# The Component Challenge in Undergraduate Engineering

**Michael VanHilst, PhD**
Florida Atlantic University, Boca Raton, Florida, USA, vanhilst@fau.edu

## Abstract
More and more, engineering systems are built by combining sophisticated off-the-shelf components. This method of construction confronts today's engineers with new challenges that cannot be addressed solely by the application of first principles. Rather than consult their textbooks, students in senior level engineering design courses find themselves pouring over tech sheets, searching the web, and telephoning customer support in the effort to make their project work. In this paper, we describe our experience with senior level design for electrical and computer engineers and our effort to teach skills to confront this real-world challenge.

## Keywords
Engineering Education, Components, Systems of Systems

## 1. Introduction

In the College of Engineering we seek to prepare our students with the knowledge and skills they need to successfully handle engineering jobs in the real world. In the final year of their studies, engineering students perform a capstone project, representing the culminating experience of the engineering program. To carry out this project, students demonstrate their ability to apply the engineering knowledge and skills they have acquired in three and a half years of education, to plan, design, and complete a real world project. Or so the theory goes.

For the capstone project, each team of students chooses a real world problem they wish to solve, and proposes a solution which will become their project for the second semester. The solution must be a system – a set of components working together, where every component is a fairly well understood device. Possible components include mechanical devices or assemblies, sensors, actuators, processors, circuits, or software. The students choose the technology to apply to each component, develop a plan, build the system, test it, and present it at the end of the second semester. Along the way, the students refine and improve each component to best fit the needs of the desired system.

Our project model works well when each component corresponds to a type of component encountered by the students in prior course work, and described in their text books. But, in today's world, fewer and fewer systems are being built purely from discrete components. Increasingly, engineers include commercial off-the-shelf (COTS) subsystem components, or modules, in the design of their systems. The resulting systems offer greater functionality at lower cost. It is not uncommon for students to find inexpensive modules, too, to replace significant parts of their project. In fact, when it reduces cost, we encourage it. Using COTS subsystem modules allows students to undertake projects that are more

ambitious.[1] However, for the students, working with COTS modules poses a new set of challenges unlike anything we teach them in the classroom.

Consider a project to build a smart pill dispenser for an elderly person with failing memory. The challenges cover the three disciplines: mechanical engineering to dispense and present the correct number and types of pills regardless of size or shape, and to encase the entire system in a robust package, electrical engineering to detect that the patient has removed the pills and when the device is running low on pills, and to power the device, and computer engineering to create an effective interface between the device and the user, and to notify a caregiver when assistance when warranted. The first time a team undertook this project, they addressed the computer engineering challenges with a microcontroller kit, a collection of button switches for input, an LCD panel to print messages, and an auto-dialer to contact the care provider and play a recorded message. The price for these components came to $800.

A recent team undertaking a similar project chose a different strategy to address the computer engineering challenge. In place of the processor, buttons, and LCD panel, they used a low end Palm PDA. The PDA costs half as much as the components it replaces, and includes a touch screen and other capabilities not available in the original design. In place of the auto-dialer, they equipped their PDA with a Bluetooth interface to communicate with, and through a similarly equipped cell phone. Through Bluetooth they can remind the user, contact the care giver, and become a subsystem in a larger health maintenance program.

The new project design looks good on paper, but it poses a new kind of challenge to implement. As a simple example, suppose a student writes code to send a message from the PDA to the cell phone and corresponding code on the cell phone to respond. When the student attempts to run the code, nothing observable happens. What should our novice engineer do next? What strategy should be applied? What skills acquired in other courses can be applied to this problem? The problem is not seen as the behavior of a low level component or programming construct. The problem occurs in the interaction between subsystems. Furthermore, because the subsystems are COTS components, the problem cannot be reduced to the behavior of low level concepts through subsystem decomposition and inspection. The COTS subsystem must be treated as a black box, with only the possibility to observe inputs and outputs.

## 2. Basic Problem Solving

In the field of Computer Science, the problem of subsystem component integration is an acknowledged issue in the domains of both software reuse and component-based software engineering (CBSE). Many a senior project student struggling with a COTS component for a seemingly simple task, would reiterate the comment made by Arrango at the 1997 ACM Symposium on Software Reusability (SSR), “when you want a banana, it comes with an 800 pound gorilla attached.” Garlan and … much cited paper on Component Mismatch describes numerous problems they encountered in creating an application from large scale subsystem applications, most notably due to incompatible models of control. Sullivan described using a formal analysis of underlying component models to identify some of the problems.

