The Third Principle: Concurrent Multidiscipline Systems Definition and Development

Do everything in parallel, with frequent synchronizations.

Michael Cusumano and Richard Selby

As the correct solution of any problem depends primarily on a true understanding of what the problem really is, and wherein lies its difficulty, we may profitably pause upon the threshold of our subject to consider first, in a more general way, its real nature: the causes which impede sound practice; the conditions on which success or failure depends; the directions in which error is most to be feared. Thus we shall attain that great perspective for success in any work — a clear mental perspective, saving us from confusing the obvious with the important, and the obscure and remote with the unimportant.

Arthur M. Wellington
The Economic Theory of the Location of Railroads, Wiley, 1887.

The first flowering of systems engineering as a formal discipline focused on the engineering of complex physical systems such as complex ships, aircraft, transportation systems, and logistics systems. The physical behavior of the systems could be well analyzed by mathematical techniques, with passengers treated along with baggage and merchandise as a class of logistical objects with average sizes, weights, and quantities. Such mathematical models were very good in analyzing the physical performance tradeoffs of complex system alternatives. They also served as the basis for the development of elegant mathematical theories of systems engineering.

The physical systems were generally stable, and were expected to have long useful lifetimes. Major fixes or recalls of fielded systems were very expensive, so it was worth investing significant up-front effort in getting their requirements to be complete, consistent, traceable, and testable, particularly if the development was to be contracted out to a choice of competing suppliers. It was important not to overconstrain the solution space, so the requirements were not to include design choices, and the design could not begin until the requirements were fully specified.

Various sequential process models were developed to support this approach, such as the diagonal waterfall model, the V-model (a waterfall with a bend upwards in the middle), and the two-leg model (an inverted V-model). These were effective in developing numerous complex physical systems, and were codified into government and standards-body process standards. The manufacturing process of assembling physical
components into subassemblies, assemblies, subsystems, and system products was reflected in functional-hierarchy design standards, integration and test standards, and work breakdown structure standards as the way to organize and manage the system definition and development.

The fundamental assumptions underlying this set of sequential processes, prespecified requirements, and functional-hierarchy product models began to be seriously undermined in the 1970s and 1980s. The increasing pace of change in technology, competition, organizations, and life in general made assumptions about stable, prespecifiable requirements unrealistic. The existence of cost-effective, competitive, incompatible commercial products or other reusable non-developmental items (NDIs) made it necessary to evaluate and often commit to solution components before finalizing the requirements (the consequences of not doing this will be seen in the failure case study in Chapter 4). The emergence of freely-available Graphic User Interface (GUI) generators made rapid user interface prototyping feasible, and made the prespecification of user interface requirement details unrealistic. The difficulty of adapting to rapid change with brittle, optimized, point-solution architectures generally made optimized first-article design to fixed requirements unrealistic.

As was shown in the “hump diagram” of Figure 6 in the Introduction, the ICSM emphasizes the principle of concurrent rather than sequential work on understanding needs, envisioning opportunities, system scoping, system objectives and requirements determination, architecting and designing of the system and its hardware, software, and human elements, life cycle planning, and development of feasibility evidence. Of course, the humps in Figure 6 are not a one-size-fits-all representation of every project’s effort distribution. In practice, the evidence and risk-based decision criteria discussed in Figures 8 and 9 of the Introduction can determine which specific process model will fit best for which specific situation. This includes situations in which the sequential process is still best, as its assumptions still hold in some situations. Also, since requirements emerge from use, working on all of the requirements and solutions in advance is not feasible, which is where the ICSM Principle 2 of incremental commitment applies.

This establishes the context for the “Do everything in parallel” quote at the beginning of this chapter. Even though preferred sequential-engineering situations still exist in which “Do everything in parallel” does not universally apply, it is generally best to apply it during the first ICSM Exploratory Phase. By holistically and concurrently addressing during this beginning phase all of the system’s hardware, software, human factors, and economic considerations (as in the Wellington quote above), projects will generally be able to determine their process drivers and best process approach for the rest of the system’s life cycle. And as discussed above, the increasing prevalence of process drivers such as emergence, dynamism, and NDI support will make concurrent approaches increasingly dominant.

Thus suitably qualified, we can proceed to the main content of Chapter 3. Our failure and success case studies are two different sequential and concurrent approaches to a representative complex hardware-software-human factors government system acquisition involving remotely piloted vehicle (RPVs). The remaining sections will discuss best practices for concurrent hardware-software-human factors engineering, concurrent requirements and solutions engineering, concurrent development and evolution engineering, and support of more rapid concurrent engineering.

