer and Advanced Technology Business Area manager.
Bob: I’ve just come back from a DSSG General Managers’ offsite about improving productivity. Corporate in Cleveland is making a big push to get the auto parts divisions to be more competitive with the Japanese, and wants everybody in TRW to focus on improving their productivity. It looks like the company will put up money for productivity initiatives if there’s a good business case for them. I think it’s worth a try. Do you think you can put something together for us?

Barry: Sure. This fits with a lot of improvements we’ve talked about but haven’t found funding for. Our TRW version of the COCOMO model provides us with a good framework for a business case. It shows how much our productivity goes up or down as we change some of the cost drivers like tool support, turnaround time, reusing components, and people factors. This last would fit with your ideas about multiple career paths for our people. We could probably use some of our local area network technology to get everybody interactively working and communicating. And we could probably get added support from some of the Defense Department’s Ada initiatives. Is Corporate looking for a full-up proposal?

Bob: Well, if we were proposing to spend the Government’s money, that’s what we would do. But here they’re spending the Company’s money, and want a clearer idea of their options and what everybody’s ideas are before they commit to spend a lot of money. So we have a couple of months to put a white paper together. Why don’t you do a part-time study with Ray Wolverton and a couple of the Ada guys and put a draft together. And let’s get everybody involved by doing a survey of what people think would best help them improve their productivity.

Barry: Great. We’ll get right on it and give you a progress report in a couple of weeks.

From this starting point, Tables 2-1, 2-2, and 2-3 summarize the progression of the TRW Software Productivity System (SPS) project through what are now called the ICSM Stage I Exploration, Valuation, and Foundations phases. Subsequently, its ICSM Stage II Development and Operations activities were incremental, in that Increment 1 focused on the SPS infrastructure and the early life-cycle tools needed first by the pilot project, with subsequent increments focusing on the pilot project’s later-phases tool needs.

**SPS Exploration Phase.**

This phase involved five part-time participants over a two-month period. As indicated in Table 2-1, the objectives and constraints were expressed at a very high level and in qualitative terms like “significantly increase,” “at reasonable cost,” etc.

Some of the alternatives considered, primarily those in the “technology” area, could lead to development of a software product, but the possible attractiveness of a number of non-software alternatives in the management, personnel, and facilities areas could have led to a conclusion not to embark on a software development activity.

The primary risk areas involved possible situations in which the company would invest a good deal only to find that

- Resulting productivity gains were not significant
- Potentially high-leverage improvements were not compatible with some aspects of the “TRW culture”

The risk-resolution activities undertaken in the Exploration phase were primarily surveys and analyses, including structured interviews of software developers and managers; an initial analysis of productivity leverage factors identified by the constructive cost model (COCOMO) and an analysis of previous projects at TRW exhibiting high levels of productivity.
Table 2.1. TRW Software Productivity System, Exploration Phase

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Significantly increase software productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraints</td>
<td>At reasonable cost</td>
</tr>
<tr>
<td></td>
<td>Within context of TRW culture</td>
</tr>
<tr>
<td></td>
<td>• Government contracts, high tech, people oriented security</td>
</tr>
<tr>
<td>Alternatives</td>
<td>Management: Project organization, policies, planning, control</td>
</tr>
<tr>
<td></td>
<td>Personnel: Staffing, incentives, training</td>
</tr>
<tr>
<td></td>
<td>Technology: Tools, workstations, methods, reuse</td>
</tr>
<tr>
<td></td>
<td>Facilities: Offices, communications</td>
</tr>
<tr>
<td>Risks</td>
<td>May be no high-leverage improvements</td>
</tr>
<tr>
<td></td>
<td>Improvements may violate constraints</td>
</tr>
<tr>
<td>Risk resolution</td>
<td>Internal surveys</td>
</tr>
<tr>
<td></td>
<td>Analyze cost model</td>
</tr>
<tr>
<td></td>
<td>Analyze exceptional projects</td>
</tr>
<tr>
<td></td>
<td>Literature search</td>
</tr>
<tr>
<td>Risk resolution results</td>
<td>Some alternatives infeasible</td>
</tr>
<tr>
<td></td>
<td>• Single time-sharing system: Security</td>
</tr>
<tr>
<td></td>
<td>Mix of alternatives can produce significant gains</td>
</tr>
<tr>
<td>Plan for next phase</td>
<td>Six-person task force for six months</td>
</tr>
<tr>
<td></td>
<td>More extensive surveys and analysis</td>
</tr>
<tr>
<td></td>
<td>• Internal, external, economic</td>
</tr>
<tr>
<td></td>
<td>Develop analysis of alternatives, top-level concept of operation and business case</td>
</tr>
<tr>
<td>Commitment</td>
<td>Fund next phase</td>
</tr>
</tbody>
</table>