The literature on “engineering problem solving” commonly describes a process of developing and refining models of the problem witht the goal of reduing the problem to a tractible set of problems that align themselves with basic engineering concepts. The engineer would then apply the known models and formulas to arrive at a solution. There are important skills, but they fall short when applied to the problem of complex systems of black box subsystems. To help our students deal with such complex problems, we have fallen back to teaching basic problem solving skills and practices.

In basic problem solving, students are advised to follow the following steps:

[1] We don’t allow students who use COTS modules to do less work than others.

1. **Take stock**: This is an important step for students and anyone new to a problem domain to avoid falling into common pitfalls. In an analogy of going into battle, this step might be called "*girding one's loins*." It's good advice when facing a tough exam question, too.
    a. Step back.
    b. Take a deep breath.
    c. Remind yourself of what you already know and what tools you can bring to the process.
2. **Define the problem**: Identify the gap between the current and the desired. In standard planning practice this step is called Gap Analysis. This step defines the problem, sets limits on its scope, and creates a framework for its resolution.
    a. What works?
    b. What do we want to have happen?
    c. What's missing?
3. **Narrow the problem**: Bound the problem to as small a set of components and issues as possible.
    a. Decompose the system conceptually into parts, steps, or functions.
    b. Find alternative ways to exercise the system so as to demonstrate whether or not each piece is working. A good way to rule out components is to find other software, perhaps even an unrelated application, that successfully exercises the component in question.
    c. Create the smallest, simplest test that clearly demonstrates the problem with the fewest dependencies on your context or setup.
4. **Consult available documents**:
    a. Identify simple terms related to the problem to use in searching an index or the Web.
    b. Read the manuals (RTFM) and component tech sheets.
    c. Search the Web
        i. Search vendors' customer support and developer support Web sites. Many manufacturers provide extensive resources including frequently asked questions (FAQ's) and troubleshooting tools. Some vendors also offer developer support sites with more resources.
        ii. Search discussion groups. The problem may have been discussed in an online discussion site. Google has a searchable newsgroup archive going back more than a decade. Searching the regular Web may reveal other discussion sites and Wiki sites.
5. **Diagnose the problem**:
    a. What could be wrong? Create a list of hypotheses about the possible cause of the problem. Focus on hypotheses that, if validated, would lead to a useful course of action to resolve the problem. A hypothesis that, even if validated, does not enable one to fix a problem is a useless hypothesis.
    b. What would we like to know? Create a list of useful questions, the answers to which could invalidate some of the hypotheses?
    c. What can we try? Define and list alternative experiments, measurements, or observations to answer one or more of the above questions. Order these experiments so as to perform the cheapest experiments first. Even if corresponding hypothesized cause seems unlikely, it is foolish to skip an inexpensive test and jump to one that is much more costly.
6. **Seek advice**:
    a. Post your problem to a discussion group.
    b. Consult local experts or others who have experience with the subsystem in question.
    c. Call customer support.

We require project teams to find an advisor, usually from within the faculty of the College of Engineering. However, teams often need the assistance of others, as well. It is not uncommon for a project team to face a situation where their project will not succeed without significant assistance from a vendor's tech support. We find the tech support staff from most vendors genuinely willing to help students with their problems. We tell the students that real companies are not so much interested in how

hard they work as in how quickly they can solve problems. Getting advice from others is often the quickest way to solve a problem. To keep relations with vendors on good terms, we give the following advice.

1. Be prepared: Make sure you have followed all of the previous steps before asking for assistance. People will be less helpful if they feel like they are doing your work.
    a. Have a simple example to describe that illustrates the problem.
    b. Have a set of specific questions that, if answered, will advance your pursuit of a solution.
    c. Know what you know and don't know so you can be clear about where you might be confused.
2. Be courteous and deferential:
    a. Ask for advice rather than help, or even advice on how to get advice. Even important people like to give advice if approached in the right way. Remember, call center staff are trained to put annoying callers on indefinite hold.
    b. Express gratitude. People are more likely to be helpful if they feel that their effort is genuinely appreciated.
    c. Be focused and objective. If you have doubts, pose a directed example in the form of, "How would this suggestion explain or solve X?"
3. Take notes:
    a. Write down the name of the person you are talking to.
    b. Take notes of any suggestions you receive – even if they don't make sense at the time.
    c. Ask if you can consult them again and take their phone number.