An example to illustrate ICSM concurrent-engineering benefits is the Unmanned Aerial System (UAS) (or Remotely Piloted Vehicle (RPV)) system enhancement discussed in Chapter 5 of the NRC Human-System Integration report [Pew and Mavor 2007]. These RPVs are airplanes or helicopters operated remotely by humans. The systems are designed to keep humans out of harm’s way. However, the current RPV systems are human-intensive, often requiring two people to operate a single vehicle. The increase in need to operate numerous RPVs is causing a strong desire to modify the 1:2 (one vehicle controlled by 2 people) ratio to allow for a single operator to operate more than one RPV, as shown in Figure 3-1 from [Pew and Mavor 2007]
A recent advanced technology demonstration of an autonomous-agent based system enabled a single operator to control 4 RPVs in flying in formation to a crisis area while compensating for changes in direction to avoid adverse weather conditions of no-fly zones. This demonstration of a 4:1 vehicle:controller ratio capability highly impressed senior leadership officials viewing the demo, and they established a high-priority rapid-development program to acquire and field a common agent-based 4:1 RPV control capability for use in battlefield-based, sea-based, and home-country based RPV operations.

3.1 Failure Story: Sequential RPV Systems Engineering and Development

A sequential approach that is representative of several recent government acquisition programs would use the demo results to create the requirements for a proposed program that used the agent-based technology to develop a 4:1 ratio system that enabled a single operator to control 4 RPVs in battlefield-based, sea-based, and home-country based RPV operations. A number of assumptions were made to sell the program at an optimistic cost of $1 billion and schedule of 40 months. Enthusiasm was such that the program, budget, and schedule were established, and a multi-service working group of experienced battlefield-based, sea-based, and home-country based RPV controllers was established to develop the requirements for the system.

The resulting requirements included the need to synthesize status information from multiple on-board and external sensors; to perform dynamic reallocation of RPVs to targets; to perform self-defense functions; to communicate status and observational information to central commanders and other RPV controllers; to control RPVs in the same family but with different releases having somewhat different controls; to avoid harming friendly forces or noncombatants; and to be network-ready with respect to self-identification when entering battle zones; establishing security credentials and protocols; operating in a publish-subscribe environment; and participating in replanning activities based on changing conditions. These requirements were included in a Request for Proposals (RFP) that was sent out to prospective bidders.
The winning bidder provided an even more impressive demo of agent technology and a proposal indicating that all of the problems were well understood, that a preliminary design review (PDR) could be held in 120 days, and that the cost would be only $800M. The program managers and their upper management were delighted at the prospect of saving $200 million of the taxpayers’ money, and established a fixed price contract to develop the 4:1 system to the requirements in the RFP in 40 months, with a System Functional Requirements Review (SFRR) in 60 days and a PDR in 120 days.

At the SFRR, the items reviewed were transcriptions and small elaborations of the requirements in the RFP. They did not include any functions for coordinating the capabilities, and included only sunny-day operational scenarios. There were no capabilities for recovering from outages in the network, from the loss of RPVs, from incompatible sensor data, or for tailoring the controls to battlefield-, sea-, or home country-based control equipment. The contractor indicated that they had hired some ex-RPV controllers who were busy putting such capabilities together.

However, at the PDR, the contractor could not show feasible solutions for several critical and commonly-occurring scenarios, such as coping with network outages, missing RPVs, and inconsistent data; having the individual controllers also coordinate with each other; performing self-defense functions; tailoring the controls to multiple equipment types; and satisfying various network-ready interoperability protocols. As has been experienced in practice [Beidel 2011], such capabilities are much needed and difficult to achieve.

Since the schedule was tight and the contractor had almost run out of systems engineering funds, their management proposed to address the problems by using a “concurrent engineering” approach of having the programmers develop the off-nominal capabilities while the systems engineers were completing the detailed design. Having no other face-saving alternative to declaring the PDR to be failed, the customers declared the PDR to be passed.

Actually, this is a pernicious misuse of “concurrent engineering,” since there is not time to produce feasibility evidence and to synchronize and stabilize the numerous off-nominal approaches taken by the programmers and the detailed designers. The almost-certain result for large systems is one or more off-nominal architecture-breakers that require large amounts of rework and throwaway software to reconcile the inconsistent architectural decisions made by the self-fulfilling “hurry up and code, because we will have a lot of debugging to do” programmers. Figure 3-2 shows the results of such approaches for two large TRW projects, in which 80% of the rework resulted from the 20% of problem fixes resulting from critical off-nominal architecture-breakers [Boehm-Valerdi-Honour 2008].
3.2 Success Story: Concurrent Competitive-Prototyping RPV Systems Development