The risk analysis results indicated that significant productivity gains could be achieved at a reasonable cost by pursuing an integrated set of initiatives in the four major areas. However, some candidate solutions, such as a software support environment based on a single, corporate, maxicomputer-based time-sharing system, were found to be in conflict with TRW constraints requiring support of different levels of security-classified projects. Thus, even at a very high level of generality of objectives and constraints, the Exploration phase was able to answer basic feasibility questions and eliminate significant classes of candidate solutions.

The plan for the Valuation phase involved commitment of 12 person-months compared to the two person-months invested in Exploration (during these phases, all participants were part-time). As with the current Valuation phase, its intent was to produce an analysis of alternatives and a recommended top-level concept of operation and business case.

SPS Valuation Phase.

Table 2-2 summarizes the Valuation phase along the lines given in Table 1 for the Exploration phase.

Table 2-2. TRW Software Productivity System, Valuation Phase
The features of Table 2-2 compare to those of Table 2-1 as follows:

- The level of investment was greater (12 versus 2 person-months).
- The objectives and constraints were more specific (“double software productivity in five years at a cost of $10,000 per person” versus “significantly increase productivity at a reasonable cost”).
- Additional constraints surfaced, such as the preference for TRW products [particularly, a TRW-developed local area network (LAN) system].
- The alternatives analyzed were more detailed (“SREM, PSL/PSA or SADT, as requirements tools, etc.” versus “tools”; “private/shared” terminals, “smart/dumb” terminals versus “workstations”).
- The risk areas identified were more specific (“TRW LAN price-performance within a $10,000-per-person investment constraint” versus “improvements may violate reasonable-cost constraint”).
- The risk-resolution activities were more extensive (including the benchmarking and analysis of a prototype TRW LAN being developed for another project).
• The result was a fairly specific operational concept document, involving private offices tailored to software work patterns and personal terminals connected to VAX superminis via the TRW LAN. Some choices were specifically deferred to the next round, such as the choice of operating system and specific tools.
• The life-cycle plan and the plan for the next phase involved a partitioning into separate activities to address management improvements, facilities development, and development of the first increment of a software development environment.
• The commitment step involved more than just an agreement with the plan. It committed to apply the environment to an upcoming 100-person pilot software project and to develop an environment focusing on the pilot project’s needs. It also specified forming a representative steering group to ensure that the separate activities were well-coordinated and that the environment would not be overly optimized around the pilot project.

SPS Foundations Phase.

Table 2-3 shows the corresponding steps involved during the Foundations phase of defining the SPS. A key decision was to clarify that the objective was project productivity and not just programmer productivity. TRW’s government projects were highly document-intensive, and integration of software development support and office support was a main objective.

The initial risk-identification activities during the Foundations phase showed that several system requirements hinged on the decision between a host-target system or a fully portable tool set and the decision between VMS and Unix as the host operating system. These requirements included the functions needed to provide a user friendly front end, the operating system to be used by the workstations, and the functions necessary to support a host-target operation. To keep these requirements in synchronization with the others, a special minispiral was initiated to address and resolve these issues. The resulting review led to a commitment to a host-target operation using Unix on the host system, at a point early enough to work the OS dependent requirements in a timely fashion.

Addressing the risks of mismatches to the user-project’s needs and priorities resulted in substantial participation of the user-project personnel in the requirements definition activity. This led to several significant redirections of the requirements, particularly toward supporting the early phases of the software life cycle into which the user project was embarking, such as an adaptation of the software requirements engineering methodology (SREM) tools for requirements specification and analysis.