Some students are timid about seeking help and, without extra pressure and encouragement, would let a problem remain unresolved for weeks rather than consult a stranger. At the other extreme, student teams have overused customer support to the point of being black listed by a vendor's support center. Once when a student posted a description of his problem to an Internet newsgroup, he found his mailbox full of mail from students at other schools facing the same problem, but no solutions. For the most part students have to find the best way to get advice on their own. We view such experience as an important part of the capstone project. It offers a unique preparation for industry employment.

On several occasions students have taken the initiative to contact a local industry for assistance. We have been pleasantly surprised by how much assistance local industries voluntarily provide. Students are taken on tours of their production facilities, advised on tools, materials, sizes, and components, and given opportunities to consult with their engineers.

## 3. Common Pitfalls in Handling Problems

Over three years, we have observed a number of pitfalls that students commonly fall into. From our experience, we have created a list of these pitfalls in hopes that students will learn from the past and not repeat the them.

1. **Optimism or Assuming that everything (or anything) will work**: Perhaps the most common pitfall involves leaving important milestones for the last week of the project and building large parts of the project without testing for small milestones along the way. Important milestones can be as simple as turning the power on for the first time without smelling burnt plastic or producing a first detectable effect – e.g. changing a single bit or a voltage.
2. **Lack of confidence or Hopelessness**: This common pitfall leads one to assume from the beginning that the problem is caused by something beyond one's abilities to understand. Common symptoms are the attitude that there is nothing one can think through or test, and students trying things that don't make sense and are even dangerous. The best way to avoid this problem is to take stock before beginning to investigate the problem, and then try to map the problem to concepts one already knows.

3. **Fatalism or Assuming the worst**: This pitfall leads students to pursue costly and complicated solutions before ruling out the inexpensive and sometimes obvious solutions. Students have spent multiple days in the lab trying to fix a circuit without checking for a reversed connection, floating ground, or lack of continuity in wires and connectors. We remind students to test the things that are easiest to test first.
4. **Laziness or Blaming others**: This pitfall leads students to believe that someone or something else must be the source of the error. The basic symptom is waiting for someone else to fix or diagnose a problem. It is not uncommon to find students returning or exchanging hardware as defective, sometimes several times, before performing tests that would verify all that can be verified. Frequent progress reviews with careful inquiry are needed to resolve this tendancy.
5. **Recklessness or Assuming that trying to solve one problem will not cause another**: Students who fall into this pitfall can be found turning the voltage way up when no response is detected, and plugging and unplugging components while the power is on. It is important to establish procedures and limits before conducting tests.

There is a joke about a man who is looking for his keys under a streetlight where it is easier to see, even though he dropped it somewhere else. Students who suffer from hopelessness or fatalism seem to have the opposite tendency. They are looking for their keys in the darkest areas.

Students, and others, who are intellectually lazy, seem to prefer hypotheses that, if confirmed, lead to a conclusion that there is nothing they can do to resolve the problem. Teaching how to develop useful models and hypotheses goes a long way toward improving skill and confidence.

In the early stages of the projects, students often present problems as excuses for not having made progress. We teach them that it is better treat problems as challenges than as excuses. This shift in perception maintains a sense of momentum, even in the face of difficulty. In the review sessions, presenting a problem as a current challenge shifts the discussion away from the stressful discussion of whether or not the student should have done better, to a more positive discussion of how to handle the problem.

We list effectively working in teams as an objective of the senior project course. We should probably also include skills for effectively working with (or using) vendor support, experts, and non-traditional sources of information as an expected outcome of this course. These skills are really more social skills than engineering skills.

## 3. Pedagogy

From a pedagogical perspective, the problem we have described is called systems of systems. Large, complex systems are built of multiple smaller systems, collaborating to address one or more concerns. The Internet is a system of systems. Web applications are designed based on the principals of systems of systems. Web Services are meant to provide a rich set of building blocks that put new powerful applications within reach of web application designers of every stripe and means. Work on the Semantic Web, and standards in the areas of Ubiquitous Computing and Seamless Mobility are meant to make it possible to create rich collaborations among subsystem components on the fly.

The study of systems of systems falls within the field of systems engineering. But the issues involved are finding their way, with increasing frequency and importance, into all fields of engineering. In computer engineering, it is now expected that application development in the future, whether on the Web, among small ubiquitous devices, or even on executing on a single chip, will involve concurrent, distributed, largely autonomous processes interacted at a high level of abstraction [Su05].

