26th Annual INCOSE International Symposium (IS 2016)
Edinburg, Scotland, UK, July 18-21, 2016

Helping Undergraduate Students of any Engineering Discipline Develop a Systems Perspective

Mario Simoni
Rose-Hulman Institute of Technology
5500 Wabash Ave, Terre Haute, IN 47803
(812) 877-8341
simoni@rose-hulman.edu

Eva Andrijcic
Rose-Hulman Institute of Technology
5500 Wabash Ave, Terre Haute, IN 47803
(812) 877-8893
andrijci@rose-hulman.edu

Bill Kline
Rose-Hulman Institute of Technology
5500 Wabash Ave, Terre Haute, IN 47803
(812) 877-8136
kline@rose-hulman.edu

Ashley Bernal
Rose-Hulman Institute of Technology
5500 Wabash Ave, Terre Haute, IN 47803
(812) 877-8623
bernala@rose-hulman.edu

Abstract. The complexity of problems that engineers are being asked to solve is increasing rapidly. Effective solutions often require the integration of mechanical, electrical, computer software, chemical, and/or biological components. In order to manage this complexity, it is becoming important for all engineering students to learn how to approach the solutions to these problems using a systems perspective (Baldwin 2014). In order to better motivate this approach to students the authors are introducing it within courses of their own engineering discipline. The authors are adapting traditional systems engineering concepts to create a framework of system models that can be introduced into courses of any engineering discipline at any level. Through the process of learning how to create these models, students gain an understanding of what is meant by a systems perspective and how this perspective can help them to solve problems. This paper discusses which systems models were incorporated into undergraduate curriculum and how each model is broken-down into pieces that are easier for undergraduates to understand and faculty to teach.

Introduction

The complexity of modern technology is making it increasingly more common for new engineering graduates to work with products and systems involving multiple disciplines and domains (Baldwin 2014). They must have a broader understanding of their job going beyond a single domain perspective or an even more limited perspective of a single sub-discipline within that domain. More employers are asking for students who understand a *systems perspective* of engineering. Unfortunately, the general idea of a *systems perspective* is a very nebulous concept that can change drastically depending on who is describing that perspective.

The authors are proposing a model of a *systems perspective* that is similar to the orthographic views in a CAD drawing as shown in Figure 1. In such a CAD drawing, the number of 2D views shown is the minimal set that completely captures all of the features of the 3D object. Achieving consistency between the views is a way of self-checking for complete coverage of those features. When solving open-ended problems, the end result is much more nebulous than the 3D CAD object, which means that many more views may be needed to capture all of the

important aspects of the solution. In the most basic sense, the authors are defining a *systems perspective* as an ability to see the solution as a system that accepts inputs and/or produces outputs and is itself composed of various subsystems. The behavior that is needed to solve the problem is understood through the interaction of the various subsystems with the system's inputs and outputs. Applying a *systems perspective* to solving problems, then, is about understanding the many different views of a system, the process of creating those views, and the process of making those views consistent with each other in order to achieve a more optimal solution (Haskins 2010).

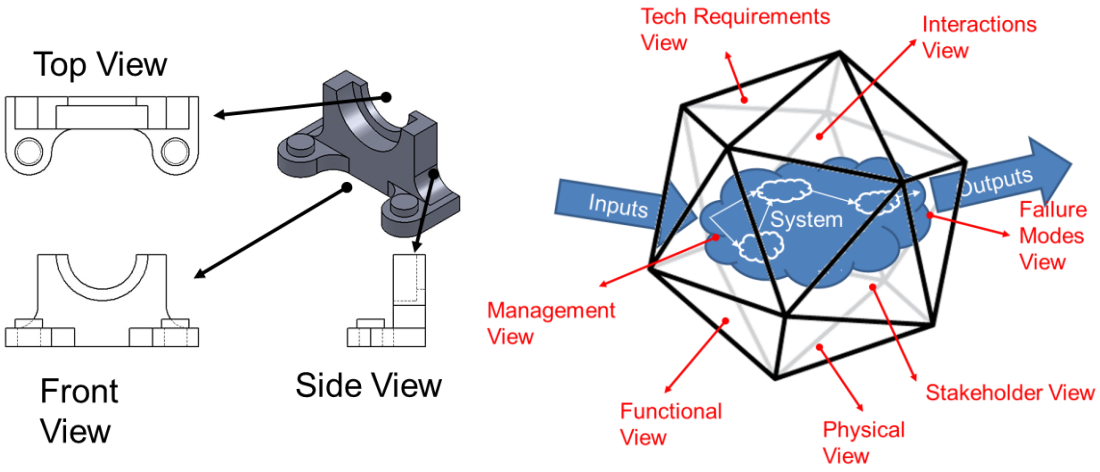


Figure 1. Definition of having a *systems perspective* as understanding multiple views of a system

In order to help define these views, a core set of technical *systems competencies* was identified in a previous effort (Schindel et al. 2011). These competencies provide a basic skill set that can be applied within any discipline and across an entire curriculum. Exposing students to these competencies within the context of their discipline enables them to still achieve a high level of technical competence within their discipline while showing them the relevance and importance of the *systems perspective*. The set of behaviors that are identified in the technical Systems Competencies include: 1. Describing the solution **as an interconnection of subsystems but also in terms of the target's interaction with the larger system that surrounds it (a functional architecture)**; 2. Applying a **system stakeholder view** of value, trade-offs, and optimization; 3. Understanding a system's **interactions and states** (modes); 4. Specifying system **technical requirements**; 5. **Synthesizing a physical architecture** from the high-level design; 6. Assessing solution **feasibility, consistency, and completeness**; and 7. Performing system **failure mode and risk analysis** (Schindel et al. 2011).