Besides Unix and its set of tools, the overall set of tools included a number of user-interface incompatibilities (the 12-language syndrome referred to the variety of languages needed for programming, compiling, build-making, version control, job control, requirements, design, test management, etc.). A high-priority effort was to develop a front end for these that minimized their incompatibilities.
Table 2-3. TRW Software Productivity System, Foundations Phase

| Objectives | User-friendly system  
|            | Integrated software, office-automation tools  
|            | Support all project personnel  
|            | Support all life-cycle phases  
| Constraints | Customer-deliverable SDE ⇒ Portability  
|            | Stable, reliable service  
| Alternatives | OS: VMS/AT&T Unix/Berkeley Unix/JISC  
|             | Host-target/fully portable tool set  
|             | Workstations: Zenith/LSI-11/…  
| Risks | Mismatch to pilot-project needs, priorities  
|        | User-unfriendly system  
|        | • 12-language syndrome; experts-only  
|        | Unix performance, support  
|        | Workstation/mainframe compatibility  
| Risk resolution | Pilot-project surveys, requirements participation  
|                | Survey of Unix-using organizations  
|                | Workstation study  
| Risk resolution results | Top-level requirements specification  
|                       | Host-target with Unix host  
|                       | Unix-based workstations  
|                       | Build user-friendly front end for Unix  
|                       | Initial focus on tools to support early phases  
| Plan for next phase | Overall development plan  
|                     | for tools: SREM, RTT, PDL, office automation  
|                     | for front end: compatible language for tools  
|                     | for LAN: Equipment, facilities  
|                     | 15 full-time personnel for 8 months to IOC  
| Commitment | Proceed with plans  

The incremental commitment level was increased from 12 to 36 person-months, followed by a planned increase to 120 person-months for developing the initial operational capability for the pilot project.

**Overall Project Results.**

The Software Productivity System developed and supported using the spiral model avoided the identified risks and achieved most of the system’s objectives. The SPS grew to include over 300 tools and over 1,300,000 instructions; 93 percent of the instructions were reused from previous project-developed, TRW-developed, or external-software packages. By 1986, over 25 projects had used all or portions of the system. All of the projects fully using the system had increased their productivity at least 50%; indeed, most had doubled their productivity (when compared with cost-estimation model predictions of their productivity using traditional methods).

However, one risk area—that projects with non-Unix target systems would not accept a Unix-based host system—was underestimated. Some projects accepted the host—target approach, but for various reasons (such
as customer constraints and zero-cost target machines) a good many did not. As a result, the system was less widely used on TRW projects than expected. This and other lessons learned were incorporated into a spiral approach to developing a next-generation software development environment reflecting new generations of networking, hardware, and software technologies. Further details can be found in [Boehm et al. 1984; Boehm 1988; and Boehm-Penedo 2009].

2.3 The Two Cones of Uncertainty and the ICSM Stages I and II

The primary drivers for the incremental commitment approaches in the ICSM Stages I and II are the two Cones of Uncertainty shown in Figure 2-3. The first Cone of Uncertainty reflects uncertainties in the nature of the system to be developed and in the applicability of various candidate solution approaches. Total commitment to a particular set of requirements and solution approaches during the early phases of system definition can be highly risky (a wide range of uncertainty corresponds with a high probability of lost value, and probability of loss times size of loss is the accepted definition of risk exposure).

Examples from Chapter 1 are the total commitment to the best-possible technical solution in the Too-Good Robot failure story, leading to total loss of the investment in the robot’s development; and the extensive incremental investments by Abbott Laboratories success story, in buying information to reduce risk via market surveys, business case analyses, field studies, prototypes, and safety analyses, as it proceeded through its Exploration, Valuation, and Foundations phases.

The second Cone of Uncertainty reflects uncertainties in whether the best solution determined at the end of Stage I will still be the best solution after a lengthy single-pass, total commitment development period. In times of slow changes in technology, competition, market demands, organizations, and leadership, the single-pass approach was not too risky. But now and increasingly in the future, the pace of changes in these factors is rapidly increasing, and total commitment to a 3-year single-pass development of a new system is likely to find that the 3-year-old delivered solution is obsolete or noncompetitive.

Again with respect to the examples in Chapter 1, the Too Good Robot project could have delivered a scalable but simpler initial capability, but opted for the single-pass full capability. The Hospira infusion pump project could have tried for a full product line of up to 6-channel pumps, but instead started with an initial offering of combinable 1- and 2-channel pumps and a scalable architecture for more complex future offerings.

Further discussion of the risks to be avoided in total-commitment approaches in complex multi-stakeholder systems are discussed next, in the context of addressing clashes among the various stakeholders’ desired product, process, property, and success models, using the MasterNet failure story as an example.

