Failure Analysis: Insights from Model-Based Systems Engineering

William D. Schindel
ICTT System Sciences
schindel@ictt.com

Abstract. Processes for system failure analysis (e.g., FMEA) are structured, well-documented, and supported by tools. Nevertheless, we hear complaints that FMEA work feels (1) too labor intensive to encourage engagement, (2) somewhat arbitrary in identifying issues, (3) overly sensitive to the skills and background of the performing team, and (4) not building enough confidence of fully identifying the risks of system failure. In fairness to experts in the process, perhaps such complaints come from those less experienced—but even so, we should care how to describe this process to encourage better technical and experience outcomes. This paper shows how Model-Based Systems Engineering (MBSE) answers these challenges by deeper and novel integration with requirements and design. Just as MBSE powered the requirements discovery process past its earlier, more subjective performance, so also can MBSE accelerate understanding and performance of failure risk analysis--as a discipline deeply connected within the SE process.

What Would We Like to Improve Upon?

Challenges of Traditional Failure Analysis Processes. Processes for system risk and failure identification, analysis, and planning are well-known, documented, and frequently supported by tools. These include Failure Modes and Effects Analysis—FMEA (Dyadem 2002, 2003; ISO/IEC 2006, 2007; US DoD 1980), Fault Tree Analysis—FTA (Hyatt 2003), Reliability Centered Maintenance Planning—RCM (Moubray 1997), Process Hazards Analysis--PHA (Hyatt 2003), and Hazards and Operability Analysis—HAZOP (Hyatt 2003). Those who perform these sometimes voice challenges of these processes, such as the following:

(1) Frequently labor intensive or tedious, adding cost and sometimes discouraging to the energy of those who face the next session;

(2) May overlook certain problems, or feel somewhat arbitrary in identifying issues;

(3) Typically outcome is very sensitive to the skills and background of the performing team;

(4) May not feel systematic in fully identifying the risks of system failure.

These lead us to ask: How can processes for failure identification and analysis be made to feel more systematic and less arbitrary and exhausting? How do we gain assurance we have found all the important failure modes and effects for a system? These and other challenges of traditional systems engineering approaches are being addressed through the use of Model-Based Systems Engineering (MBSE).

Assumed MBSE Background We'll Need

The Emergence of Model-Based Methods. Model-based methods supplement the use of natural language prose in traditional engineering documents with the use of "models" which are explicit data structures (typically relational tables and formal diagrams). The structure of these models can be exploited to create analyses and checks that would be much more difficult and subjective to perform using purely prose-based methods. When applied well, they can also more effectively convey shared meaning to human readers. There is a growing literature on Model-Based Systems Engineering (MBSE) (Estafan 2009; Hybertson 2009; INCOSE 2009; Schindel 2005a). In this paper, we will focus on how failure analysis can be more deeply integrated as a part of such MBSE models.

Base MBSE Metamodel. The failure analysis approach this paper describes uses the fact that the requirements and high level design of a subject system can be represented in an information structure summarized by the Base SE Metamodel of Figure 1:

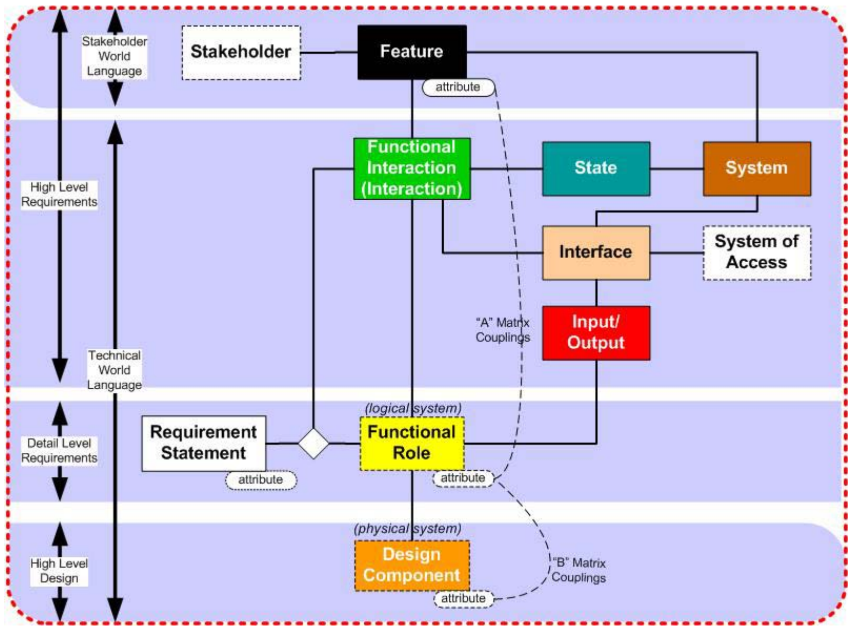

Figure 1: Summary of Base SE Metamodel

Among the impacts of this Metamodel is the re-positioning of prose functional Requirements Statements, which become a formal part of the model, as input-output relationships describing external system "black box" behavior during Interactions with external actors—a kind of "prose transfer function". This is important to the results discussed in this paper, and is described and illustrated in (Schindel 2005a).

The failure analysis approach this paper describes also uses the fact that the (modeled) Features for a system summarize, in stakeholder language, (all of) the behaviors of the system that will be valued by (all of) the system's stakeholders.

The balance of this paper assumes the availability of a systems requirements and design model that is based on the above metamodel. When we build on the foundation of the MBSE Metamodel, some surprising, powerful, and unifying simplifications begin to appear.

Model-Based Failure Analysis: Unifying Concepts

Features, Failures, and their Impacts. Let us assume that we have a modeled set of product requirements, based on the above metamodel. Because we have available all modeled system Features satisfying all system stakeholders, it follows that a failure is then synonymous with not delivering what a Feature promised. Because they are stakeholder ideas, modeled Features are typically not very technical in their descriptions, but in fact summarize everything that a system should deliver to its stakeholders. (This may include stakeholder-quantified feature attributes.)

Each Feature is used to generate one or more Failure Impacts. , summarizing the impact of not delivering (at least some aspect of) the Feature's promise to the stakeholder. For example:

- Feature = "The system delivers medication on a dose accurate basis."
- Stakeholder impacts of not delivering Feature = "Illness", "Disability", "Death", etc.
- Severity of impacts: 3, 4, 5

As illustrated above, each impact can also have a pre-populated severity of associated with it, describing the stakeholder-rated severity of such an impact ever occurring. Notice that this has been done so far without reference to the physical design of the system.

To cover all the Stakeholders, Features may include issues important not only to system end users, but also to those who manufacture, distribute, sell, or support the system, as well as shareholders in the profit-making enterprise, etc. We may or may not be interested in Failure Impacts on all these stakeholders, and are offered the opportunity to explicitly decide. If a failure analysis is to be limited to certain stakeholder and feature subsets, such as medical harms to patients, then the only features that need to be considered are those that have those impacts on patients.

Surprise Number 1. Our first "surprise" is that *the only Effects (the E in FMEA) that a Failure can have are non-delivery of Feature promises*—and these can be pre-modeled with each of the Features, as Failure Impacts. If we claim to know our stakeholders and their modeled Features, we can "pre-populate" the only possible Effects of Failures. If we think we have discovered an Effect that is not implied by an existing modeled Feature, we need to inform the Feature Modeler that they may have missed an important product Feature. If we don't have a model of our system's stakeholders and their modeled Features, the extended team has important homework to do before we can perform an FMEA or similar analysis. (This was always true in any method, but is made more transparently obvious by the model-based approach.)

Requirements, Interactions, and Counter-Requirements. This approach also uses the fact that the (model-based) functional requirements statements for a system describe its required behavior, occurring during the interactions the subject system has with external systems (actors). Any failure of that system will include at least one instance of an interaction behavior by the system with at least one external system, having negative stakeholder consequence. At a black box level, these are the functional failures identified in FMEA, RCM, or other failure analyses. This method builds failure analysis on top of the system's requirements model, suggesting that the failure analysis cannot be completed without an agreed set of functional requirements, in model form. (Note that model-based requirements of the type described here are a technical characterization of relevant aspects of the system's black box behavior. This degree of "completeness" is characteristic of model-based requirements of the type discussed here. This "completeness" will now come in handy, for generating FMEA Functional Failures. This also makes it even more obvious why the system requirements as viewed by the requirements analyst, designer, and failure analysis review team should all be the same modeled requirements—and that each team can improve upon the shared model work of the others.)