The field of model based systems engineering (MBSE) has produced a framework and standard vocabulary for applying these behaviors in order to create tangible model-based representations of projects (Buede 2000). The models developed are generally graphical, making it easier to see relationships between different system-level aspects of a solution. In addition to facilitating student learning, taking a model-based approach provides significant advantages to faculty. Having a standardized procedure for creating the models and vocabulary for describing them means that pedagogy can be developed that will be applicable to all types of projects and disciplines. The regular structure and appearance of the models across vastly different projects makes it much easier for faculty to guide students through a solution process with meaningful feedback. The model structure also makes it much easier to more fairly assess the students' final results using standardized rubrics.

This paper describes how the authors have adapted formal MBSE approaches to make them more accessible to undergraduate engineering students from several engineering disciplines. The models and approach to developing them provide students with opportunities to practice, assess, and refine the technical systems competencies. The authors represent several different academic departments and have developed pedagogical materials for different academic levels and disciplines, illustrating that these competencies have relevance to engineering education in a very general sense. Rather than creating a separate course to introduce these concepts, they have been included as supplementary content in already existing courses. The application of these competencies has yielded measurable improvements in student understanding of system related issues.

The authors would also like to acknowledge that many of the ideas and models presented in this paper are described with terminology that might be used slightly differently than in traditional systems engineering. The authors are not trying to suggest changing any of the traditional definitions or to push for any new standardization; however, we found these changes helpful when describing these concepts to undergraduate students, who have limited background knowledge or experience. The authors encourage the reader to adapt these ideas in whatever way is most comfortable. Terms that were intentionally chosen and adapted to this framework are highlighted by italics the first time they are used. Definitions for each of these terms are also provided when they are first used in order to describe the authors' intended meaning in the context of this framework.

Descriptions of the System Models and Design Process

This section describes each of the different models that were developed, the processes that students would go through to create them, and the assessment tools that faculty and students would use to assess them. The models are presented in the order in which they are introduced to the students as shown in the top-down linear progression with feedback in Figure 2 (Schindel 2011a). The authors have found this order to be helpful in the sense that it gradually exposes the students to new concepts and puts emphasis on the ultimate goal of satisfying the Stakeholders' needs. However, strong emphasis is placed on the central idea that the optimal solution is reached when all of the different models are consistent, not when a particular design process is completed. As such, system design is a continual process of iteration and revision of models, and then cross checking for consistency. Once the models are well understood, the design process can be thought of as more of a circular process rather than linear as indicated by the illustration on the right in Figure 2.

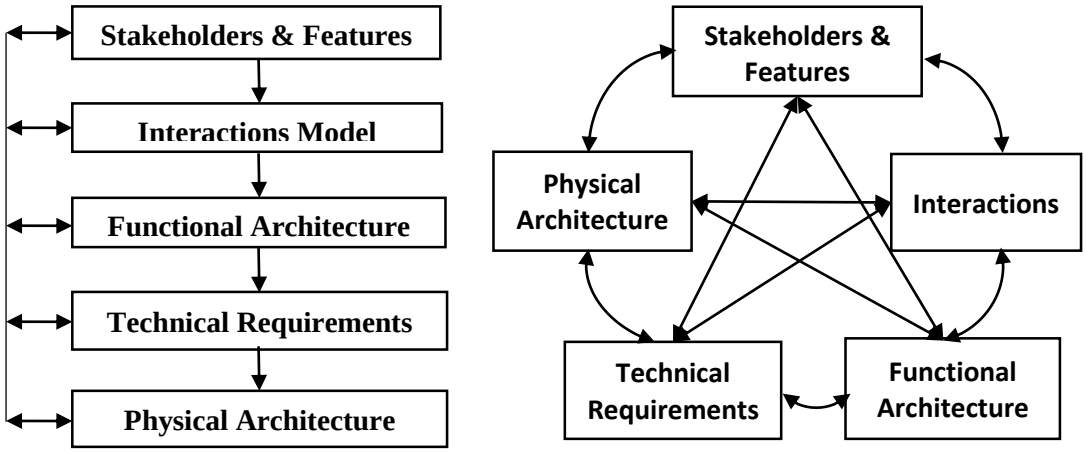


Figure 2. Models for the process that was used to present systems competencies to students and faculty.

The authors have found it helpful to first expose students to these systems concepts using a familiar example, which for this paper will be a remote control for consumer electronics. These examples are being vetted in undergraduate engineering courses at the authors' institution and also in workshops that introduce these concepts and models to other faculty. These examples and the pedagogical theory behind them are continually being refined as the authors are discovering new ways to think about and present them. One of the biggest challenges to presenting these concepts to undergraduates and other faculty is to simplify them as much as possible without losing the meaning.

Stakeholder and Feature Model

One of the bad habits that the authors have observed in undergraduate engineering students is a strong tendency to begin constructing a solution without fully understanding who the stakeholders are and what it is that they want. Without knowledge of the stakeholders and features that they desire, it is difficult to understand when a solution is achieved much less the quality of the solution. Without some sense of what is more or less valuable to the quality of the solution, it is difficult for students to make design decisions. The purpose of this first model is to help students understand what is valued from the solution to the problem.

One of the first steps in creating a stakeholder and feature model is to identify *stakeholders* as the individuals or groups that have vested interest in the outcome of the project/problem. To help students identify stakeholders the following probing questions were found to be beneficial: 1) Who is dissatisfied with the current situation?; 2) Who is affected by changing the situation?; 3) Who will authorize a change in the situation?; 4) Who benefits from a solution?; 5) Who benefits from no solution being enacted?; 6) Who is affected if we fail to enact a solution?; 7) Who is affected if our solution fails?

A master list of common stakeholders is provided to the students to ensure that they didn't miss a key stakeholder. Common stakeholders include: end users, clients, other engineers or scientists, regulatory agencies, those who maintain/repair/update, societal groups (i.e. government, police/fire departments, new generation of teenagers), manufacturers, shipping departments, marketing/sales/retail departments, legal departments, and those responsible for disposal/deletion of the software.

