Systems engineering best practices
White paper
December 2009

Rational software

Systems engineering—the foundation for success in complex systems development.

Hans-Peter Hoffmann, chief systems methodologist, Rational software, IBM Software Group

Chris Sibbald, product manager, Rational software, IBM Software Group

Jonathon Chard, systems marketing manager, Rational software, IBM Software Group

Contents

Executive summary

Delivering complex systems requires that you develop optimal designs on time, within budget and with the right level of quality. But even the best detailed design cannot compensate for poor system architecture. Systems engineering isn't just a technical activity in the product lifecycle—it determines the commercial viability of the entire project. It's also a demanding activity that takes place when there are many degrees of freedom. How do you ensure that you have assessed all the options—and picked the best? This white paper looks at a number of key techniques to optimize the systems engineering process and help ensure project success. It examines the business benefits the techniques can provide and looks at IBM Rational® solutions that are available to support the techniques.

The shortcomings of traditional systems engineering approaches

Systems engineering—which is about clearly specifying functionality, where that functionality resides in the system, and therefore what interfaces are required—is by definition a complex activity that involves an array of players and a multitude of considerations. As if the pressure of efficiently and accurately developing systems in the face of tight budgets isn't enough, competitive considerations are forcing companies to develop increasingly sophisticated systems faster.

Customers expect that the products they purchase will provide innovative functionality that can adapt to specific needs and interact with users and other products and systems in new ways. In these new, smarter products and systems, software is intricately combined with microelectronics, sensors and mechanical technologies, and often other systems. The behavior of these intelligent, instrumented and interconnected systems not only results from the interactions of many components, but some of the components are outside the control of any one development organization as well. For example, consider a stolen vehicle tracking system where a unit on the car might interact with a global positioning system, a mobile phone system, and the vehicle security and power-train systems.

Highlights

In this type of richly functional system, multiple development organizations are involved, and bringing together the organizations and functional components to create the system of systems presents enormous technical challenges.

In most organizations, multiple siloed teams contribute to the systems engineering process, often using inconsistent processes and separate tools. Product quality is often compromised by the resulting integration problems. Project risk is high as a result of this situation and only increases further as other organizations or a consortium gets involved.

Multiple, siloed teams contributing to systems engineering processes can lead to product integration challenges and higher project risks.

Document-centric approaches to systems engineering frequently add to the challenges. They typically use requirements documents and interface control documents which, when suitably structured, can capture traceability and formalize interface specifications. On their own, however, they can be an inefficient way of analyzing and understanding system architecture and behavior: while text is the preferred medium of exchange for contractual purposes, it is not well suited to describing complex functional concepts and fails to provide abstractions to support stakeholder and developer understanding of the system under development.

Highlights

The ultimate goal with any systems engineering project is to create systems that are predictable, competitive, profitable and compliant.

Optimizing systems engineering: a look at key approaches

To be sure, systems can have widely different scales and needs for critical availability, performance and timing capabilities whether they are embedded systems, such as pacemakers; complex systems, such as aircraft; or distributed systems, such as telecommunications networks. In the end, however, most organizations have similar goals when it comes to systems engineering, which include creating systems that are:

- ***Predictable****—perform consistently to specifications and do not confront stakeholders with surprises.*
- ***Competitive****—deliver the right functionality and performance for the right price.*
- ***Profitable****—deliver optimum return on investment.*
- ***Compliant****—comply with relevant industry or government regulations.*

Of course, every organization or team will also have its own specific business goals and objectives, and the overall challenge is to select the appropriate best practices to achieve these goals, implement processes and tools, assess progress and implement changes, and stay on track. What follows are recommendations that IBM believes are key to getting the systems engineering process right and underpinning success.

Model-based systems engineering has emerged as one of the best ways to increase productivity and quality.

Model-driven paradigm

Model-based systems engineering has emerged as one of the best ways to increase productivity and quality. Models can capture both the structure of the system (architecture) and behavior, and model-based systems engineering helps address complexity by raising the level of abstraction, enabling teams to view systems models from many perspectives and different levels of detail while ensuring that the system is consistent.

Highlights

Using SysML for modeling helps improve quality by reducing ambiguity and supporting reviews of the dynamic behavior of systems as well as automation capabilities.

The Systems Modeling Language (SysML), which is a dialect of the Unified Modeling Language (UML), is becoming an accepted standard for modeling in the systems engineering domain. The formal nature of languages such as SysML and UML helps improve quality by reducing ambiguity. In fact, models can now show the dynamic behavior of systems, including how they transition between modes and how the system behaves overall. Formal languages can also support automation capabilities, including the execution of models to verify system behavior. Moreover, rather than requiring a separate activity, design documentation can now be produced as an automated output of modeling activities. Combining these activities helps reduce errors and inconsistencies associated with aggregating and transcribing information while also saving time.

