Balancing Opportunities and Risks in Component-Based Software Development

Barry Boehm, *University of Southern California*

Jesal Bhuta, *Infosys*

The Incremental Commitment Model provides a process framework for successful opportunistic software development and decision-making. It's based on balancing opportunities and risks in software development projects.

Increasingly rapid IT changes require software development projects to continuously monitor and adapt to new sources of opportunity and risk. Projects that do not capitalize on new opportunities will generally find their products unable to compete. Projects that do not assess and manage their risks will generally find themselves awash in rework, overruns, and product defects.

With respect to expected value of a software-intensive system, opportunity and risk are two sides of the same coin. For any new source of potential loss in system value, the risk exposure RE equals P(Loss) * S(Loss), where P(Loss) is the probability that the new development will decrease the system's value and S(Loss) is the resulting loss. Similarly, for any new source of potential value gain, the opportunity exposure OE equals P(Gain) * S(Gain), where P(Gain) is the probability that the new development will increase the system's value and S(Gain) is the resulting gain.

Potential sources of value gain include improved commercial-off-the-shelf (COTS) products, reusable components from corporate or open source repositories, maturing open standards, strategic partnerships, and Web 2.0 mashups. The value gain might come in the form of faster time to market, reduced life-cycle costs, stronger capabilities, or increased customer satisfaction. On the other hand, losses in system value might emerge from user-interface discontinuities, complications in vendor support or intellectual property rights, increased life-cycle costs from losing evolutionary control, and increased development costs—often accompanied by performance losses—because of architectural mismatch among components.

A good example of the risks of architectural mismatch in component mashups was the Aesop system.[1] It was a mashup of a GUI builder, an object-oriented database management system, and two middleware packages, which was supposed to take 6 months and 1 person-year. Because of mismatches in architectural styles, component interfaces, and assumptions about ownership of control, the mashup ended up taking 2 years and 5 person-years to produce something that was sluggish in performance and hard to modify.

The Incremental Commitment Model (ICM) provides a process framework to improve project monitoring and decision-making based on balancing opportunities and risks.[2,3] We've developed an