Table 1 shows a sample list of stakeholders for the remote control example. This part of the model is a simple table that identifies each stakeholder, assigns each a level of importance, and provides a more detailed description. Typically students are capable of developing a list of stakeholders with relative ease. The definition of each stakeholder is an important part of the model because it forces students to fully identify the stakeholder and eliminate ambiguity. For example, if the students simply list "manufacturer" without a definition, they could be talking about the remote control manufacturer or the television manufacturer that wants to include the remote with their television. The Importance column is included in order to help students understand how to make design decisions with respect to the stakeholders. Those stakeholders who are more important should have more influence when making decisions.

The next step to develop the model is to identify features. *Features* are characteristics of the system that are valued by stakeholders—they express the value space/ fitness space/tradeoff space of the stakeholder. Features should be written in stakeholder language and typically include "-able" words. Some of the more common features include: something to describe the project's primary purpose, affordable, small-size/form factor/weight/compactness, easy to use, adaptable, recyclable, secure, robust, efficient, environmentally friendly, simple, and repairable. Table 2 shows some of the features that are associated with the remote control

example. Single words or phrases are helpful when referring back to the features, but they are insufficient to fully capture the value of the feature in the stakeholders' language. Defining each feature in more detail is important to help students clarify and better communicate what they value.

It is important for students to understand that features provide different levels of value to the stakeholders. Students and engineers often like to include a large number of features in a system design. It may not be possible to implement all of the features in the first version of the solution. In some cases, interesting technical features may be included that provide little or no value. Often students will forget to include the primary purpose of their project/problem they are trying to solve when listing features.

Table 1. Stakeholder model including a description of the stakeholder and ranking of the level of importance

Stakeholder	**Importance**	**Description**
Consumer	highest	The end user who purchases the remote control; physically interacts with the remote device to complete tasks such as turning on the TV with the remote, changing channels, (etc.)
Manufacturer	moderate	The company responsible for manufacturing the remote

Table 2. Features desired by the stakeholders along with descriptions of the feature

Feature	**Feature Definition**
Controllable	The remote must have the necessary interfaces for controlling consumer electronic devices that were produced from 1990-2020.
Versatile	The remote should be able to control a wide variety of features for consumer electronic devices produced from 1990-2020.
Durable	The remote must continue functioning normally despite harsh environmental disturbances such as spilled liquids, being dropped, and being chewed on by animals and babies.
Programmable	The remote should be programmable in order to produce the required control signals for the different consumer electronic devices.

After creating these lists, the students create a graphical model (see Figure 3) that maps the stakeholders to the features that each stakeholder desires. By developing a comprehensive list of stakeholders and features, students may be able to notice conflicts between features and/or the complexity of the required design in order to meet all of these features. For example, although the recycler would like the product to be easily recyclable the durability requested by the consumer might prevent the most optimal material for recyclability to be selected. At this point in the process, the students should be able to have fruitful conversations with their clients or key stakeholders to prioritize features.

To help the students self-assess their work, a rubric is provided to the students, as shown in Table 3. The rubrics are divided into Objective goals, which are almost binary in nature, and Qualitative goals, which are more subjective. The Objective Goals are meant for the students to apply themselves in order to help guide their efforts as they are developing the models. By applying these rubrics, the first versions of the models that the students create are more complete than they would be otherwise. As such, faculty assessment of the models can focus less on the models' surface-level appearance and more on the quality of the model to fully

represent the solution. The Qualitative Goals are meant more for the instructor or an experienced engineer to use for this higher-level assessment of the models.

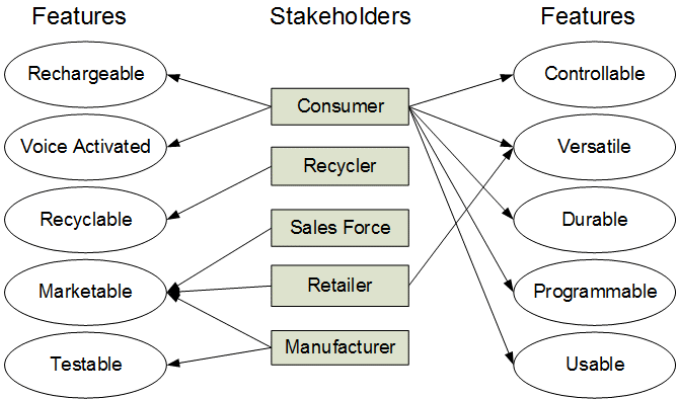


Figure 3. Stakeholder/feature model mapping stakeholders to their desired features

Table 3. Rubrics to assess the Stakeholder/Feature model

Objective Goals	Check
Relevant stakeholders from the master list are included in the model	
Relevant features from the master list are included in the model	
The primary intended purpose of the solution is included as a feature	
Each identified feature has a least one stakeholder	
Each identified stakeholder is mapped to at least one feature	
The team has identified a subset of the most important stakeholders and features	
Each identified stakeholder and feature is given a definition	
Qualitative Goals	
The solution would be satisfactory if it had only these stakeholders and features	
The mapping of features to stakeholders is complete	

Interactions and Functional Architecture

Another consequence of the students' tendency to rush into the physical solution is that they do not have a clear idea of what constitutes the system boundary. For example, in the remote control example, the power supply could be something that is part of the solution space, or it could be outside the solution as something that the end consumer supplies. Even if the students have some idea of what the boundary is, they tend to focus only on the design inside the system boundary and ignore what is going on outside the system. For example, they may begin the project by purchasing parts and building things, without considering how the user will interact with the remote. The next three modeling steps are closely intertwined and help students to identify the system boundary and then describe the system both looking into the boundary from the outside and also looking out of the boundary from the inside.

