SEI Insights

SEI Blog

The Latest Research in Software Engineering and Cybersecurity

The Method Framework for Engineering System Architectures

POSTED ON AUGUST 20, 2012 BY **DONALD FIRESMITH** [/AUTHOR/DONALD-FIRESMITH] IN **ACQUISITION** [HTTPS://INSIGHTS.SEI.CMU.EDU/SEI_BLOG/ACQUISITION/]

Engineering the architecture for a large and complex system is a hard, lengthy, and complex undertaking. System architects must perform many tasks and use many techniques if they are to create a sufficient set of architectural models and related documents that are complete, consistent, correct, unambiguous, verifiable, and both usable by and useful to the architecture's many stakeholders. This blog posting, the first in a two-part series, presents the **Method Framework for Engineering System Architectures (MFESA)** [http://www.sei.cmu.edu/library/abstracts/presentations/mfesatutorialoneday20081210.cfm], which is a **situational process engineering** [http://en.wikipedia.org/wiki/Method_engineering#Situational_method_engineering] framework for developing system-specific methods to engineer system architectures. This posting provides a brief historical description of situational method engineering, explains why no single system architectural engineering method is adequate, and introduces MFESA by providing a top-level overview of its components, describing its applicability, and explaining how it simultaneously provides the benefits of standardization and flexibility.

Two Approaches to System/Software Development Methods

Just as **software engineering** [http://en.wikipedia.org/wiki/Software_engineering] is the engineering discipline in which software engineers create software applications by integrating software components, **method engineering** [http://en.wikipedia.org/wiki/Method_engineering] is the discipline in which process engineers (which I'll refer to as "methodologists" below) create system/software development methods by integrating method components. Process engineers roughly fall into one of the following two camps:

1. A methodologist performs *method creation* to construct a reusable system/software development method and then process engineers use ***method tailoring*** to tailor this method for reuse on multiple projects.
2. A methodologist performs ***situational method engineering*** to create a system/software process framework and then process engineers use the method framework to generate multiple project-specific methods.

Process engineering has traditionally been performed by methodologists who created what they hoped would become de facto standard development methods. The 1960s and 1970s saw the development of structured methods, such as **Structured Programming** [http://en.wikipedia.org/wiki/Structured_programming], **Structured Analysis** [http://en.wikipedia.org/wiki/Structured_Analysis], and **Structured Design** [http://en.wikipedia.org/wiki/Structured_Design]. During the early 1980s, **Information Engineering** [http://en.wikipedia.org/wiki/Information_engineering] grew out of database design.

From the mid-1980s to the early 1990s, a large number of similar **object-oriented analysis and design (OOAD)** [http://en.wikipedia.org/wiki/Object-oriented_analysis_and_design] methods (such as **OMT** [http://en.wikipedia.org/wiki/Object-modeling_technique] and the **Booch method** [http://en.wikipedia.org/wiki/Booch_method]) were developed. In the late 1990s, members of the **Agile** [http://en.wikipedia.org/wiki/Agile_software_development] Community began to create their own methods, such as **Extreme Programming** [http://en.wikipedia.org/wiki/Extreme_programming] and **SCRUM.** [http://en.wikipedia.org/wiki/Scrum_%28development%29] Finally during the early 2000s, the overabundance of OOD methods led to groups of methodologists collaborating together to consolidate their individual methods into unified methods such as the **Rationale Unified Process (RUP)** [http://en.wikipedia.org/wiki/RUP] and the less well known **Object-oriented Process, Environment and Notation (OPEN) method** [http://www.open.org.au/].

What all these approaches have in common is that they are individual stand-alone development

methods. The intent is to tailor them slightly for use on individual projects to produce individual systems. In other words, they represent the first of the two ways listed above to perform process engineering.

Situational method engineering (SME) [http://en.wikipedia.org/wiki/Situational_method_engineering#Situational_method_engineering] takes a different approach. Methodologists and process engineers who advance and use SME believe that system/software development projects, development organizations, and systems vary so widely that no single method, no matter how tailorable, is optimal for all situations. Instead of taking a de facto standard method and tailoring it, they start with a method framework that includes a repository of reusable method components. After understanding the specific methodological needs and constraints of the specific project, they select the relevant method components, tailor or extend them as needed, and then integrate them to produce a project- and system-specific development method.