0740-7459/08/$25.00 © 2008 IEEE

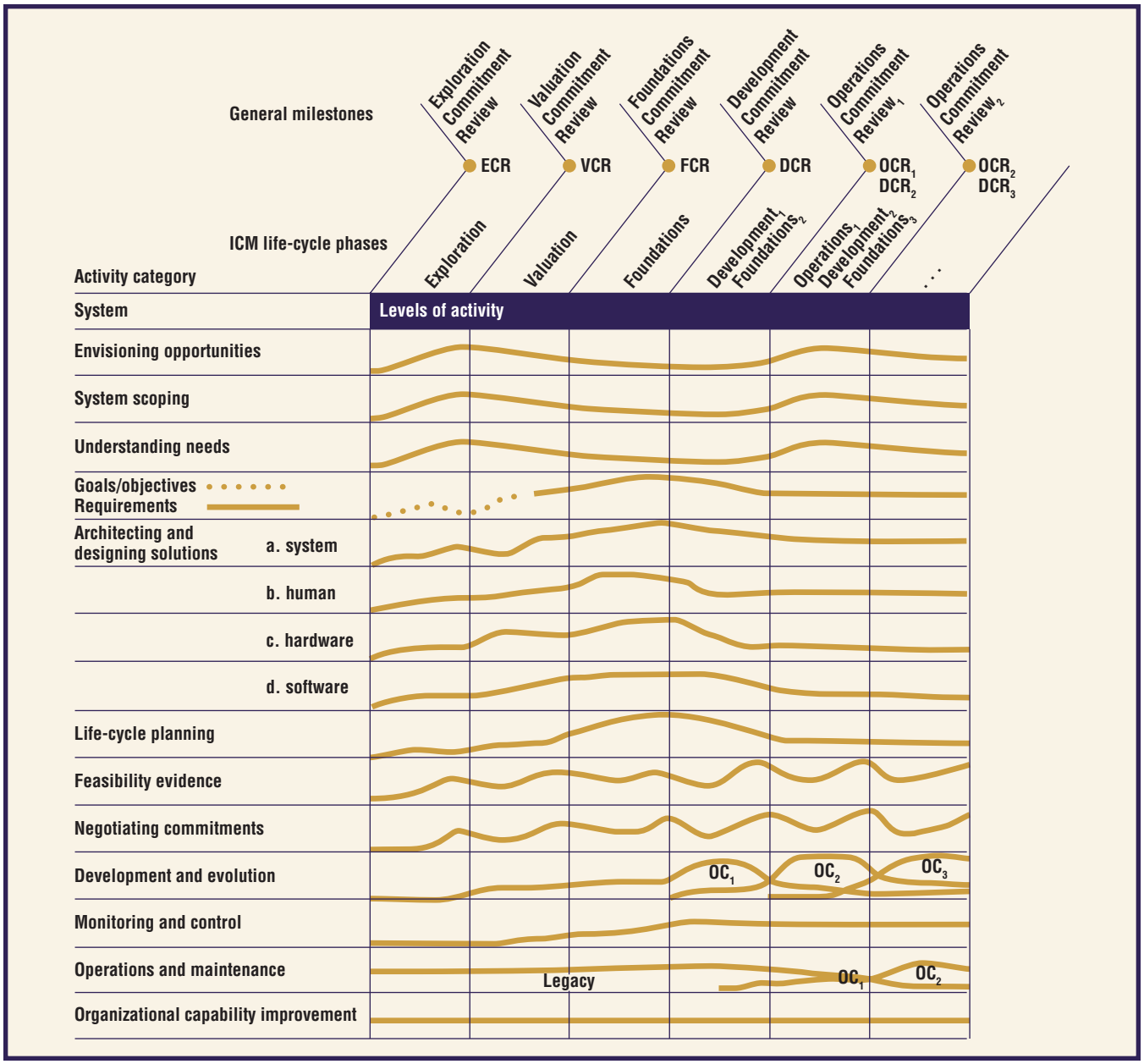


Figure 1. Incremental Commitment Model levels of effort. The levels of effort for success-critical activities vary over a project life cycle. The general milestone reviews synchronize concurrent activities. At these reviews, the system's key stakeholders review feasibility evidence from the preceding phase to determine whether to commit their resources to proceeding with the next phase. The DCR and OCR subscripts indicate that for each increment, the stakeholders review the feasibility of committing to the operation of the currently developed increment and to the development of the next increment. The subscripted operational capability (OC) callouts refer to the levels of effort devoted to the development-and-evolution and operations-and-maintenance activities of each OC increment.

ICM-based framework and automated tool that specifically identify the risks in architectural mismatches in component-based system development.

The ICM Process Framework

To adapt rapidly and successfully to increasing rates of change, projects must be able to assess and manage opportunities and risks; requirements, solutions, plans, and business cases; and hardware, software and human factors concurrently rather than sequentially. Figure 1 illustrates ICM levels of effort in concurrent activities over a project life cycle. The illustration builds on Philippe Kruchten's "hump" diagram for the Rational Unified Process (RUP).[4]

As with RUP, the magnitude and shape of ICM

Pass/Fail Feasibility Evidence Description

Evidence *provided by the developer* and *validated by independent experts* that if the system is built to the specified architecture, it will

- satisfy the requirements: capability, interfaces, level of service, and evolution;
- support the operational concept;
- be buildable within the plan's budgets and schedules;
- generate a viable return on investment;
- generate satisfactory outcomes for all the success-critical stakeholders.

Shortfalls in evidence are sources of risk:

- All major risks are resolved or covered by risk-management plans.

Risk items and risk-mitigation strategies serve as a basis for the stakeholders' commitment to proceed.

Figure 2. Pass/fail feasibility evidence description. The FED provides a checklist to help stakeholders determine where independent experts are needed to review the criteria for stakeholders' commitment to proceed.

levels of effort are risk-driven and likely to vary from project to project. In particular, they're likely to have many small, short-lived risk- and opportunity-driven peaks and valleys, rather than the smooth curves shown for simplicity in Figure 1. Figure 1 mainly emphasizes the necessary concurrency of the primary success-critical activities listed down the left column. So, in the exploration phase, a primary activity is system scoping. But scoping systems well also involves considerable effort in the activities of envisioning opportunities, understanding needs, identifying and reconciling stakeholder goals and objectives, architecting solutions, life-cycle planning, evaluating alternatives, and negotiating stakeholder commitments.