The first step includes identifying and defining actors. *Actors* are physical entities that interact with the system. An important distinction for students is that actors are described by nouns and generally provide and/or accept energy, materials, or information from the system of interest. It is helpful to keep the focus of a model limited to a particular concept or idea. The authors have observed students mixing several different ideas into a model, which makes it difficult to interpret what a model means. For example, students often try to call "Repair" an actor, which is a **behavior** rather than a **thing** that interacts with the system. Limiting actors to be nouns helps students to focus on what an actor is. Some of the stakeholders listed may be actors but

actors include more than simply people. Actors can include environmental factors such as wind, rain, and/or other devices that physically come into contact with the system being designed. Table 4 shows an example of actors that interact with the remote.

Table 4. Examples of actors along with a definition of each actor

Actor Name	**Definition**
User	The person using the remote
Wall Outlet	The outlet that is used to recharge the battery
Assembly Worker	Personnel who assembles the remote
Unintended Actors	Objects and entities that are not intended to interact with the remote, but could have significant impacts on the remote. Examples include hard surfaces, the environment, babies, animals, other electronic equipment, etc.

Once actors are identified, the inputs and outputs (I/Os) are determined. The *I/Os* describe what is being transferred between actors and the system. I/Os are also limited to be described with nouns. Common I/Os include: signals, power, energy, force, information, mass, light, heat voltage, and current. I/Os describe **what** is being transferred between two or more entities that are playing "roles" in the interaction. Each actor is responsible for at least one I/O if not more. Table 5 shows examples of I/Os for the remote control system.

Table 5. Examples of I/Os for the remote system

I/O Name	**Definition**
User Requests	User commands that are supplied to the remote. These could be in the form of touch, pushing buttons, voice commands, hand motions, etc.
Electric Power IN	Electrical power from the wall outlet that is used to recharge the power source
Unintended Inputs	Any mass, force, environmental change that is unintended as an input to the system. Examples include impact forces from dropping the remote onto a hard surface, saliva from babies and other animals, and spilling drinks onto the remote.

The students can now develop a *black-box model* of the system architecture which displays all of the actors on the outside of the system, as well as the I/Os that are being transferred between the actors and the system. Figure 4 shows a generalization of a black-box model of a system. At this point in the modeling process the students don't yet know exactly how the actors interact with the system, or how the system's inputs are converted into the outputs. For example, students might know that a voice signal or a physical push ought to somehow be supplied to the system, but they have not defined how the remote control will transform those input signals into an output that can control consumer devices.

Using the black-box model, students can identify how each actor interacts with the system. An *interaction* is another modeling construct that describes exchanges of information, materials, or energy between the actor and the system. In order to help students focus on what interactions mean, the names of interactions are limited to verbs followed by nouns. Often many interactions must be defined to fully describe a system. The Interactions model is a table that lists and describes all of the different interactions at the black-box level. At this stage in the process, no internal system detail has been developed so all interactions occur between actors and the system in general or between different actors.

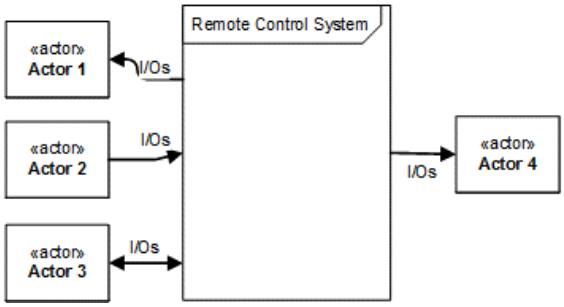


Figure 4. Black-box model of the system

Table 6 provides two of the interactions in an Interactions model for the remote control example. The Actors, System I/Os, and Features columns are added to the table in order to facilitate cross-checking between models. Every actor and I/O should be included in at least one interaction, and if not, then one or the other is not necessary. In order to ensure that the entire feature set is covered by the model, each feature should be associated with at least one interaction and every interaction should be associated with a feature. This Interactions Model can be generated by systematically looking at each actor and I/O and thinking of all the different ways that it interacts with the system.

Table 6. Examples of how interactions are represented.

Interactions	**Description**	**Actors**	**System I/Os**	**Features**
Change Settings	The user adjusts all possible device settings such as volume, channel, and guide navigation and the remote responds with the system state of the devices and the remote	User Consumer Electronic Device	User Requests Com Signal	Controllable Versatile Usable Voice Activated
Recharge the battery	The user connects the remote to the power grid and the remote signifies the user when the power supply is fully charged.	User Wall Outlet Power Grid	Plug Forces Electrical Power IN System State Info	Rechargeable

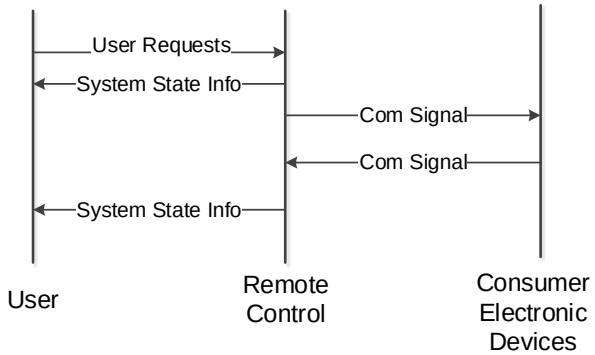


Figure 5. Example of a sequence diagram for the Change Settings interaction.

A way to help students think about and define a particular interaction is through the development of a sequence diagram. A sequence diagram depicts the order of exchange of inputs and outputs between actors and the system. A sample sequence diagram for the Change Settings interaction for the remote control system is shown in Figure 5. The vertical lines represent the system and each actor that is involved in the interaction. The exchange of I/Os

(represented as the arrows) describe the temporal order from top to bottom of exchanges during the interaction.