A concurrent incremental-commitment approach, using the ICSM process and competitive prototyping, would recognize that there were a number of risks and uncertainties involved in going from a single-scenario proof-of-principle demo to a fieldable system needing to operate in more complex scenarios. It would decide that it would be good to use prototyping as a way of buying information to reduce the risks, and would determine that a reasonable first step would be to invest $25M in an Exploration phase. This would initially involve the customer and a set of independent experts developing operational scenarios and evaluation criteria from the requirements in Section 3.1.1 above (to synthesize status information from multiple on-board and external sensors; to perform dynamic reallocation of RPVs to targets; to perform self-defense functions; etc.) These would involve not only the sunny-day use cases but also selected rainy-day use cases involving communications outages, disabled RPVs, and garbled data.

The customer would identify an RPV simulator that would be used in the competition, and would send out a Request for Information to prospective competitors to identify their qualifications to compete. Based on the responses, the customer would then select four bidders to develop virtual prototypes addressing the requirements, operational scenarios, and evaluation criteria, and providing evidence of their proposed agent-based RPV controllers’ level of performance. The customer would then have the set of independent experts evaluate the bidders’ results. Based on the results, it would perform an evidence- and risk-based Valuation Commitment Review to determine whether the technology was too immature to merit further current investment as an acquisition program, or that there were acceptable levels of system performance, cost, and risk to invest a next level of resources in addressing the problems identified and developing initial prototype physical capabilities.

As was discovered much more expensively in the failure case above, the prospects for developing a 4:1 capability were clearly unrealistic, but the prospects for a 1:1 capability were sufficiently attractive to merit another level of investment, corresponding to a Valuation phase. This phase was funded at $75 million, some of the more ambitious key performance parameters were scaled back, the competitors were downselected to three, and some basic-capability but multiple-version physical RPVs were provided for the competitors to control in several physical environments.

The evaluation of the resulting prototypes confirmed that the need to control multiple versions of the RPVs made anything higher than a 1:1 capability infeasible. However, the top two competitors provided sufficient evidence of a 1:1 system feasibility that a Foundations Commitment Review was passed, and $225 million was provided for a Foundations phase.

In this phase, the two competitors not only developed operational RPV versions, but also provided evidence of their ability to satisfy the key performance parameters and scenarios. They also developed an ICSM Development Commitment Review package, including the proposed system’s concept of operation, requirements, architecture, and plans, along with a Feasibility Evidence Description providing evidence that a system built to the architecture would satisfy the requirements and concept of operation, and be buildable within the budgets and schedules in the plan.

The feasibility evidence included a few shortfalls, such as remaining uncertainties in the interface protocols with some interoperating systems, but each of these was covered by a risk mitigation plan in the winning competitor’s submission. The resulting Development Commitment Review was passed, and the winner’s proposed $675 million, 18-month, three-increment Stage II plan to develop an Initial Operational Capability
was adopted. The resulting 1:1 IOC was delivered on budget and two months later than the original 40-month target, with a few lower-priority features deferred to later system increments.

![Sequential and Concurrent Process Timelines](image)

**Figure 3-3. Comparative Timelines**

Of the $1 billion spent, $15 million (M) was spent on the three discontinued Exploration phase competitors, $40M was spent on the two discontinued Valuation phase competitors, and $100M was spent on the discontinued Foundations phase competitor (part of the funds were spent on the customer preparation and the independent experts’ evaluations). Overall, the competitive energy stimulated and the early risks avoided make this a good investment. However, the $125M spent on the experience built up by the losing finalist could be put to good use by awarding the finalist with a contract to perform independent verification and validation (IV&V) of the winner’s deliverables.

Actually, it would be best to announce such an outcome in advance, and to do extensive team building and award fee structuring to make the IV&V activity constructive rather than adversarial. It would also provide the customer with some insurance against downstream vendor lock-in.

While the sequential and concurrent cases have been constructed in an RPV context from representative projects, they show how a premature total commitment without significant modeling, analysis, and feasibility assessment will often lead to large overruns in cost and schedule, and a performance that is considerably less than initially desired. However, by “buying information” early, the concurrent incremental commitment and competitive prototyping approach was able to develop a system with much less late rework than the sequential total commitment approach, and with much more visibility and control over the process.

The competitive prototyping approach spent about $155 million in unused prototypes, but the overall expenditure was only $1 billion as compared to $3 billion for the total-commitment approach, and the capability was delivered in 42 versus 80 months, which indicates a strong return on investment. Further, the funding organizations had realistic expectations of the outcome, so that a 1:1 capability was a successful realization of an expected outcome, rather than a disappointing shortfall from a promised 4:1 capability. And the investment in the losing finalist could be put to good use by capitalizing on their experience to perform an IV&V role.