Each system requirement statement is used to generate at least one counter-requirement statement. For example:

- Requirement = “The system shall deliver at least 3 hours of operation on one battery.”
- Counter-requirement = “The system does not deliver at least 3 hours of operation on one battery.”

A complete set of counter-requirements can be rapidly generated in a simple way from the system’s requirements, by “reversing” them.

Surprise Number 2. *All FMEA functional failures can be rapidly generated as Counter - Requirements, from MBSE modeled functional requirements*.

Some requirements may generate more than one counter-requirement. For example:

- Requirement = “The system shall maintain temperature in the range 70-74 degrees.”
- Counter-requirement 1 = “The system allows temperature to exceed 74 degrees.”
- Counter-requirement 2 = “The system allows temperature to fall below 70 degrees.”

Furthermore, because the Requirements were already associated with the Features of a system model, the Counter-Requirements can be easily associated with Impacts, which are the (feature non-delivery) “effects” of an FMEA analysis, without “from scratch” analysis.

Surprise Number 3: *All associations (match-ups) of FMEA Functional Failures with FMEA Effects can be generated from the association of the violated Requirements with its associated Stakeholder Feature.*

Modes (States): Failure Modes. The MBSE requirements approach referenced also uses the fact that the interactions a system has with external systems can be thought of as associated with the system being in a certain state, or mode. The behavior (external interaction) of a system is different if it is “Off”, “On”, “Idling”, etc. Each of these are states (or modes) of that system’s behavior. These are all “normal” modes, in the sense that while they occur in different circumstances, the associated system behavior is considered normal (that is, what is described by requirements).

In addition, a system can sometimes enter an “abnormal” mode, in which its behavior is undesirable—such as “Overheated”. Sometimes abnormal states are called failure modes when the associated behavior is bad enough.

Interaction-State Chains; Causes. This approach further uses the fact that the Design Components, States, Interactions, Requirements, and Features information of the Figure 1 Metamodel can be unfolded (split) across normal and abnormal behavior, and across “causality chain” sequences. The resulting models add further to the information used to populate a Failure Analysis (e.g., FMEA table).

In all these cases, the current mode (state) of the system can be viewed as the immediate reason that it is behaving a particular way. That behavior is characterized by the interactions the system is currently able to perform (the interactions associated with that state).

If we then ask how the system came to be in its current state, we find that a previous interaction of some sort will have "placed it in the current state". This leads to the idea that there are "causality chains" that take the form of sequences of alternating interaction, state, interaction, state, etc. For example:

- Interaction: Turn On the System
- State: System On
- Interaction: Request System Menu
- State: Displaying Menu

This same idea works for abnormal states:

- Interaction: Insert Battery
- State: Battery Inserted Backwards
- Interaction: Turn On System
- State: System Inoperative

In all these cases, the idea of cause can be pursued by looking to earlier parts of the chain. We can say that a later part of the chain is "caused" by the states and interactions of an earlier part of the chain.

Pre-Populating A Library of Failure Modes. The Counter-Requirements and Feature Failure Impacts described earlier above depend only upon the structure of requirements and stakeholder expectations for a system—they are independent of its design. In contrast, the Failure Modes of a system depend upon its design—specifically, upon its physical Design Components. Each such Design Component has an expected behavior, based upon the logical roles and requirements allocated to it, and a set of Failure Modes, which are abnormal states that physical component type may enter in which it will display behavior violating its allocated logical roles and requirements.