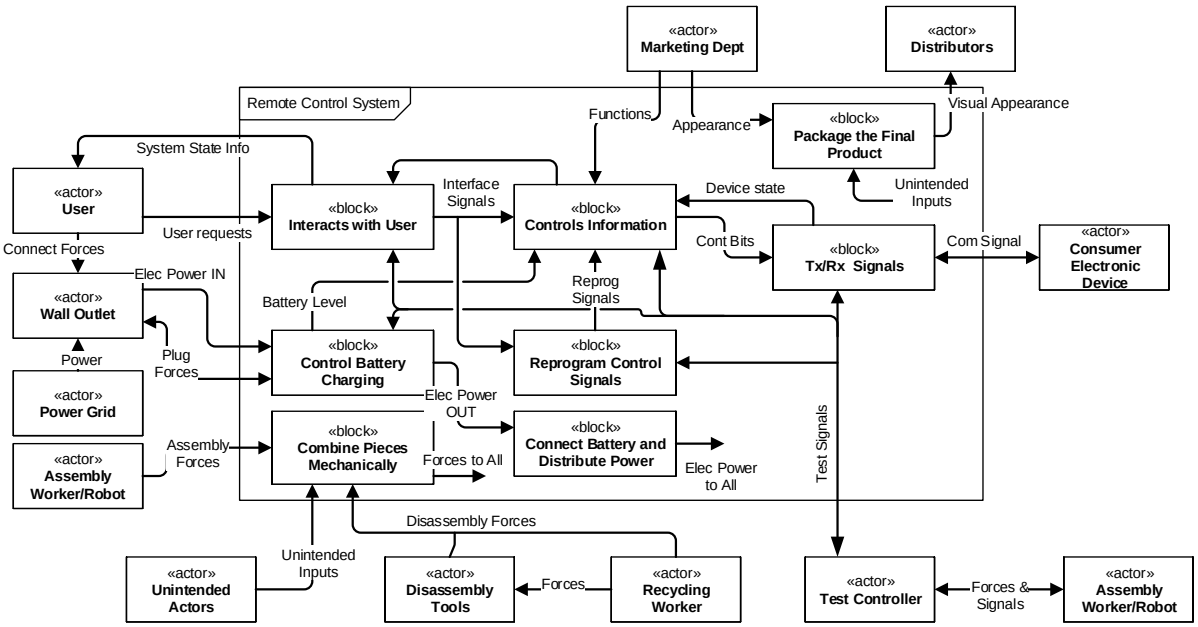
A set of binary rubrics similar to Table 3 were also developed for the Interactions model. These rubrics help students to ensure that 1) every actor and I/O is included in at least one interaction, 2) every interaction is named as a verb followed by a noun, 3) Every interaction includes at least one or more actors and I/Os, 4) every feature is addressed by at least one interaction, and 5) the primary purpose of the system is represented as an interaction. The qualitative goal is to ensure that the interactions are sufficient to fully cover the feature set.

Creating the Functional Architecture

Once the students have identified how the system interacts with what is outside of the system boundary, they are asked to think about how the system internally implements those interactions. Again students have a tendency to jump to a physical solution so the next models are intended to keep them thinking at a higher level in order to facilitate exploring different design tradeoffs. The students are asked to decompose the system into its basic functions and describe them in the *Functional Architecture*. A *function* is a transformation of one or more inputs into one or more outputs. Functions may act on I/Os at the system boundary, but also may interact with other functions inside the system boundary. Because functions define the relationships between inputs and outputs independent of the physical form, the students are taught that functions are described with a verb followed by nouns, not as physical objects.

The scope of the Functional Architecture depends upon the system *mode*, which is an operational capability of a system (i.e., *on, off, hibernate*). The primary mode of interest is the general operational state, but students are encouraged to think about behaviors that will occur during other modes of the system's lifecycle. For example, testing for quality control is a different mode than the operational state. An effective solution will need to consider how the system will behave during testing, but if all of the unique actors and functions of the testing mode are included, the Functional Architecture could become so complicated that it would be less useful. To achieve a compromise, students are asked to represent behaviors, actors, and I/Os from these other modes as simply as possible. An example related to testing might be to include a single actor and I/O that represents all of the test equipment and all of the test signals that transfer between the system and the test equipment. The interaction of all of the functions must describe the overall behavior of the system for the specified mode(s).

The physical structure of the Functional Architecture consists of functional blocks inside the system boundary of the black-box model to create the *white-box* model and a table that identifies and describes each of the functions. Figure 6 shows a possible Functional Architecture for the remote control system and describes a subset of the functions. As indicated by the Function column, we utilize the "function is a verb followed by a noun and not a physical object" syntax. This allows students to be more flexible on what the "Control Battery Charging" function really means and to explore different ways of implementing this behavior. Note in the description column that the function is described for two different system modes: operating mode in which the remote is being used by a user, as well as the testing mode when the remote is being tested for final use. While the description of the function does not presume any specific physical form, each function must address some of the stakeholder features identified in an earlier step. Those specific features will help to guide the functional decomposition and the mapping of the functions onto the physical architecture.



Function	Description	I/Os	Feature(s)
Control Battery Recharging	Takes *Electrical Power IN* from *Wall Outlet* and converts it into an *Electrical Power OUT* that is compatible with recharging the *Battery*. *Mechanical Forces* are applied between the *Wall Outlet* and the remote control system by the object that transfers energy from the wall outlet to the system. Monitors the *Battery Level* and communicates that information to the *Controls Information* function. Must accept and provide *Test Signals* during quality control testing.	Plug Forces Electric Power IN Electric Power OUT Battery Level	Rechargeable

Figure 6. A functional architecture and an example of a function definition for the remote control system.