For example, to explore the initial scoping of a system of systems (SoS) for a metropolitan area's disaster-relief effort, you wouldn't just interview stakeholders and compile a list of their expressed mission needs. You would also explore opportunities for reusing parts of other metropolitan-area relief systems as well as for obtaining development funds from federal agencies and for applying virtual-collaboration technologies. In the activity to understand needs, you would concurrently assess the capability and compatibility of existing local disaster-relief systems to determine which ones would need the least work to reengineer into a SoS. You would assess each existing system's scope of authority and responsibility to determine whether the best approach would be a truly integrated, centrally managed SoS or a best-effort interoperable set of systems. Other concurrent activities might include exploring alternative architectural concepts for developing and evolving the system; developing alternative phased plans to determine which improvements would provide the best early benefits and foundations for future growth; evaluating relative feasibility, benefits, and risks for stakeholders to review; and negotiating resource commitments to proceed into the valuation phase.

Similar concurrency applies to all projects—from large SoSs to small, time-constrained Web applications. Every year, we use the ICM to teach software engineering by having over a dozen student teams define, design, develop, and deploy Web applications for real clients in a fixed 24-week time period with a 92 percent success rate.

Synchronizing Concurrent Engineering

The anchor-point milestone review at the end of each life-cycle phase is the ICM mechanism for synchronizing, stabilizing, and assessing risk.[5,6] These reviews, labeled across the top of Figure 1, focus on developer-produced evidence along with PowerPoint charts and Unified Modeling Language (UML) diagrams. The evidence helps key stakeholders determine the next level of commitment.

Consider a system requirement to complete a real-time, safety-critical task in one second. Developers in this case must provide evidence based on prototyping, benchmarking, modeling, or simulation using representative workloads. The evidence must show that the as-designed system will meet the task-completion time requirement. This differs from a promise to "build it now and tune it later," which is frequently the practice in ad hoc or agile development. If the requirement were for a 1-second desirable, 3-seconds acceptable user-response time for a noncritical system, agile methods would generally be fine.

The Exploration Commitment Review (ECR) focuses on an exploration-phase plan that includes the proposed scope, schedule, deliverables, and resource commitments by a key subset of stakeholders. The plan content is risk-driven and might be only a single page for a small, uncontroversial exploration for which the risk of getting it wrong is minimal. A riskier exploration phase would require a more detailed plan outlining how the project will evaluate and manage the risk of going forward.

The Valuation Commitment Review (VCR) is similarly risk-driven. The content includes the exploration-phase results and a valuation-phase plan, and the review includes all valuation-phase stakeholders.