Since Counter Requirements and Feature Failure Impacts can be pre-populated independent of design, is it possible that Failure Modes can be pre-populated independent of system requirements? This turns out to be connected to knowing what roles and (decomposed, or white box) requirements will be allocated to the physical part. For most physical parts playing typical or "standard" roles, it turns out that we have such a prediction available even if the (parent black box) requirements of the total system are not currently visible. For example:

- Design Component = Madsen Model P53 Centrifugal Pump
- Normal Allocated Roles = Liquid Transport, Liquid Containment, Powered Safe Operation
- Failure Modes = Bearing Failure, Leakage Seal Failure, Short to Case
- Probabilities of Occurrence = 0.002, 0.00045, 0.000001 (per 10,000 service hours)

Probability of Occurrence. As illustrated above, for each pre-populated failure mode, we can also include probability of occurrence parametric information that characterizes the likelihood of the physical component entering the failure mode from the interactions it will experience in its typically assigned roles. This will later help to drive the failure risk scoring process in the usual manner.

Combinatorial Matching Up of Requirements and Design Data. The Functional Failures (counter requirements) and Failure Effects (feature failure impact) data can be pre-populated independent of the system's internal design, and the Failure Mode data for standard component roles can be pre-populated independent of the system's external requirements. So, when both the requirements and a candidate design have become known, how do these two halves of the failure analysis model get connected to each other? This turns out to be a combinatorial algorithm.

First, it turns out that the counter-requirements (functional failures) obtained by reversing the requirements statements may describe some hypothetical external behaviors that are never (or with probability too small to matter) caused by component failure modes. This will cause some pre-populated functional failures to be dropped. For example, a requirement that a product weigh less than one pound has a counter-requirement that it weighs more than one pound. It may be determined that there is no component failure mode that impacts weight, so that this functional failure is dropped from the list. (Notice that even this failure mode could happen for some products—for example, a hazard protection suit that becomes wet weighs more.)

Second, it turns out that some failure modes of a physical component have no consequence on the product's required behavior, because the failure mode describes a role not allocated to the part in this particular product design. For example, an integrated circuit may have built-in circuitry for performing certain functions which are not used by a certain product's design, even though other portions of that chip are used.

The connection of the requirements half of the failure analysis to the design half of the failure analysis is made by matching up "mating" pairs, and discarding what is left as not applicable (after checking for missed cases this approach also helps us find—another benefit). The matching up is accomplished through the matching of counter-requirements with failure modes. Each failure mode causes some abnormal behavior. All abnormal behavior is described by counter requirements. When we find a counter-requirement belonging to a failure impact is equal to a counter-requirement for a failure mode, that pair is associated together, completing two major sections of a row in a failure analysis table. (Some failure modes may connect to multiple counter requirements and some counter requirements may connect to multiple failure modes.)

This process may use two levels of requirements, in the form of system black box requirements and their decomposed white box requirements (allocated to physical parts), in which case counter-requirements may be developed at both levels. A simpler alternate method is to use only one level of counter-requirements, with the component failure modes associated directly with the resulting abnormal behavior at the black box level—in which case the association of failure modes with abnormal behavior is dependent upon knowing the system level design. Likewise, the states discussed above may be at two levels, representing states (and failure modes) of system components and the whole system, or simplified to states of the whole system, in which case the failure modes are modes of the whole system and again dependent upon its design.

The discussion above assumes failure modes originate in internal system components, typical of analyses such as a Design FMEA (D-FMEA). Also discussed later below are failure modes of external people or processes that impact upon the subject system, as seen in an Application FMEA (A-FMEA) or a Process FMEA (P-FMEA). The counter-requirements matching-up approach is substantially the same in these cases.

Order of Occurrence versus Order of Analysis; Checking; FMEA versus Fault Tree. FMEA analysis typically reasons from component failure modes to system level counter-requirements, to the stakeholder impacts (failure effects, such as user injury). This traditional analysis thus occurs in the sequence of cause-to-effect, and the methodology described here supports that order of reasoning. In a traditional FMEA table, it proceeds more or less from left to right. This traditional order of reasoning is why FMEA is said to work for analysis of single failure modes but not multiple simultaneous failure modes.