In the process of developing the functional architecture, students must also think about developing or deriving new inputs and outputs, ones that are internal to the system, and that connect different functional blocks. Notice also that the functional architecture in Figure 6 contains functions and actors that are related to several system life-cycle stages, namely, operational, testing, assembly and recycling, and marketing and distribution. A separate functional architecture could be developed for each of these life cycle stages if more detailed functions were needed.

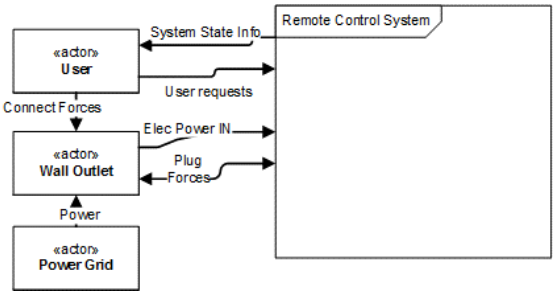
A set of binary rubrics similar to Table 3 were also developed for the Functional Architecture model. These rubrics help students to ensure that 1) every I/O at the system boundary goes to a functional block, 2) the system and each function block has at least one I/O, 3) the primary purpose of the system is represented, 4) each actor, I/O, and function has a definition, 5) each actor and I/O is labeled as a noun, and 6) each function is labeled as a verb followed by a noun. The qualitative goal is to ensure that the functional architecture is sufficient to satisfy the feature set.

Writing Technical Requirements

Within the context of traditional undergraduate engineering courses, students have very little understanding of how to create requirement statements or why they are necessary. Within the context of the systems framework in this paper, they are presented as one of the first steps towards synthesizing the Physical Architecture. The authors are defining *technical requirements* of the system as descriptions of the interactions in finer detail with quantitative or measurable values. The technical requirements are realized as text statements that help define the characteristics of the system in clear and quantitative terms so that it can be properly designed and realized (Schindel 2005a).

The term "requirements" can be used in several different ways in the systems engineering community. What are defined as *stakeholder features* in this paper can sometimes be called requirements because they describe characteristics that the stakeholders want to see fulfilled. While describing value to the stakeholder, those feature definitions are usually not described in scientific or engineering language and so they are not as useful for engineers to create a physical implementation of the system. A distinction is also made between *design constraints* , which describe limitations imposed by stakeholders, and *technical requirements*, which describe relationships between actors and the system or function blocks. A common example that students face is that the client may specify that the problem be solved with a particular piece of hardware. This request would be considered a *design constraint* because it limits the solution space rather than specifying a relationship. In addition to specific client requests, other common design constraints include standards and limitations imposed by regulatory agencies.

The process of creating technical requirements is first presented for the black-box level by systematically describing each of the interactions at the system boundary. Each of those interactions can be traced in the functional architecture from an actor into the system, from the system to an actor, or as a combination of both. Theoretically, it should not matter how a system is implemented, as long as it satisfies all of the black-box requirements, it should implement all of the stakeholder features.



Interaction	Block	ID	Requirement	Feature(s)	Verified By
Recharge the Battery	Wall Outlet System	RB-1	The Wall Outlet must *provide* an Electrical Power In at a [Wall Voltage of 120VRMS].	Rechargeable	Instrument test
Recharge the Battery	System User	RB-2	When charged, the remote should *produce* a flashing Battery charge indicator with a [Charging Symbol of a filling battery] and a [Minimum brightness of 10 lumens].	Rechargeable Usable	Demonstration Instrument test

Figure 7. Example of a how the Technical Requirements can be derived from the Interactions and the Functional Architecture.

For the remote control example, Figure 7 illustrates two possible technical requirements that describe the "Recharge the Battery" interaction. Note that in addition to the actual description of the technical requirement, several other columns are included in order to facilitate the cross-checking of different views of the system. The Interaction column is included in order to ensure that every interaction is addressed. The Block column is included to ensure that every Actor is included in at least one technical requirement. An ID is given to more easily refer to a specific requirement because many technical requirements can be generated. The requirement description has a special syntax in order to more easily extract information for cross checking. The I/Os that are involved in the requirement are underlined. The attribute of the requirement that is being evaluated and its measurable value is contained in square brackets []. The relationship between the system and the involved actor(s) is *italicized*. The Feature(s) column is included to ensure that every stakeholder feature is addressed by at least one technical requirement.

Emphasis is placed on the idea that each technical requirement must also be measurable and testable. The Verified By column in Figure 7 requires students to provide a method for verifying each technical requirement. For example, requirement RB-1 will be verified by instrument test of the wall outlet. In order to orient them on the level of detail required at this level of the design process, students are provided some examples of possible verification strategies: inspection, instrument test, analysis and simulation, or demonstration.

The number of black-box requirement statements that are necessary to fully describe a system is highly contextual. The authors have found the general guideline of 10-30 requirements to be helpful to students. Fewer than 10 indicates that the students likely have not thoroughly thought through the problem. More than 30 indicates that the students are likely thinking at too low of a level or made the requirements more specific than they need to be. In which case, it may be possible to combine some of the requirement statements in order to simplify record keeping.

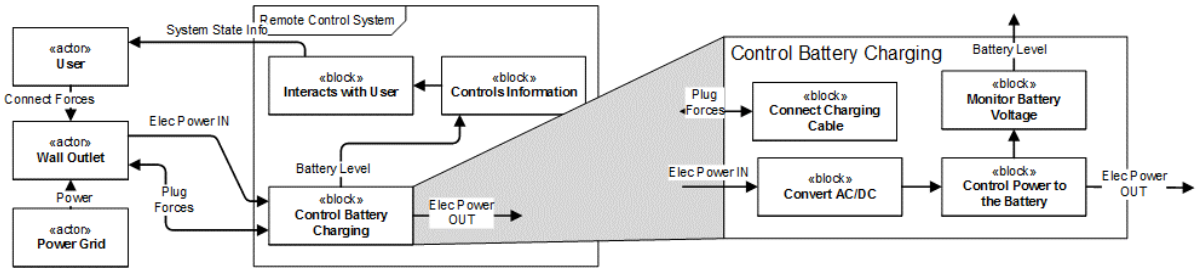
A set of binary rubrics similar to Table 3 were also developed for the Technical Requirements model. These rubrics help students to ensure that 1) each requirement contains at least one I/O, 2) each requirement has a measurable value, 3) each I/O is included in at least one requirement, 4) each interaction is described by at least one requirement, 5) each feature is covered by at least one requirement, and 6) each requirement has a verification measurement. The qualitative goal is to ensure that the requirements are sufficient to satisfy the feature set and that each requirement is achievable.