Both the Foundations Commitment Review

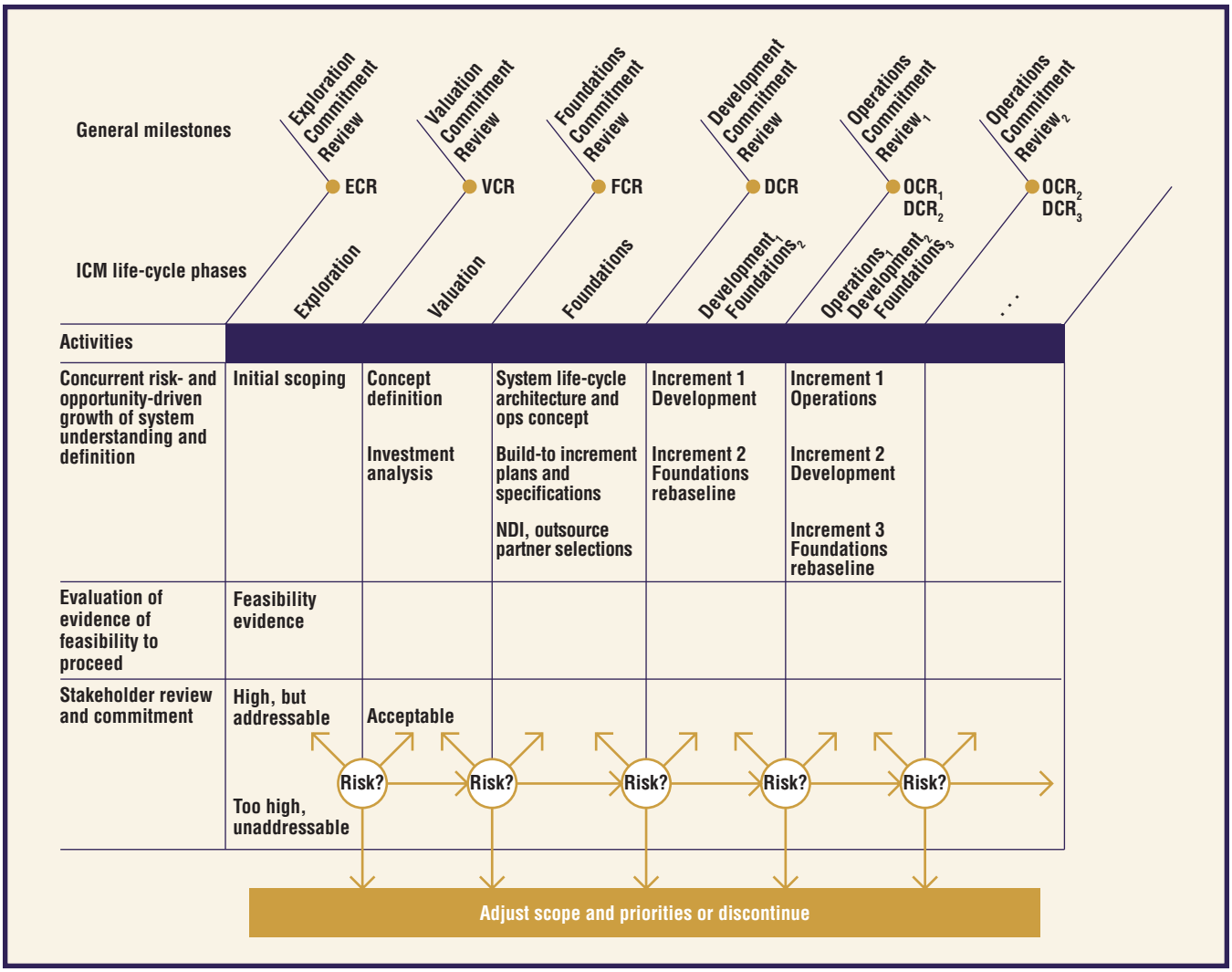


Figure 3. The Incremental Commitment Model life-cycle processes. The ICM identifies the concurrently engineered life-cycle phases, the anchor-point reviews for stakeholder commitments, and the risk-based decision options stakeholders have for proceeding to the next phase, skipping the next phase, extending the current phase, or deciding to either terminate or rescope the project if the risks are too high or unaddressable.

(FCR) and the Development Commitment Review (DCR) are based on the highly successful AT&T Architecture Review Board procedures.[6] The FCR elaborates only high-risk aspects of the operational concept, requirements, architecture, and plans. The FCR requires evidence that at least one combination of those artifacts satisfies the feasibility-evidence criteria shown in Figure 2 (similar to the RUP life-cycle objectives milestone). At the DCR, developers must demonstrate that the particular choice of artifacts to be used for development will satisfy the feasibility-evidence criteria.

Risk-Driven Commitment Milestones

Figure 2 emphasizes that shortfalls in feasibility evidence are sources of risk and so require risk-management plans. Stakeholders must consider the degree of risk in going forward to the next phase as the key decision criterion for committing their resources to it.

Figure 3 presents an overview of the ICM life-cycle process. Like Figure 1, it identifies the concurrently engineered life-cycle phases, the stakeholder commitment review points, and the use of feasibility rationales to assess the compatibility, feasibility, and risk associated with the concurrent-engineering artifacts. Each review point offers several alternatives:

- The risks are negligible, and no further analysis or evaluation activities are needed to proceed to the next life-cycle phase.
- The risk is acceptable, and work can proceed to the next phase.

The risk-mitigation strategies show different opportunistic solutions resulting from different stakeholder value propositions.

- The risk is addressable but requires backtracking.
- The risk is too great and requires rescoping or halting the development process.

The system's success-critical stakeholders assess these risks. They base their commitment on whether the current system definition gives sufficient evidence that the system will satisfy their value propositions. Many risk-driven paths exist through the life cycle. A more risk-seeking set of stakeholders will tend to go forward or skip phases at a decision point. For the same risk level, more risk-averse stakeholders might choose to extend the previous phase, rescope, or discontinue the project.