It can be seen that this methodology also supports the generation of Fault Tree analyses. Whereas an FMEA analysis traditionally begins from each possible component level failure mode and reasons to its effect (typically a one-to-one process generating a row of an FMEA table), a Fault Tree analysis traditionally begins with each effect and reasons backwards to identify each possible component failure mode that might cause it (typically a one-to-many process generating a many-branched fault tree under a single effect). Each path of the fault tree is roughly equivalent to a row of the FMEA table. The information models described here describe both approaches, differing only by the order in which the data model is filled in during the analysis process.

The use of MBSE failure analysis allows reasoning in other directions—because it is really about an underlying information model, not an order of reasoning, we can populate that information model in different orders. These include backwards reasoning from failure effect to cause (as in a Fault Tree Analysis) and middle-out reasoning, from system counter requirement to both their upstream causes and downstream effects. This is of major value, as it facilitates completeness checking of the resulting failure analysis table. We can independently check the effects against a complete library of all possible feature-based impacts. We can independently check the middle (the system counter-requirements) against a complete library of all possibilities, based on the listed system requirements. This improves completeness and coherence of the FMEA or other analysis, including its inspectability.

Faults versus Failures; Fault Tolerant Systems; Fail Safe Aspects. In the specific language (Anderson and Lee 1981) of fault tolerant systems (which is not always used the same in failure analysis procedures) faults and failures are undesirable states or behaviors, but don't mean the same thing. A fault is an abnormal component or subsystem condition (state), which may or may not result in a system level failure. Remembering from above that failures are not delivering agreed upon stakeholder features, we can say that a fault tolerant system is a system that does not fail (continues to deliver features) in spite of component or subsystem faults. (That is, it tolerates faults in its own components, while continuing to deliver external features.)

For example, aircraft hydraulic systems typically employ redundancy, so that they can deliver safe flight services while tolerating a fault in a hydraulic line.

In the language of failure mode analysis, the term "failure mode" is frequently used to describe an abnormal state of a component or subsystem, even if the overall system was designed to keep delivering all its external services in the presence of that component failure mode. This is not so inconsistent if you consider that the subsystem or component is not delivering its "external" services, but it can be a little confusing if you don't expect the term or keep track of system levels.

Sometimes a system internal fault can present risk of a serious (e.g., life or property threatening) failure behavior by the subject system. In those cases, mitigations are sometimes planned such that, although the system may fail to deliver all of its promised features, it protects from presenting a more serious failure. That is, it still fails, but "fails safely". This is called a fail safe system.

Subsystem Causing Failure: D-FMEA. In a system, an abnormal state of a component may cause a system level failure. We can reason forward from the component state to the system failure it causes, or backward from the component state to its cause. For example, the following failure mode is "caused" by the interaction shown:

• (Interaction) Cause of Failure Mode:	Normal Wear
• Component Failure Mode State:	Gear Train Binding/Lash-Up

Remembering the idea of interaction-state chains, we can see that many such failure mode states can be said to be caused by a previous interaction, whether it is a normal use interaction or some extraordinary damaging interaction. If the causal interactions are "normal" behavior by the external systems performing them, then we could say that the failure mode is effectively inherent to the design of the subject system in its normal use. Analyzing failures of this kind is typically the subject of D-FMEA (Design Failure Mode Effects Analysis) work. Sometimes this leads to a different design to reduce the likelihood of the failure mode occurring, or in other cases to other controls (mitigations) intended to reduce the impact of the failure mode when it occurs.

In all those cases, it could be said that the role played by the subject system in normal interactions eventually leads to the failure mode of the system's component. However, it is alternatively possible that the system design is not the cause, but rather that the external systems are behaving abnormally. This case is covered in the next two sections.

Peer System Causing Failure: A-FMEA. External systems interacting with the subject system are sometimes called "peer" systems, or "actors". Unlike the subsystems or components discussed above, they are external to the subject system.