2.3.1 Comparison to the TRW Software Productivity System (TRW-SPS) Case Study

Although it used the cost drivers in a software cost estimation model to explore options and estimate likely impacts, the TRW-SPS project did not just define productivity in terms of increasing a project’s delivered source lines of code per person-month. It also surveyed developers and managers to determine what other factors would improve the quality of their products and work life.
The overall result was not only to develop a software development environment and tools, but also to improve the work environment and to develop technical as well as management career paths. Several options were explored and evaluated in increasing detail during each definition phase, and the results, feasibility evidence, and risk assessments reviewed and used to guide decisions at the equivalent of the ICSM Validation, Foundations, and Development Commitment Reviews. And the project was bonded to an initial TRW production project to ensure that the system would meet real users’ needs. The end results were not perfect (the homebuilt forms management package was expensive to maintain and fell behind COTS capabilities, and the lack of platform-independence limited its usage), but software productivity and personnel satisfaction were significantly improved.

2.4 Alternative Incremental and Evolutionary Development Models

The primary models of incremental and evolutionary development focus on different competitive and technical challenges. The time phasing of each model is shown in Figure 2-4 below in terms of the increment (1, 2, 3, ...) content with respect to the definition (Df), development (Dv), and production, support, and utilization (PSU) stages used in the Life Cycle Models knowledge area in the Systems Engineering Body of Knowledge (Pyster et al., 2012-), whose figures were adapted from (Boehm and Lane, 2010). The Definition stage corresponds to the ICSM Stage I; the Development stage corresponds to the ICSM Development phase for each increment; and the PSU stage corresponds to the ICSM Operations and Production phase for the increment.
Figure 2-4. *Primary Models of Incremental and Evolutionary Development*

The Figure 2-4 notations (Df_{1..N} and Dv_{1..N}) indicate that their initial stages produce specifications not just for the first increment, but for the full set of increments. These are assumed to remain stable for the pre-specified sequential model but are expected to involve changes for the evolutionary concurrent model. The latter’s notation (Dv_1 and Df_{3R}) in the same time frame, PSU_1, Dv_2 and Df_{3R} in the same time frame, etc.) indicates that the plans and specifications for the next increment are being re-baslined by a systems engineering team concurrently with the development of the current increment and the PSU of the previous
increment. This offloads the work of handling the change traffic from the development team and significantly improves its chances of finishing the current increment on budget and schedule.

In order to select an appropriate life cycle model, it is important to first gain an understanding of the main archetypes and where they are best used. Table 2-4 summarizes each of the primary models of single-step, incremental and evolutionary development in terms of examples, strengths, and weaknesses, followed by explanatory notes.

**Table 2-4. Primary Models of Incremental and Evolutionary Development**

The Pre-specified **Single-step** and Pre-specified **Multi-step** models from Table 2-4 are not evolutionary.

<table>
<thead>
<tr>
<th>Model</th>
<th>Examples</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-specified</td>
<td>Simple manufactured products: Nuts, bolts, simple sensors</td>
<td>Efficient, easy to verify</td>
<td>Difficulties with rapid change, emerging requirements (complex sensors, human-intensive systems)</td>
</tr>
<tr>
<td>Single-step</td>
<td>Vehicle platform plus value-adding pre-planned product improvements</td>
<td>Early initial capability, scalability when stable</td>
<td>Emergent requirements or rapid change, architecture breakers</td>
</tr>
<tr>
<td>Pre-specified</td>
<td>Small: Agile</td>
<td>Adaptability to change, smaller human-intensive systems</td>
<td>Easiest-first, late, costly fixes, systems engineering time gaps, slow for large systems</td>
</tr>
<tr>
<td>Multi-step</td>
<td>Larger: Rapid fielding</td>
<td>Mature technology upgrades</td>
<td>Emergent requirements or rapid change, SysE time gaps</td>
</tr>
<tr>
<td>Evolutionary</td>
<td>Stable development, Maturing technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>Rapid, emergent development, systems of systems</td>
<td>Emergent requirements or rapid change, stable development increments, SysE continuity</td>
<td>Overkill on small or highly stable systems</td>
</tr>
</tbody>
</table>

Pre-specified multi-step models split the development in order to field an early initial operational capability, followed by several pre-planned product improvements (P3Is). An alternate version splits up the work but does not field the intermediate increments. When requirements are well understood and stable, the pre-specified models enable a strong, predictable process. When requirements are emergent and/or rapidly changing, they often require expensive rework if they lead to undoing architectural commitments.