Opportunity and Risk Management Alternatives: A COTS Example

Suppose that at a project's beginning, an opportunity exists to choose either a higher-performance COTS product B or a comparable but lower-performance COTS product C. A pure opportunistic approach would choose B and have C as a fallback. However, during integration, the project finds out that B has serious architectural mismatches with another project-essential COTS product A. This will cause project overruns of 3 months and US$300K.

During integration, the probability of the mismatch is 1.0, so it's not a risk but a problem. But earlier, its probability was uncertain, and an early ICM milestone review would have identified it as a source of risk. In accordance with the FED (see Figure 2), it would require a risk-management plan, which could use one or more of the main risk-mitigation strategies—namely, buying information, risk avoidance, risk transfer, risk reduction, or risk acceptance. In general, buying information is the best strategy to try first because it provides more insight into which other strategies to employ. We illustrate these five strategies with our COTS example. (Other COTS-specific process guidelines are less specific about how COTS-integration risks drive development decisions. The ICM provides ways for strengthening their application.[7–10])

Buying information. The project decides to spend $30K prototyping the integration of COTS packages B and C with COTS package A. It finds the architectural mismatches between A and B as well as the likely costs and schedules to resolve them. It also finds that COTS package C would integrate easily with A, but with a 10 percent performance loss. This information enables the stakeholders to better evaluate the other risk mitigation strategies.

Risk avoidance. The customer agrees that performance reduction is preferable to the prospect of late delivery. So, the project proceeds with COTS product C rather than B.

Risk transfer. The customer decides that the performance increase is worth the extra time and money and establishes a risk reserve of 3 months and $300K. These are used if they're needed during integration, but the developer receives award fees to the extent that fewer resources are needed.

Risk reduction. The developer and customer agree to integrate A and B in parallel early in the project, with added cost but with no delay in delivery schedule.

Risk acceptance. The developer decides that having a proprietary solution to integrating A and B will provide a competitive edge on future projects, and decides to fund and patent the solution.

Assessing Software Integration Risks

The risk-mitigation strategies show different opportunistic solutions resulting from different stakeholder value propositions. The following assessment framework and corresponding tool, Integration Studio (iStudio),[11] offer a strategy to balance risks and opportunities for the example we highlighted earlier.

The framework is modeled using three components: a COTS-interoperability analyzer, COTS-representation attributes (to define components), and integration rules. Framework inputs are various COTS component definitions and the high-level system architecture. Output is an interoperability assessment report, which includes results of three major analyses:

- Internal-assumption mismatch analysis, which identifies mistaken assumptions that interacting COTS systems make about each other's internal structure.[12]
- Interface (or packaging) mismatch analysis, which identifies incompatible communication interfaces between two components.
- Dependency analysis, which identifies facilities that the COTS packages used in the system require (for example, Java-based customer-relationship-management solutions require Java Runtime Engine).

The framework identifies 42 attributes, which appear in Figure 4. These attributes define as-