In an A-FMEA (Application Failure Mode Effects Analysis), attention is focused on the effect of abnormal behavior by external systems that are typically human "users" of the subject system. It could be said that the original failure modes in this case are states of the external system. For example:

1	Failure Mode (Pilot State):	Attention Overloaded
2	Interaction:	Select Target (assume wrong value entered)
3	State (of Weapons System):	Awaiting Weapon Release Confirmation
4	Interaction:	Confirm Weapon Release
5	State:	Delivering Weapon

As illustrated by the above example, we can have a failure to deliver overall system features even though the subject system meets all of the requirements assigned to it. However, it is also possible for an external system to drive the subject system into its own abnormal (e.g., damaged) state, after which it no longer meets requirements assigned to it. For example:

1	Cause of Failure (Interaction):	Poor User Training
2	Resulting Failure Mode (State):	User Unaware
3	Interaction:	User Closes Valve (Over-Tightening)
4	Resulting System Component State:	Valve Seal Failure

Both of these cases are of interest in an A-FMEA. The second case looks a lot like a D-FMEA after the point of driving the subject system into a bad state.

Notice that "users" are not the only external systems whose failure modes can damage the subject system's state. Other faulty systems in the Application Domain may also have to be considered.

When the external actor that is in an abnormal state is a human being, the MBSE model is in the territory of modeling human behavior. This is further discussed in (Schindel 2006).

Peer System Causing Failure: P-FMEA. One special external system traditionally analyzed is the subject system's manufacturing system. This is the subject of a P-FMEA (Process Failure Mode Effects Analysis). The nature of a manufacturing system is to create the subject system, so it may be found that all the P-FMEA failures of interest result in bad product system states. For example:

1.	(Interaction) Cause of Failure Mode:	Glue Build-Up on Nozzle During Use
2.	Component Failure Mode State:	Nozzle Obstructed
3.	(Interaction)	Not enough glue applied
4.	Subject System State	Part Loose

There can also be manufacturing process failures that fail in the sense of not delivering on all the other manufacturing process systems features, as when manufacturing yield, manufacturing operating cost, or manufacturing safety are impacted by manufacturing faults. Depending on the intended scope of the P-FMEA, these may or may not be of interest to include and analyze.

Other major processes, such as the commercial Distribution Process, can have faults that create bad states in the subject system. For example:

1.	(Interaction) Cause of Failure Mode:	Transport Packaged Product
2.	Component Failure Mode State:	Package Seal Fractured
3.	(Interaction)	Tolerate Exposure to Contaminants
4.	Component Failure Mode State:	Food Product Contaminated

Depending on the intended scope of the P-FMEA, these other processes may also be considered.

D-FMEA, A-FMEA, P-FMEA, and Unified FMEA. Although it may be desirable to separate the D-FMEA, P-FMEA, and A-FMEA "reports" for attention by different groups, and to generate and review them using different subject matter experts, it is also desirable to generate them from a consistent underlying information model. For example, all three FMEA types depend on the same system level counter-requirements and feature impacts. If this consistency is used, then it is easier to understand the different FMEAs in a consistent way, and to judge their accuracy and completeness.

While there may be reasons to differently format or label the tabular "reports" that are generated for these different types of failure analysis, the approach described here at least intends to generate them from a common base of underlying information, and to minimize differences in labeling except where it improves the outcome.

Further Leveraging the Results

Patterns As Re-usable Models. This paper describes the use of Model-Based Systems Engineering information in failure analysis, to improve results. If an enterprise needs to perform failure analysis on different products or systems that are somewhat related but vary in their specific configuration (e.g., product lines), then a more powerful extension is also available. This is called Pattern-Based Systems Engineering. The basic idea is to make the models configurable and re-usable, so that they can rapidly be re-used in future projects, and can also be used to accumulate learning. This is a bigger idea than accumulating standard lists of failure modes.