Function Decomposition and Synthesis of a Physical Architecture

The high-level system models that have been described thus far are useful to gain a general understanding of the system and to ensure that the system description adequately addresses the stakeholder features. However, if the system is to be implemented, a physical implementation of the system must eventually be created. If the system is complicated enough, the high-level views may not have sufficient detail for the students to understand how to implement each system function. It may be necessary to first *decompose* (divide into smaller sub-systems) the high-level Functional Architecture into more basic functions while also including more detail in each of the I/Os between actors and function blocks.

An illustration of a decomposition based on the remote control example is provided in Figure 8 for the "Control Battery Charging" function. There are many significant behaviors that are modeled by this function such that it would be difficult to produce a physical implementation

without explicitly describing these more basic functions. The emphasis to the students is to ensure that the decomposed functions are still equivalent to the higher-level functional block. One way to do this is to ensure that each I/O of the higher level description is accounted for in the decomposed version.



Interaction	Block	ID	Requirement	Feature(s)
Recharge the battery	Wall Outlet Control Battery Charging	RBD-1	The Convert AC to DC function must *accept* Electrical Power In at a [Wall Voltage of 120VRMS] and *produce* DC Volts at a [Supply Voltage of 3VDC].	Rechargeable

Figure 8. Examples of a function decomposition and technical requirement decomposition for the remote control example.

One of the important questions that arises when doing the decomposition is the level of detail to which a function should be decomposed. While there is not a single answer to this question, any effective answer is going to depend on the particular problem and the experience of the engineers trying to solve it. Instead of giving students well-defined boundaries, we present more general guidelines and questions for the students to think about.

1. **Can you imagine a way to physically implement a functional block?** If not, then the block needs to be decomposed further in order to gain that understanding.
2. **Does the name of the functional block uniquely identify each behavior performed by that block?** If not, then the block may need to be decomposed further to illustrate those behaviors.
3. **Are all of the *I/Os* uniquely identified in the *Functional Architecture*?** If not, then further decomposition may be necessary in order to account for them.
4. **What if there are multiple ways to decompose a *Function*?** There is always more than one way, the key is to evaluate each approach with respect to the *Stakeholder Features*.
5. **What if you don't know enough about a *Function* to decompose it?** In order to have confidence in a solution, it is effective to know how to decompose a *Function* at least one level further than is necessary for physical implementation. If that knowledge is missing, the students should take the appropriate measures to learn it.
6. **It may take an iterative approach.** The students may go too far or not far enough for the first draft, but continue to revise it with improved experience.
7. **Rely on guidance from a more experienced engineer.** Effective decomposition is a skill that gets better with practice, thus the students should seek focused advice and feedback from those with more experience.

Once the Functional Architecture has been adequately decomposed, the students can synthesize the physical architecture by mapping functions to different physical components. This process is typically what comes to mind when students think of "engineering design", and is usually

where they begin when given an open-ended problem to solve. However, with the systems approach, the students begin to see that the physical design is only one view of the solution space. If they are ignorant of the other views, there is no way to evaluate the effectiveness of the solution or to know if a more optimal solution is available. In the process of synthesizing the physical architecture the students are able to explore different implementation options. Depending on the type of problem being solved and the time available, they can be explicitly asked to evaluate different solutions.

The technical requirements can also be decomposed along with the functions. An example of a decomposed requirement is also shown in Figure 8. At this point some of the details may not yet be known, such as the actual quantitative value of the [Supply Voltage] attribute. This is the reason for naming the quantitative values with attributes. When the physical design is completed the quantitative values can be filled in.

The rubrics that were presented for the Functional Architecture and the technical requirements are also applicable to these steps because system decomposition follows the same process as these other steps.

Conclusions

The authors have developed a framework for introducing systems concepts to undergraduate engineering students. What the authors have found most useful about this approach is the limitations imposed on the concepts and models that help students to organize their thoughts and efforts. For example, naming functions with a verb followed by a noun phrase is helpful to get students to think about high-level behaviors and explore system-level tradeoffs before immediately developing a poor quality physical implementation. The rubrics for each model also help students to produce much better quality models. The independence of the models on the particular problem being solved makes it much easier for faculty to assess a set of vastly different projects because the physical structure of the views that the students produce is the same. The problem-independence also makes it easier for faculty to teach a common approach to problem solving (Simoni 2014, 2015a, 2015b).

The authors have found it to be overwhelming for students to be exposed to all of these models at one time. A scenario that seems to be more effective is to introduce students to the system modeling concepts using familiar systems, hence the remote control example. In order for students to understand the usefulness of a systems approach, though, it is important for them to apply the concepts on an open-ended unfamiliar project of increased complexity. The models presented in this paper have been used in senior design, freshman design, Engineering Grand Challenges projects, and systems engineering courses. In design courses in which the students use the full suite of these models, the focus is on continual iteration of the model to achieve greater detail within each model and consistency between the models. The students receive feedback and are allowed to revise based on that feedback. The final version is only collected and analyzed after many cycles of guided revision. Interim assessment of student work is based on how well they respond to the feedback and the quality and timeliness of the revisions they make. This system modeling process has improved both the quality of the results of the design projects and the students' understanding of their projects.