### 3.3 Concurrent Development and Evolution Engineering

As good as the success story in Section 3.2 appears to be, it could have a fatal flaw that is shared by many outsourced system acquisitions. This is that its primary focus is on satisfying today’s requirements as quickly and inexpensively as possible. This builds architectural decisions into the system that make it difficult to adapt
to new opportunities or competitive threats. From an economic standpoint, it neglects the Iron Law of System Evolution:

For every dollar invested in developing a sustained-use system, be prepared to pay at least two dollars on the system’s evolution.

Data from hardware-intensive systems from [Redman et al., 2008] indicates that the average percentage of life-cycle cost spent on Operations and Support (O&S%) is a relatively small 12% for single-use systems such as missiles, but is 60% for ships, 78% for aircraft, and 84% for ground vehicles. For software-intensive systems, [Koskinen 2010] cites O&S% from seven studies ranging from 60-70% to over 90%.

Still, many projects (and some system acquisition guidance documents) continue to emphasize such practices as “maximizing system performance while minimizing system acquisition costs.” Such practices generally lead to brittle, point-solution architectures that overconstrain evolution options and inflate evolution costs; and to the lack of key system deliverables for reducing operations and support costs, such as maintenance and diagnostic tools and documentation, test case inputs and outputs, and latest-release COTS components (COTS vendors generally only support their latest three releases. In one maintenance study, we encountered a system which was delivered with 120 COTS products, 66 of which were on releases that were no longer supported by the vendor).

Several good practices for avoiding such situations can be applied in the initial ICSM Exploration phase. These include early addressal of post-deployment and aftermarket considerations such as development of a full Operations Concept Description, including:

- Identification and involvement of key operations and maintenance stakeholders;
- Agreements on their roles and responsibilities;
- Inclusion of total ownership costs in business case analyses;
- Addressal of post-deployment supply chain management alternatives;
- Identification of development practices and deliverables needed for successful operations and maintenance

Since operations and maintenance costs can consume 60 to 90% of an enterprise’s resources, it is also important to build up a knowledge base on their nature, and to apply the knowledge to reduce their costs and difficulties. For example, this was done for the two TRW projects summarized in Figure 3-2. As indicated in Figure 3-2, their major sources of rework effort were found to be off-nominal architecture-breakers. This source of risk was added to the TRW risk management review guidelines for future projects. Also, their additional major sources of life-cycle change were determined to be hardware-software interfaces, new algorithms, subcontractor interfaces, user interfaces, external application interfaces, COTS upgrades, database restructuring, and diagnostic aids, as shown in Table 3-1.

### Table 3-1. Projects A and B Cost-to-Fix Data

<table>
<thead>
<tr>
<th>Category</th>
<th>Project A</th>
<th>Project B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra long messages</td>
<td>3404+626+443+326+244= 5045</td>
<td></td>
</tr>
<tr>
<td>Network failover</td>
<td>2050+470+360+160= 3040</td>
<td></td>
</tr>
<tr>
<td>Hardware-software interface</td>
<td>620+200= 820</td>
<td></td>
</tr>
<tr>
<td>Encryption algorithms</td>
<td>1247+368= 1615</td>
<td></td>
</tr>
<tr>
<td>Subcontractor interface</td>
<td>1100+760+200= 2060</td>
<td></td>
</tr>
<tr>
<td>GUI revision</td>
<td>980+730+420+240+180 =2550</td>
<td></td>
</tr>
<tr>
<td>Data compression algorithm</td>
<td>910</td>
<td></td>
</tr>
<tr>
<td>External applications interface</td>
<td>770+330+200+160= 1460</td>
<td></td>
</tr>
<tr>
<td>COTS upgrades</td>
<td>540+380+190= 1110</td>
<td></td>
</tr>
<tr>
<td>Database restructure</td>
<td>690+480+310+210+170= 1860</td>
<td></td>
</tr>
<tr>
<td>Routing algorithms</td>
<td>494+198= 692</td>
<td></td>
</tr>
<tr>
<td>Diagnostic aids</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td><strong>3620</strong></td>
<td><strong>13531</strong></td>
</tr>
</tbody>
</table>

Following the information-hiding principles in [Parnas 1979], these sources of change were encapsulated in the architectures of similar projects, and additional systems engineering effort was devoted to addressing off-
nominal architecture breakers. Figure 3-4 shows the result of applying this knowledge to the subsequent highly-successful Command Center Processing and Display System-Replacement (CCPDS-R) system described in [Royce 1998]. By investing more effort in systems engineering and architecting, the usual exponential growth in cost to make changes was flattened in both the Implementation and Maintenance phases, as seen in Figure 3-4. The resulting savings in total cost of ownership are shown in Figure 3-5 [Boehm-Lane-Madachy 2010]. It shows that the added investment in CCPDS-R was recouped via rework reduction by the end of the initial Development cycle, and generated increasing savings in later cycles.