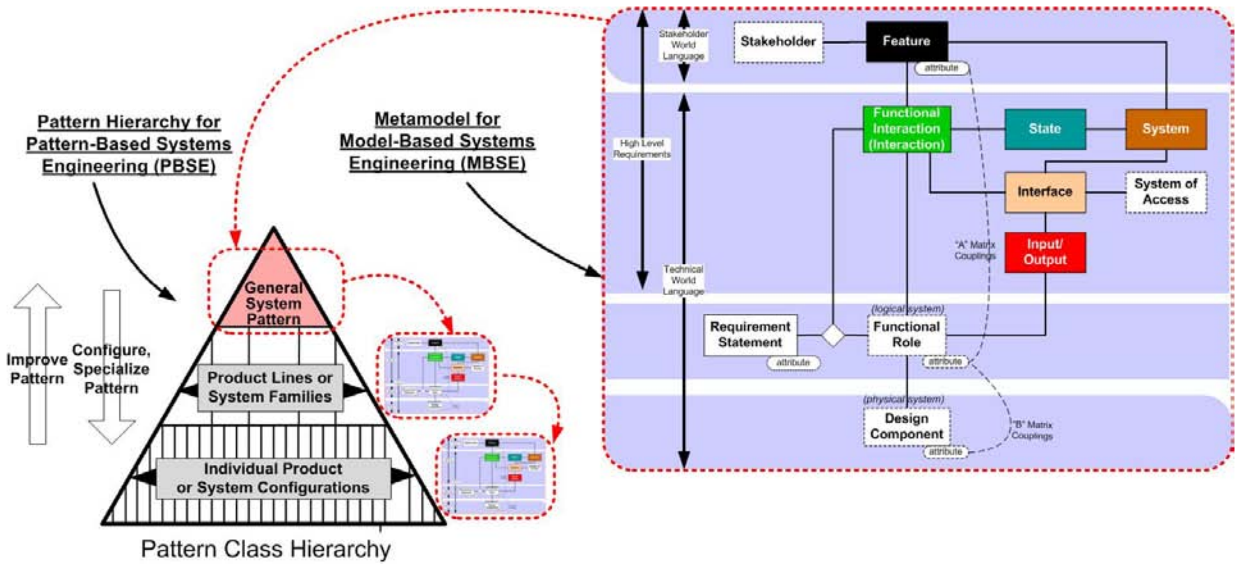

Figure 2: Patterns Are Re-Usable, Configurable Models

This approach to Systems Engineering Patterns treats a pattern as a configurable, re-usable model of requirements and design, described further in (Schindel 2005b; Schindel and Smith 2002).

Enhanced Use of FMEA and Risk Analysis Tools. A number of basic and more advanced commercial automated tools are available for use in generating FMEA and other forms of failure or risk analysis. In their most basic use, the analyst manually enters data into relatively fixed forms and generates resulting reports. In their more advanced form, these tools support customization or configuration of reports, data entry, and some aspects of the underlying information models. Some also support accumulation and use of re-usable standard categories or other data, and some support integration with other engineering tools, such as requirements management tools.

The model-based concepts, methodology, and procedures described in this document can be used with a number of these commercial tools, improving their value. In general, the more powerful and flexible the tool, the more aspects of this methodology may be used.

The simplest, but least beneficial, way to initially do this is to configure the tables and reports of a tool to accept manual entry of data of the type described in this document.

A more sophisticated approach allows re-use of data from a pattern of requirements, design, and failures (patterns). Since patterns are relational models, this is more powerful than simply having lists of standard pull-down items.

This methodology also enhances the ability to integrate an FMEA or failure analysis tool with a Requirements Management tool, by using Counter-Requirements that are associated with the system level Requirements. This is more powerful than simply having links between data items in two tools. In fact, if a requirements and design model is available in MBSE form, then tool-based combinatorial algorithms can be used to automatically generate an initial draft FMEA table. Of course, this does not replace human analysis, but does reduce the drudgery of initial generation, freeing the analyst to do deeper thinking and analysis of the failure data.

Results to Date

We have seen these methods help both experienced FMEA analysts as well as newcomers to more productively generate well-organized failure analyses, in applications including manufacturing and health care. The approach is not at odds with traditional methods, in producing substantially the same form of deliverable—but provides a stronger basis for understanding the meaning and degree of coverage that deliverable represents, while more tightly integrating failure analysis with requirements and design data.