The **Evolutionary Sequential** model involves an approach in which the initial operational capability for the system is rapidly developed and is upgraded based on operational experience. Pure agile software development fits this model. If something does not turn out as expected and needs to be changed, it will be fixed in thirty days at the time of its next release. Rapid fielding also fits this model for larger or hardware-software systems. Its major strength is to enable quick-response capabilities in the field. For pure agile, the model can fall prey to an easiest-first set of architectural commitments which break when, for example, system developers try to scale up the workload by a factor of ten or to add security as a new feature in a later increment. For rapid fielding, using this model may prove expensive when the quick mash-ups require extensive rework to fix incompatibilities or to accommodate off-nominal usage scenarios, but the rapid results may be worth it.

The **Evolutionary Opportunistic** model can be adopted in cases that involve deferring the next increment until: a sufficiently attractive opportunity presents itself, the desired new technology is mature enough to be added, or until other enablers such as scarce components or key personnel become available. It is also appropriate for synchronizing upgrades of multiple commercial-off-the-shelf (COTS) products. It may be expensive to keep the SE and development teams together while waiting for the enablers, but again, it may be worth it.
The *Evolutionary Concurrent* model involves a team of systems engineers concurrently handling the change traffic and re-baselining the plans and specifications for the next increment, in order to keep the current increment development stabilized. An example and discussion are provided in Table 2-5, below.

### 2.4.1 Incremental and Evolutionary Development Decision Table

Table 2-5 provides some criteria for deciding which of the processes associated with the primary classes of incremental and evolutionary development models to use.

<table>
<thead>
<tr>
<th>Model</th>
<th>Stable, pre-specifiable requirements?</th>
<th>OK to wait for full system to be developed?</th>
<th>Need to wait for next-increment priorities?</th>
<th>Need to wait for next-increment enablers*?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-specified Single-step</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-specified Multi-step</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolutionary Sequential</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Evolutionary Opportunistic</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolutionary Concurrent</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Example enablers: Technology maturity; External-system capabilities; Needed resources; New opportunities

The *Pre-specified Single-step* process exemplified by the traditional waterfall or sequential Vee model is appropriate if the product’s requirements are pre-specifiable and have a low probability of significant change and if there is no value or chance to deliver a partial product capability. A good example of this would be the hardware for an earth resources monitoring satellite that would be infeasible to modify after it goes into orbit.

The *Pre-specified Multi-step* process splits up the development in order to field an early initial operational capability and several P3I’s. It is best if the product’s full capabilities can be specified in advance and are at a low probability of significant change. This is useful in cases when waiting for the full system to be developed incurs a loss of important and deliverable incremental mission capabilities. A good example of this would be a well-understood and well-prioritized sequence of software upgrades for the on-board earth resources monitoring satellite.

The *Evolutionary Sequential* process develops an initial operational capability and upgrades it based on operational experience, as exemplified by agile methods. It is most need in cases when there is a need to get operational feedback on an initial capability before defining and developing the next increment’s content. A good example of this would be the software upgrades suggested by experiences with the satellite’s payload, such as what kind of multi-spectral data collection and analysis capabilities are best for what kind of agriculture under what weather conditions.

The *Evolutionary Opportunistic* process defers the next increment until its new capabilities are available and mature enough to be added. It is best used when the increment does not need to wait for operational feedback, but it may need to wait for next-increment enablers such as technology maturity, external system capabilities, needed resources, or new value-adding opportunities. A good example of this would be the need
to wait for agent-based satellite anomaly trend analysis and mission-adaptation software to become predictably stable before incorporating it into a scheduled increment.

The *Evolutionary Concurrent* process, as realized in the ICSM (Pew and Mavor 2007; Boehm and Lane 2007) and shown in Figure 2-6, has a continuing team of systems engineers handling the change traffic and re-baselining the plans and specifications for the next increment, while also keeping a development team stabilized for on-time, high-assurance delivery of the current increment and employing a concurrent verification and validation (V&V) team to perform continuous defect detection to enable even higher assurance levels. A good example of this would be the satellite’s ground-based mission control and data handling software’s next-increment re-baselining to adapt to new COTS releases and continuing user requests for data processing upgrades.

![Figure 2-6. Evolutionary- Concurrent Rapid Change Handling and High Assurance](image)

The satellite example illustrates the various ways in which the complex systems of the future, different parts of the system, and its software may evolve in a number of ways, once again affirming that there is no one-size-fits-all process for software evolution. However, Table 2-5 can be quite helpful in determining which processes are the best fits for evolving each part of the system, and the three-team model in Figure 2-6 provides a way for projects to develop the challenging software-intensive systems of the future that will need both adaptability to rapid change and high levels of assurance.

**References**


A. Pyster et al., The Systems Engineering Body of Knowledge (SEBoK), 2012 - . www.sebokwiki.org