Using a formalized modeling language makes it possible for tools to execute the models—so you can make pictures "live."

Model execution

Historically, the primary use of modeling in systems engineering has been to capture designs graphically, and it stopped there. The use of a formalized modeling language, however, opens up the possibility for tools to support the execution of models. In other words, it's the difference between just drawing pictures and making pictures live. By automating this process, requirements can be verified as complete and correct, helping to improve the quality of the delivered system.

Use a best practices workflow

Adapting the model-driven paradigm to systems engineering creates the opportunity to streamline and improve project workflows. Ultimately, the key objectives of a model-based systems engineering approach should include:

- *Identifying and deriving the required system functions.*
- *Identifying the associated system modes and states.*
- *Designing a subsystem structure with responsibilities that satisfy the system functions as well as modes and states.*

Highlights

IBM has identified several best practices that are key to effective model-based systems engineering.

These objectives imply a top-down approach and a high level of abstraction. The focus is on identifying and allocating needed functionality and state-based behavior rather than on the details of functional behavior. IBM has identified several best practices that are key to effective model-based systems engineering. With the help of the best practices, you can define preconditions and postconditions and reuse them, mapping to individual workflows.

- ***Requirements analysis****—Translate stakeholder requirements into systems requirements and establish traceability links. Use cases are then employed as a means of grouping functional requirements into functionally related sets that will be used to prioritize the order of analysis and design synthesis iterations. Traceability links are then established from the use cases back to functional requirements.*
- ***System functional analysis****—Translate each use case into a model that describes the requirements graphically. Model execution is then used to verify the completeness and correctness of the requirements. This may result in system requirements being modified and new system requirements being derived. Traceability links are then created between the verified model artifacts and the system requirements.*

Highlights

Design synthesis takes place in three phases: architectural analysis, architectural design and detailed architectural design.

Software analysis and design also benefit from a model-based systems engineering approach by supporting a clear handoff between the two activities

- ***Design synthesis****—At this point, the functional requirements are known to be complete and correct, so the team can consider how to best realize those requirements. Design synthesis takes place in three phases:*
  - *Architectural analysis, which is achieved through trade studies, also taking into account customer constraints and nonfunctional requirements.*
  - *Architectural design, which is where functional and nonfunctional requirements are realized by the architecture. Because traceability exists from stakeholder requirements through architectural model artifacts, full impact analysis can be performed when a requirement is changed. The process handles both functional and nonfunctional requirements; therefore, traceability is maintained for both requirement types.*
  - *Detailed architectural design, which is where the ports and interfaces and state-based behaviors are specified for system blocks at the lowest level of architectural decomposition. The completeness and correctness of the system model can then be established by executing the complete architecture.*
- ***Software analysis and design****—While outside the scope of this paper, software analysis and design also benefit from a model-based approach. Because UML is the de facto standard for analysis and design, model-based systems engineering can support a clearly defined (and automated) handoff between the two activities, thus reducing errors and improving efficiency. The key artifact of the handoff from systems engineering to the subsequent system development is the* baselined *executable model, which is the repository from which specification documents such as hardware and software requirements specifications and interface control documents are generated. The scope and content of the handoff is dependent on the characteristics of both the project and the organization.*

Highlights

Collaboration

Given the scope of systems that companies are creating today, collaboration has become increasingly challenging. Teams are often widely dispersed across cities and countries and even companies in vertical development silos. As a result, coordinating workflows and providing tool support are critical to project success.

For dispersed teams to collaborate effectively, it is important to standardize and document both the usage of the tooling infrastructure and the use of languages and notations.

Simply adopting tools to support the SysML language is not enough; to collaborate effectively, you need to standardize and document both the usage of the tooling infrastructure and the use of languages and notations. Systems engineering is implicitly cross-discipline, and therefore any use of language must be kept domain independent. Furthermore, systems engineering is a complex activity, so minimizing the number of language constructs used in the workflow is essential to limit that complexity. Adopting standards and guidelines for the use of languages and the construction of models can therefore help improve collaboration and efficiency.

Use tools to support the approach

To manage complex systems engineering among multiple teams, it is also important to choose the right tools for requirements management and traceability, system and software development, change and configuration management, and documentation.

The use of formalized modeling language and the execution of models is only part of the equation for effective systems design: Tools for requirements management and traceability, system and software development, change and configuration management, and documentation are also essential to managing complex systems engineering across multiple teams. The right mix of tools can deliver capabilities and benefits beyond simply enabling you to execute.