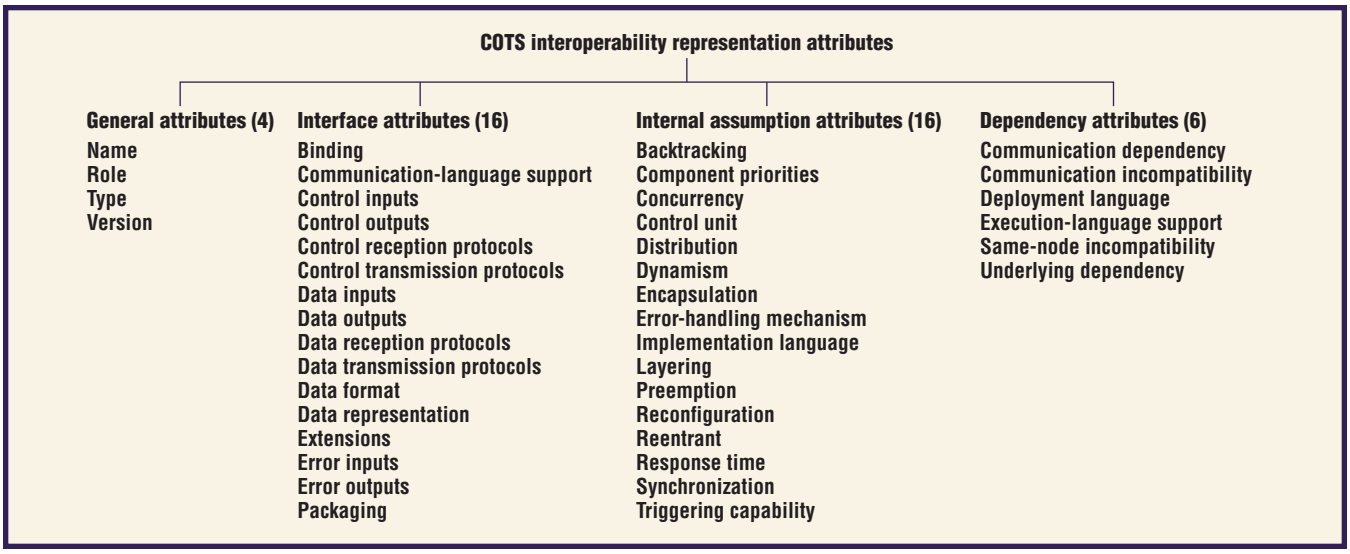


Figure 4. COTS interoperability representation attributes. A framework for assessing component interoperability identifies 42 attributes that define assumptions, interfaces, and dependencies about components.

sumptions about components, along with their interfaces and dependencies. In addition, the framework defines 62 interoperability-analysis rules to identify architectural mismatches for a given architecture. Of these 62, 50 rules address internal-assumption mismatches, seven rules identify interface mismatches, and five rules detect dependency mismatches.

For example, an interface-mismatch rule defines a data-interaction mismatch that can occur when two interacting components exchanging data don't support any common data-exchange connectors or protocols. When the framework (or the framework-supported tool) identifies a mismatch, it recommends possible strategies for use in resolving it. (Details of interoperability-assessment attributes and rules are available elsewhere.[11])

The iStudio tool developed on the basis of this framework inputs high-level, component-based deployment architectures together with component-interoperability characteristics. The component characteristics are based on interoperability analyses and the aforementioned attributes (in XML). iStudio specifies every interaction in the architecture by

- interaction type (control and/or data),
- interaction direction (unidirectional, bidirectional), and
- interaction initiator (the component initiating the interaction).

Using the deployment architectures, iStudio determines the system deployment topology as well as the distribution of COTS products. It uses the 62 interoperability analysis rules for performing architecture-mismatch analysis.

Upon completing its analysis, the tool outputs a report that includes the three assessment-results groups: internal assumptions, interfaces, and dependencies. Developers can use this report to identify the amount of effort required to integrate the COTS products. The tool also packages a Web site that lets analysts create XML-based COTS product definitions, then store and reuse them across multiple architectural assessments.

COTS Assessment and Selection Scenario

A project at NASA's Jet Propulsion Laboratory (JPL) offers a COTS assessment and selection problem derived from several existing challenges. This assessment scenario helps illustrate our framework's utility and ground it in an existing real-world problem.

Four JPL planetary scientists in Pasadena, California, are responsible for managing several gigabytes of planetary-science data that includes digital content and the corresponding metadata. The JPL scientists need to share their data with colleagues at the European Space Agency (ESA) in Madrid, Spain, who also manage several gigabytes of planetary data. In addition, thousands of external users, including other planetary scientists and educators (each with their own preferences), are customers of the data available through JPL and ESA's independent planetary-data systems.

To support the planetary scientists' needs, JPL and ESA commissioned a team of software architects and engineers to design and implement a