Conclusions

1. Failure analysis data and processes can be more deeply integrated with system requirements data and processes, using model-based methods, with benefits to depth of shared team understanding, productivity, process cohesion, coverage, and lower level of entry expertise for participants.
2. A subset of FMEA analysis can occur in advance of, or independent of, system design, using the structure of model-based stakeholder features and functional requirements to pre-populate the space of potential functional failures and their prioritized effects.
3. Another major subset of failure analysis data can be pre-populated that is requirements independent, in the form of libraries of physical components (or technologies), their typically assigned roles, and their failure modes and associated abnormal behaviors.
4. Modeled system design introduces failure mechanisms for D-FMEA, while human, process, and equipment actors introduce failure sources for A-FMEA and P-FMEA, all of which can be better integrated.
5. FMEA, Fault Tree, and other forms of analysis can be viewed as different views of the same underlying modeled data, for different purposes and emphases.
6. Patterns, when formed as re-usable, configurable models of system requirements and design, can include failure risk analysis, whose coverage and quality can be improved from project to project, in support of a learning organization.
7. Automated tools for failure analysis, requirements management, design, simulation, and other aspects of the systems engineering process can be integrated more deeply than simply linking their data records, by configuring their databases to take advantages of the integrated underlying MBSE/PBSE metamodel.

References

Anderson, T. and Lee, P. 1981. *Fault tolerance: Principles and practice*. Prentice-Hall (1981) ISBN-10: 0133082547

Dyadem. 2002. *Guidelines for failure mode and effects analysis, for automotive, aerospace and general manufacturing industries*, ISBN 0-9731054-1-0. CRC Press, 2002

---. 2003. *Guidelines for failure modes & effects analysis for medical devices*, ISBN 0849319102, Richmond Hill, Ontario, Canada: Dyadem Press, 2003.

Estefan, J. 2009. "Survey of model-based systems engineering (MBSE) methodologies", INCOSE-TD-2007-003-01, Rev B

Hyatt, N. 2003. *Guidelines for process hazards analysis, hazards identification & risk analysis*. ISBN 0849319099, Richmond Hill, Ontario, Canada: Dyadem Press, 2003.

Hybertson, D. 2009. *Model-oriented systems engineering science*. CRC Press, 2009.

INCOSE 2009. INCOSE Model driven system design working group web site, `http://www.incose.org/practice/techactivities/wg/mdsd/` .

ISO/IEC. 2006. "IEC International standard 60812, analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)", 2006.

---. 2007. ISO/IEC 14971:2007, Medical devices—application of risk management to medical devices. ISO/IEC. 2007.

Moubray, J. 1997. *Reliability-centered maintenance*, Second Edition, New York: Industrial Press, Inc., 1997.

Powell, M. 2005. "Dealing with uncertainty in systems engineering". INCOSE IS2005 Tutorial F02. July, 2005

Schindel, W. 2005a. "Requirements statements are transfer functions: an insight from model-based systems engineering", *Proceedings of INCOSE 2005 Symposium*, July, 2005.

---. 2005b. "Pattern-Based Systems Engineering: An Extension of Model-Based SE", INCOSE IS2005 Tutorial TIES 4.

---. 2006. "Feelings and physics: emotional, psychological, and other soft human requirements, by model-based systems engineering", *Proceedings of INCOSE 2006 Symposium*, July, 2006.

Schindel, W., and Smith, V. 2002. "Results of applying a families-of-systems approach to systems engineering of product line families", SAE International, Technical Report 2002-01-3086, November, 2002.

US DoD 1980. MIL-STD-1629A, "Military standard procedures for performing a failure modes, effects, and criticality analysis", 1980.

BIOGRAPHY

William D. Schindel is president of ICTT System Sciences, a systems engineering company, and developer of the Systematica™ Methodology for model and pattern-based systems engineering. His 40-year engineering career began in mil/aero systems with IBM Federal Systems, Owego, NY, included service as a faculty member of Rose-Hulman Institute of Technology, and founding of three commercial systems-based enterprises. He has consulted on improvement of engineering processes within automotive, medical/health care, manufacturing, telecommunications, aerospace, and consumer products businesses. Schindel earned the BS and MS in Mathematics, and was awarded the Hon. D.Eng by Rose-Hulman Institute of Technology for his systems engineering work.

V1.2.1