The SME approach has received support from **university researchers** [http://cui.unige.ch/db-research/SREP05/Papers/Preface.pdf]. There are, however, considerably fewer system/software development process engineering frameworks than system/software development methods. One is the **OPEN Process Framework** [http://www.opfro.org], which evolved out of the previously-mentioned OPEN unified method. Over time, the **Unified Process (UP)** [http://en.wikipedia.org/wiki/Unified_Process], which evolved from RUP, has also become more of a process framework than a tailorable general-purpose method.

One Size Does Not Fit All

Most systems-development projects rely on the reuse of an existing architecture engineering method to specify how their system architects should develop the system architecture. This method may have been documented in a book or may be part of the development organization's standard system engineering approach. Often, the method is selected because it is the method with which the chief architect(s) are most familiar, not because it is the best fit for the project and system to be architected.

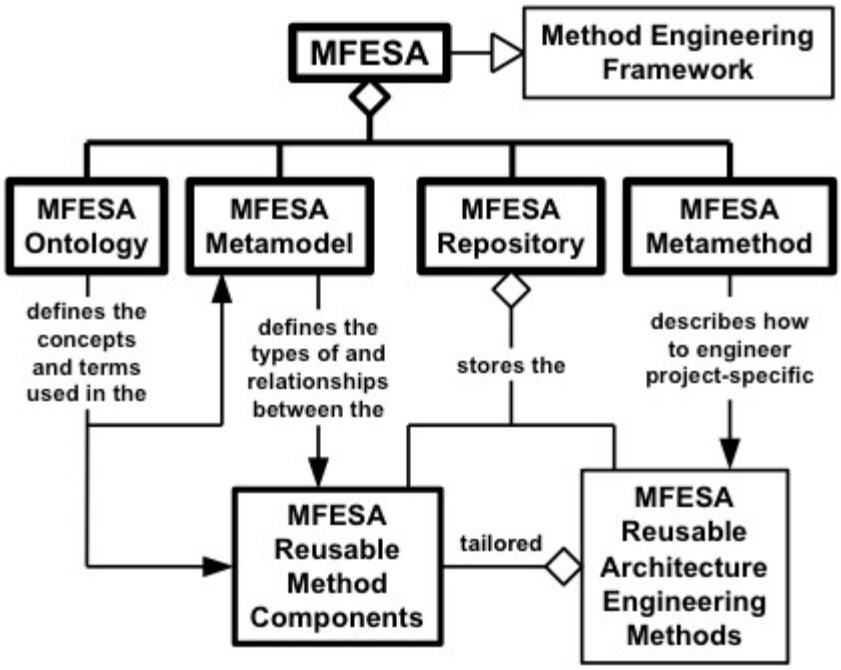
There are risks, however, with the blanket application of an existing standard method to a specific arbitrary system. Systems vary greatly in terms of application domain, autonomy, complexity, criticality, emergent behavior, functionality, geographical distribution, homogeneity, intelligence, operational dependence on other systems, reconfigurability, reuse, size, technology, and uniqueness. Similarly, development organizations vary greatly in terms of degree of centralized control, geographical distribution, management and engineering culture, number, size, staff

experience, and type.

Endeavors also often differ greatly in terms of contracts (formality and type), duration, funding, lifecycle scope, new versus legacy development, schedule, system scope, and type (project versus program of projects v. enterprise). Moreover, stakeholders vary greatly in terms of accessibility, authority, expertise, degree of trust, motivation, needs, number, type, and volatility. No single method is optimal for all combinations of these parameters, and tailoring can only go so far. To provide the most effective method for all these possible situations, a method framework is needed.

The Method Framework for Engineering System Architectures (MFESA)

MFESA is a method framework that enables system architects to create system architecture engineering methods that meet the specific needs of their systems, development organizations, projects, and stakeholders. MFESA is not a method for engineering system architectures. Rather, it is a method framework for creating system architecture engineering methods that are appropriate for engineering the architecture of specific systems.



As shown in the figure above, MFESA consists of the following four components:

- **The MFESA ontology of concepts and terminology** clearly defines the concepts and terms

underlying system architecture engineering. The MFESA ontology is more than a glossary of system architecture engineering concepts and terminology; it is an information model that defines both the concepts used to create system architecture engineering methods and the relationships between these concepts.