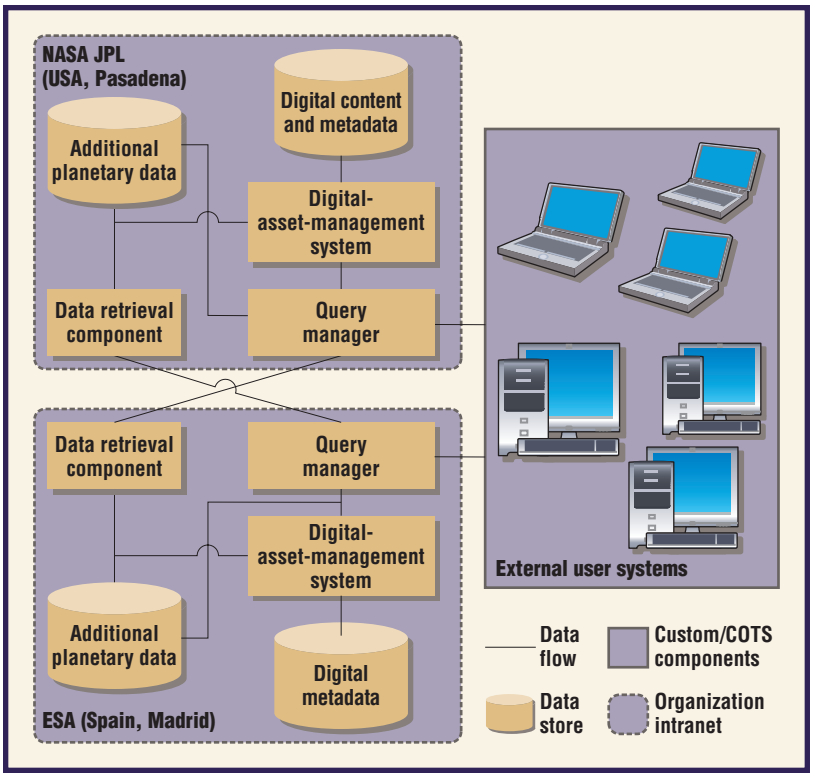


Figure 5. A potential architecture for a large-scale data-distribution scenario. The architecture supports planetary scientists' needs at NASA's Jet Propulsion Laboratory and the European Space Agency in Madrid as well as thousands of external users.

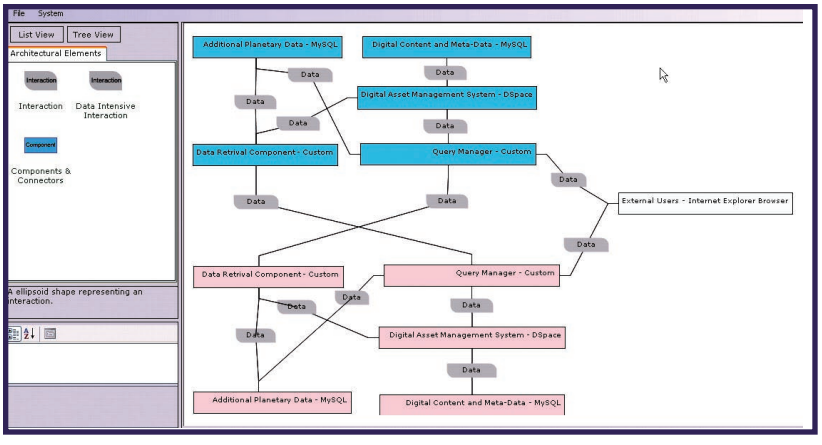


Figure 6. Integration Studio tool interface snapshot. The iStudio tool supports integration analysis for COTS product interactions.

software system that can support the data-distribution tasks outlined between JPL and ESA. The system must also serve the external users.

Figure 5 displays a potential architecture for such a system. The systems based at JPL and ESA use a COTS digital-asset-management system such as DSpace, data stores that include at least one type of database system such as Oracle or Sybase, and two custom components—one of which manages user queries while the other retrieves data from its counterpart system at periodic intervals.

Solving the data-sharing challenge boils down to answering the question of how to select the appropriate COTS components for supporting data distribution, given so many heterogeneous systems involved. Several considerations might guide the COTS technologies to deploy in such a scenario—for example, an organization's requirements, the skill levels of programmers tasked with implementing the system, or even the system's architecture. It's important, however, to assess and determine COTS-integration risks before developing, acquiring, and deploying the components to avoid rework caused by architecture mismatches.

Our example has two major considerations in assessing COTS products and implementation technologies to identify interoperability conflicts:

- interoperability conflicts when integrating the digital-asset-management system with the database, and
- language selections for developing the custom components to minimize the development effort by leveraging existing support that the COTS products provide.

To address these two considerations, the analyst will use the iStudio tool and provide the following information for every interaction in the proposed architecture:

- data and/or control interaction,
- unidirectional or bidirectional interaction, and
- which component initiates the interaction.