Figure 3-4. Traditional TRW vs. Change-Architected Cost of Change Growth

Figure 3-5. TOC’s for Projects A, B, and C (CCPDS-R) Relative to Baseline Costs
[Figure needs fixing to add 5% and 25% initial SysE costs at left]
3.4 Concurrent Engineering of Hardware, Software, and Human Factors Aspects

Not every system has all three hardware, software, and human factors aspects. But when a system has more than one of these aspects, it is important to address them concurrently rather than sequentially. A hardware-first approach will often choose best-of-breed hardware components with incompatible software or user interfaces, or commit to a functional-hierarchy architecture that is incompatible with layered, service-oriented software and human-factors architectures [Maier 2006]. Software-first or approaches can similarly lead to architectural commitments or selection of best-of-breed components that are incompatible with preferred hardware architectures or hard to migrate to new hardware platforms (e.g., multiprocessor hardware components). Human-factors-first approaches can often lead to the use of hardware-software packages that initially work well but are hard to scale to extensive use.

Other problems may arise from assumptions by performers in each of the three disciplines that their characteristics are alike, when in fact they are often very different. For systems having large production runs or limited need or inability to modify the product once fielded (e.g., batteries, satellites), the major sources of life cycle cost in a hardware-intensive system are during development and manufacturing. However, in other cases, hardware maintenance costs dominate, such as the 60-84% of life cycle costs cited for ships, aircraft, and ground vehicles in (Redman et al. 2008) above. For software-intensive systems, manufacturing costs are essentially zero. For information services, the [Koskinen 2010] range of 60-90% of the software life cycle cost going into post-development maintenance and upgrades is generally applicable. For software embedded in hardware systems, the percentages would be similar to the [Redman et al. 2008] ranges above. For human-intensive systems, the major costs are staffing and training, particularly for safety-critical systems requiring continuous 24/7 operations. A primary reason for this difference is indicated in rows 2 and 3 of the table. Particularly for widely-dispersed hardware such as ships, submarines, satellites, and some ground vehicles, making hardware changes across a fleet can be extremely difficult and expensive. As a result, many hardware deficiencies are handled via software or human workarounds that save money overall but shift the life-cycle costs toward the software and human parts of the system.

<table>
<thead>
<tr>
<th>Difference</th>
<th>Hardware</th>
<th>Software</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major lifecycle cost</td>
<td>Domain</td>
<td>Lifecycle</td>
<td>Training, Operations</td>
</tr>
<tr>
<td>Ease of implementation</td>
<td>Generally</td>
<td>Good architecture framework</td>
<td>Very good, but dependent</td>
</tr>
<tr>
<td>Indivisibility</td>
<td>Manual, Intensive</td>
<td>Electronic, inexpensive</td>
<td>Need retraining, can expensive</td>
</tr>
<tr>
<td>User tailorability</td>
<td>Generally limited</td>
<td>Often mission driven</td>
<td>Often mission-driven</td>
</tr>
<tr>
<td>Underlying science</td>
<td>Physics, continuous mathematics</td>
<td>Discrete mathematics, linguistics</td>
<td>Behavioral and cognitive science</td>
</tr>
<tr>
<td>Testing</td>
<td>By test much continuity</td>
<td>By test little continuity</td>
<td>By</td>
</tr>
</tbody>
</table>

Table 3-2. Differences in Hardware, Software and Human System Components

As can be seen when buying hardware such as cars or TVs, there is some choice of options, but they are generally limited. It is much easier to tailor software or human procedures to different classes of people or purposes. It is also much easier to deliver useful subsets of most software and human systems, while delivering a car without braking or steering capabilities is infeasible.

The science underlying most of hardware engineering involves physics, chemistry, and continuous mathematics. This often leads to implicit assumptions about continuity, repeatability, and conservation of
properties (mass, energy, momentum) that may be true for hardware but not true for software or human counterparts. An example is in testing. A hardware test engineer can generally count on covering a parameter space by sampling, under the assumption that the responses will be a continuous function of the input parameters. A software test engineer will have many discrete inputs, for which a successful test run provides no assurance that the neighboring test run will succeed. And for humans, the testing needs to be done by the operators and not test engineers.