- **The MFESA reuse repository** contains a class of highly reusable method components for creating situation-specific system architecture engineering methods. The MFESA repository of reusable method components is to system architecture engineering methods what a Java class library is to a Java program. Just as a software engineer selects the relevant Java classes and integrates them into a functioning Java program, a process engineer selects the relevant method components and integrates them to produce the system architecture engineering method.
- **The MFESA metamodel (model of a model)** defines the types and relationships between reusable method components in the MFESA repository. The MFESA metamodel defines the foundational metaclasses (e.g., work products, work units, and workers) that are instantiated to produce the reusable method components. The metamodel provides the structure of and theoretical foundation for the contents of the MFESA reuse repository.
- **The MFESA metamethod** is used to engineer effective and efficient project-specific system architecture engineering methods. Process engineers apply the MFESA metamethod to select and tailor the relevant method components from the MFESA reuse repository before integrating them to produce the actual system architecture engineering method for use on the project.

Applicability

In creating MFESA, our goal was to give system architects a tool that would help them effectively and efficiently design a consistently high-quality system architecture for a system of systems, a single system, and/or a system's subsystems. MFESA is designed for wide applicable, including (but not limited to)

- the entire system lifecycle (from conception through development including initial small-scale production, full-scale production, utilization, and sustainment to retirement)
- acquired systems, as well as systems designed in-house
- new "**greenfield**" development, as well as evolving legacy systems
- built-from-scratch components, as well as development involving systematic reuse of existing components (e.g., commercial off-the-shelf (COTS), government off-the-shelf (GOTS), open-source and freeware)
- systems-of-systems, including product lines, families, and networks of systems

Standardization and Flexibility Process engineers have two opposing goals. First, they want the benefits of standardization, which improves quality and productivity. Process engineers also want

flexibility so that their system architecture engineering method is appropriate for their specific situation.

MFESA primarily provides standardization by the MFESA reuse repository of method components. All resulting system architecture engineering methods are composed of subsets of the same method components. These components share standard concepts and terminology provided by the MFESA ontology. These method components share common standardized properties derived from the MFESA metamodel. Finally, the appropriate method components are selected, tailored, and integrated via use of the standard MFESA metamethod.

MFESA enhances flexibility via enabling process engineers to only select those method components that are appropriate for their current situations. Process engineers are also free to tailor out unnecessary parts of these components or even extend the repository by adding additional method components if the repository is found to be incomplete. Finally, process engineers can even develop different system architecture engineering methods for different subsystems within a single system or for different systems within a system-of-systems, if that strategy is appropriate and cost-effective.

In addition to standardization and flexibility, another benefit of MFESA is that much hard work of method engineering has already been completed. The reuse repository is highly complete because MFESA is restricted to the creation of system architecture engineering methods. All necessary system architecture engineering method components already exist within the repository. These components are also complete so that tailoring is primarily a case of removing what is unneeded rather than performing a gap analysis and supplying missing parts. An added benefit is that since the MFESA method components were constructed to implement system architecture engineering best practices, system architects who use it are assured of having incorporated engineering best practices into their projects.

The next installment in this series will dive deeper into MFESA, including a closer look at its reusable method components, how to use it to create an appropriate situational-specific system architectural engineering method, and potential future directions.

Additional Resources

MFESA is primarily documented in the book ***The Method Framework for Engineering System Architectures*** [http://www.crcpress.com/product/isbn/9781420085754], published in 2009 by CRC Press. The book was co-authored by a six-member team from the Software Engineering Institute, MITRE, and the US Air Force. The book was recently added to the **Intel Corporation's Recommended**

Reading List [http://noggin.intel.com/rr] .

To see a tutorial on MFESA presented at the 2011 IEEE International Systems Conference (ISC) in Montreal, Quebec, Canada please go to
http://donald.firesmith.net/home/publications/publicationsbyyear/2011/2011-MFESA_ISC.pdf
[http://donald.firesmith.net/home/publications/publicationsbyyear/2011/2011-MFESA_ISC.pdf]

To see a tutorial on MFESA presented at the 21st System and Software Technology Conference (SSTC) in Salt Lake City, Utah, please go to
http://donald.firesmith.net/home/publications/publicationsbyyear/2009/MFESA-SSTC.pdf
[http://donald.firesmith.net/home/publications/publicationsbyyear/2009/MFESA-SSTC.pdf]

About the Author

Donald Firesmith

Contact Donald Firesmith [https://www.sei.cmu.edu/contact.cfm]
Visit the SEI Digital Library for **other publications by Donald**
[http://resources.sei.cmu.edu/library/author.cfm?authorID=4637]
View **other blog posts by Donald Firesmith**
[/author/donald-firesmith]