The authors are also developing workshops to inform other faculty of this definition of a systems perspective and how to use the different systems model (Simoni 2015a, 2015b). One workshop was given at the annual ASEE meeting in 2015 to approximately 40 participants and was very well received. Another workshop was provided at the authors' host institution to 13 faculty from nine different departments including Math and Physics. Some of the ideas to come

out of that local workshop were to use some of the system models to help students solve word problems in mathematics courses and to introduce circuits as behavioral functions in addition to studying their electrical properties. One faculty member who attended introduced the models to his students in the Principles of Optics course, in which he asked the students to address a NASA Challenge problem as the final project. Another idea to come out of the workshops is that all the models do not have to be introduced at the same time. By introducing only what is necessary to help understand a particular topic, students will see the benefit of using a systems perspective.

References

Baldwin, Kristen (2014), Expanding Systems Awareness: Our Role In Introducing Systems Engineering To All Future Engineers. *INSIGHT*: 17(3), 15–16.

Blanchard, Benjamin S, and W. J. (Wolter J.) Fabrycky. (2006). *Systems Engineering And Analysis*. Upper Saddle River, N.J: Pearson.

Buede, Dennis M. (2000). *The Engineering Design of Systems*. New York: Wiley.

Haskins, C., ed. (2010). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Version 3.2. Revised by M. Krueger, D. Walden, and R. D. Hamelin. San Diego, CA (US): INCOSE.

Hatley, Derek J, Peter Hruschka, and Imtiaz A Pirbhai. (2000). *Process For System Architecture And Requirements Engineering*. New York: Dorset House Pub.

Schindel, W. (2005a). "Requirements Statements Are Transfer Functions: An Insight From Model-Based Systems Engineering, *Proceedings of INCOSE 2005 Symposium*.

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

-----. (2011a). "What Is the Smallest Model of a System? *Proceedings of INCOSE 2011 Symposium*.

-----. (2012c). "Integrating Materials, Process & Product Portfolios: Lessons From Pattern-Based Systems Engineering", *Proc. of the Society for the Advancement of Materials and Process Engineering (SAMPE)*, Baltimore.

Schindel, W., and Peterson, T. (2012. 2013). Tutorial--"Introduction to Pattern-Based Systems Engineering (PBSE): Leveraging MBSE Techniques", *Proceedings of the INCOSE 2013 Symposium*, June, 2013.

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.

Schindel, W., Peffers, S., Hanson, J., Ahmed, J., Kline, W. (2011). "All Innovation Is Innovation of Systems: An Integrated 3-D Model of Innovation Competencies", *Proceedings of the 2011 Conference of the American Society for Engineering Education (ASEE)*.

Simoni, M., & Schindel, W. D., & Mu, X., & Moore, D., & Padgett, W. T. (2014), "Practicing and Assessing Formal Systems Competencies in ECE Senior Design" *Paper presented at 2014 ASEE Annual Conference,* Indianapolis, Indiana. https://peer.asee.org/22923.

M. Simoni, B. Kline, E. Andrijcic, S. Kirkpatrick, A. Bernal, D. Moore (2015a). "Helping Students Achieve a Systems Perspective of Engineering." *Presented at 2015 ASEE Annual Conference, Seattle, Washington*.

M. Simoni, B. Kline, E. Andrijcic, S. Kirkpatrick, A. Bernal, D. Moore (2015b). "Helping Students Achieve a Systems Perspective of Engineering." Summer workshop at Rose-Hulman Institute of Technology. Terre Haute, IN. August 17-18, 2015.

Biographies

Mario Simoni is an Associate Professor of Electrical and Computer Engineering at Rose-Hulman Institute of Technology. He earned his Ph. D. degree from the Georgia Institute of Technology in 2002. He has held temporary positions performing integrated circuit design at INTEL and Texas Instruments. Currently, his research interests are focused on 1) designing an application specific computer processor for rapid and scalable biologically-accurate simulations of neural systems; 2) improving learning in introductory Fourier theory courses; 3) encouraging the study of electromagnetics in high-school physics education; and 5) improving learning and student performance in engineering design education.

Eva Andrijcic received her PhD and MS in Systems and Information Engineering from University of Virginia, where she worked at the Center for Risk Management of Engineering Systems. She received a BS in mathematics from Randolph-Macon Woman's College. Her major interests are in the areas of risk analysis and management, critical infrastructure management and protection, interdisciplinary engineering education, and risk education. She has worked on several large research projects, including projects for the Department of Homeland Security and the Mitre Corporation. Eva has several publications in the area of risk analysis and systems engineering, and she is a member of the Society for Risk Analysis, American Society for Engineering Education, and International Council on Systems Engineering.

Bill Kline is Professor of Engineering Management and Associate Dean of Innovation at Rose-Hulman. He joined Rose-Hulman in 2001 and his teaching and professional interests include systems engineering, quality, manufacturing systems, innovation, and entrepreneurship. He is currently a partner in Inside Out Innovations, a consulting venture focused on innovation tools and systems. Prior to joining Rose-Hulman, he was a company co-founder and Chief Operating Officer of Montronix, a company in the global machine monitoring industry. Bill is a graduate of the University of Illinois at Urbana Champaign where he received a Ph.D. degree in Mechanical Engineering.

Ashley Bernal received her PhD from Georgia Institute of Technology in 2011. She currently is an Assistant Professor of Mechanical Engineering at Rose-Hulman Institute of Technology, USA. She was an American Society of Mechanical Engineers (ASME) teaching fellow and a Student Teaching Enhancement Partnership (STEP) fellow. Prior to receiving her PhD, she worked as a subsystems engineer at Boeing on the Joint Unmanned Combat Air Systems (JUCAS) program. Her research areas of interest include piezoelectrics, nanomanufacturing, and optical measuring techniques.