Assume an assessment scenario where DSpace is the digital-asset-management system and MySQL is the database server. The analyst will provide the definitions for DSpace and MySQL and define the deployment architecture, along with various component interactions, using the iStudio tool for assessment. (Figure 6 shows a snapshot of the tool interface.)

Our framework's integration-analysis component will use COTS attributes and the deployment architecture definition and apply the rules from the integration-rules repository to identify

- common interfaces supported by MySQL and DSpace, bridging connectors and the required glue-code type (communication, conversion, coordination, or a combination thereof);[11] for example, DSpace and MySQL lack a common data-communication interface, so this will require glue code.

- internal-assumption mismatches between MySQL and DSpace;[11] for example, if either MySQL or DSpace is inactive while the other is transmitting information, the system will lose this data. Another example occurs when data being transferred from MySQL is later backtracked, causing system inconsistencies.
- COTS dependencies and verify that they've been satisfied in the given architecture.
- recommended languages for the query manager and data-retrieval component to simplify glue-code development between COTS and custom components. For example, the tool will recommend that custom components, such as query-manager and data-retrieval components, be built using Java because DSpace supports it.

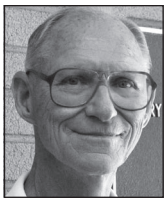
In this example, where the two interacting components (DSpace and MySQL) lack common interfaces, the framework and iStudio tool will recommend a connector requirement that can enable communication between the two components (in this case, a JDBC-MySQL driver). Our framework-iStudio tool will output these findings in a report for the project analyst. The project analyst can use these findings to determine the software integration risks if the project selects the given components for implementation.

Software projects and organizations can increase their success rates in software development by better assessing and balancing their opportunities and risks. By requiring early evidence that opportunistic approaches to software development are feasible (where possible via tools, such as iStudio, and otherwise via COTS integration experiments or prototypes), the ICM can help projects determine whether attractive options such as Web 2.0 mashups are likely to lead to satisfying success or frustrating failure.

References

1. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse Is So Hard," *IEEE Software*, vol. 12, no. 6, 1995, pp. 17–26.
2. R.W. Pew and A.S. Mavor, *Human-System Integration in the System Development Process: A New Look*, Nat'l Academy Press, 2007.
3. B. Boehm and J. Lane, "Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering," *CrossTalk*, Oct. 2007, pp. 4–9.
4. P. Kruchten, *The Rational Unified Process*, Addison-Wesley, 1999.
5. B. Boehm, "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, 1996, pp. 73–82.
6. M. Maranzano et al., "Architecture Reviews: Practice and Experience," *IEEE Software*, vol. 22, no. 2, 2005, pp. 34–43.
7. C. Albert and L. Brownsword, *Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview*, tech. report CMU/SEI2002-TR-009, Carnegie-Mellon Univ., July 2002.
8. B.C. Meyers and P. Oberndorf, *Managing Software Acquisition: Open Systems and COTS Products*, Addison-Wesley, 2002.
9. K. Wallnau, S. Hissam, and R. Seacord, *Building Systems from Commercial Components*, Addison-Wesley, 2002.
10. Y. Yang et al., "Value-Based Processes for COTS-Based Applications," *IEEE Software*, vol. 22, no. 4, 2005, pp. 54–62.
11. J. Bhuta, "A Framework for Intelligent Assessment and Resolution of Commercial-Off-The-Shelf Product Incompatibilities," doctoral dissertation, Computer Science Dept., Univ. of Southern California, 2007.
12. C. Gacek, "Detecting Architectural Mismatches during Systems Composition," doctoral dissertation, Computer Science Dept., Univ. of Southern California, 1998.

About the Authors



Barry Boehm is the TRW professor of software engineering in the University of Southern California's Computer Science Department and the director of its Center for Systems and Software Engineering. His research interests involve recasting systems and software engineering into a value-based framework. Boehm received his PhD in mathematics from UCLA. He's a fellow of the IEEE, ACM, AIAA, and International Council on Systems Engineering, and a member of the US National Academy of Engineering. Contact him at boehm@usc.edu.

Jesal Bhuta is a senior associate with Infosys Consulting. His research interests include commercial-off-the-shelf based software development and risk management. Bhuta received his PhD in computer science from University of Southern California. Contact him at jesal_bhuta@infosys.com.