- ***Efficiency and accuracy in constructing models****—Tools can be dedicated to SysML/UML notation and help ensure quality by providing facilities for the static checking of models, for example.*
- ***A single point of reference repository of design information****—Providing a way for distributed teams to work from a "single version of the truth" can help improve collaboration and reduce versioning/configuration issues.*

Highlights

- ***Traceability***—*Tools can help ensure that all requirements have been considered (coverage analysis) and simplify change management (impact analysis) by supporting the linkage of models to requirements.*
- ***An automated documentation process***—*Through automation, the right tools can help improve efficiency, accuracy and timeliness of project documentation for contractual obligations, compliance, design reviews and project management.*

Measure and improve decision making

The model-driven approach outlined in this paper has opened up the possibility of improved measurement throughout a project, helping to support better decision making.

Although it is more of an evolving capability and benefit, the model-driven approach outlined in this paper has opened up the possibility of improved measurement throughout a project. Although the metrics and methodology are still being defined, the model-driven paradigm is poised to reduce development time and improve efficiency and quality by helping teams identify and address questions or issues at the right level—before proceeding to the next step. As a result, organizations that adopt the model-driven paradigm will continue to see new advantages down the road.

Highlights

The IBM model-driven paradigm for systems engineering enables teams to verify the completeness and correctness of models through model execution.

The advantages of a model-driven paradigm

Model-driven systems engineering with model execution can potentially lead to significant project improvements. Until now, modeling in systems engineering has meant capturing static graphical models of systems—effectively just drawing pictures. Through model execution, the IBM model-driven paradigm for systems engineering enables teams to verify the completeness and correctness of both systems requirements and the chosen architecture in which each change request represents a unit of capability that is added to the system. With the right tools, automated linkage of each modified or new change request to the model artifacts can help simplify the development and subsequent maintenance of the system. Furthermore, the linkage of modeling and requirements facilitates traceability from stakeholder needs to detailed design—a capability that is essential to ensuring that development is aligned to stakeholder needs. It also facilitates effective change management processes based on impact analysis.

An executable, model-driven paradigm changes the definition of test and verification in the systems development cycle—moving it from the right-hand side to the left-hand side of the V development cycle. The shift helps improve risk management in projects by moving the testing to an earlier phase of the project. Moreover, the verification tests performed during systems functional analysis and architectural design synthesis can lead to a paradigm shift in testing because model-based tests can form the basis of testing from the bottom-up system integration phases, including, for example, component verification, integration and system acceptance tests. Potential benefits include improved test planning along with reduced testing time.

The model-based paradigm helps substantially reduce ambiguity in models, helps turn the "test early, test often" adage into reality and provides a framework for comparing different approaches.

Overall, the use of formal notation in the model-driven paradigm helps substantially reduce ambiguity and misunderstandings in models. It also supports model execution, helping organizations turn the "test early, test often" adage into reality. Finally, it provides a framework for comparing different approaches and performing trade studies to identify which approaches are optimal.

Highlights

IBM provides a comprehensive, model-based systems engineering solution consisting of process, tools and services.

Because no single process or toolset will satisfy the needs of all project teams, IBM systems engineering practices include built-in guidance on how to tailor and deploy the process and tools and continually improve the process.

IBM Rational solutions for best practice systems engineering

IBM provides a comprehensive, model-based systems engineering solution consisting of process, tools and services built around the IBM systems engineering practices. IBM systems engineering practices include descriptions of the tasks, work products and workflows needed to effectively define and verify system requirements and architecture and to perform trade studies. Integrated tool support, in the form of tool utilities, templates and tool guidance for IBM Rational tools, automates various aspects of the process to help drive efficiency. IBM systems engineering practices include out-of-the-box support for IBM Rational DOORS®, IBM Rational Rhapsody®, IBM Rational Change and IBM Rational Synergy software.

Because no single process or toolset will satisfy the needs of all project teams, IBM systems engineering practices also include built-in guidance on how to tailor and deploy the process and tools and continually improve the process. Tailoring and deployment is managed using IBM Rational Method Composer software—a tool based on the Object Management Group (OMG) Software Process Engineering Meta-Model (SPEM) version 2.0 specification—for defining, tailoring, managing and communicating the process and best practices. Rational Method Composer software provides simple, form-based editors for changing, adding or deleting tasks, roles and work products and tailoring workflows to address the needs of the team. The resulting tailored process can be easily published to a Web site and exposed directly within development tools to provide practitioners with the context-sensitive process, tool, workflow, role and work product guidance they need—when and where they need it.

For more information

To learn more about model-based systems engineering and IBM systems engineering practices, contact your IBM sales representative or IBM Business Partner, or visit:

ibm.com/software/rational/offerings/ppm/process.html

RAW14193-USEN-00