Systems thinking as an engineering discipline has its origins in the late 1930's with the work of Ludwig von Bertalanffy (von Bertalanffy 1975). Von Bertallanfy applied the insights he had learned from the study of biological systems to systems in general. In the late 1940's Norbert Wiener, having worked on predictive servomechanical anti-aircraft guns with self-correcting feedback behavior, developed a disclipline of communications and engineering modeling called Cybernetics (Wiener 1948). In servo control, The error between desired motion and the actual motion is used to choose the input to make the error smaller. Cybernetics had a colorful history, influenced by politics and optimistic predictions about robots and computers. It has lately been somewhat revived by a movement to introduce systems and feedback models in the K-12 curriculum (Ossimitz 2000)(Bernstein 2003).

In the 1960's Jay Forrester, another MIT professor, applied the systems approach to building dynamic models of complex social systems, forming a disclipline called System Dyamics (Forrester 1968)(Forrester 1969). Forrester's work had a profound impact on the new field of ecology. Within MIT, Forrester's work influenced the undergraduate curriculum in the 1970's and continues as a factor in their introductory engineering courses. Outside MIT, Forrester's work has had more impact on the social sciences.

Work on systems modeling as a tool for problem solving in complex systems evolved largely in the social and biological sciences. Don Schön, an Urban Planning professor at MIT, was inspired by Forrester's modeling work to developed a discipline of reflection and meta-modeling[2] (Schön 1983)(Schön 1987). In reflection, the process of modeling itself becomes a subject of study, and we come to understand how our view of a problem, and its possible resolutions, is affected by the kinds of models we construct. It is this kind of thinking that we try teach our students in Engineering Design to help them avoid the pitfalls described above.).

**4. Related Work**

The NSF-ITR project on Foundations of Hybrid and Embedded Software Systems supports research at a number of universities working to revise the engineering curriculum to better address hybrid systems. Hybrid systems combine both discrete event and continuous time models and phenomena in a single system. The project participants from Vanderbilt University teach a course on building robust large scale distributed realtime embedded (DRE) systems using component technology. The course, however, appears to focus on modeling and the use of QoS enabled component middleware, and may thus avoid issues posed by Black Box COTS components. The instructional theories they have adopted, of just-in-time (JIT) learning and anchored inquiry, seem well suited to the issues described here (C&T Group 1993).

The Dartmouth/Thayer approach to engineering problem solving is a "framework for bringing problems of the real world into the classroom." It's author's claim that it is creative problem solving suitable for social and complex problems. The approach is an iterative search for a viable solution that involves carefully defining and redefining the problem "to remove bias", and brainstorming to think of alternative solutions. Removing bias from the definition of the problem, and considering alternative solutions, are logical bits of advice for any kind of problem solving. But the guidelines do not offer enough specific advice, let alone a model, for solving system of systems problems.

Recent workshops on Debugging and Testing Parallel and Distributed Systems might reveal ideas for our poor student faced with the non-functional interface between Bluetooth subsystems. However, a review of the programs for the two workshops held so far reveals a narrow focus on classic timing and concurrency issues (PADTAD 2004).

[2] Meta-modeling is itself now an important discipline in Computer Science.

## 5. Conclusion

Over several iterations of the interdisciplinary Engineering Design senior project course, we have observed students increasing facing a class of problems posed by the use of subsystem modules or components. Standard engineering problem solving disdciplines do not seem well suited to tackling these problems. Instead, we have fallen back on teaching our students basic problem solving skills. We have also turned to the systems thinking discipline of problem solving for complex systems as taught in the social sciences.

Systems thinking and related problem solving theories are important for advancing the field of engineering particularly as it applies to systems and components. We find that they help the students to better handle many of the kinds of challenges they face in the interdisciplinary engineering senior project classes. It would probably help the students even more if, as at MIT, we threaded systems thinking a little more into other parts of the curriculum. Even so, the poor student with Bluetooth code and no observable response still faces a daunting challenge. In the future we will be looking more into problem solving pedagogies for additional and perhaps better ways to address the component challenge.

## References


Bernstein, D.W. (2003) "Introducing Signals, Systems, and Control in K-12" IEEE Control Systems Magazine, April 2003

Cognition & Technology Group at Vanderbilt (1993) "Anchored Instruction and Situated Cognition Revisited," Educational Technology, 33(3), 52-70, March 1993.

Forrester, J.W. (1968) *Principles of Systems*, 2nd Edition, Pegasus Communications, 1968

Forrester, J.W. (1969) *Urban Dynamics*, 2nd Edition, Pegasus Communications, 1969

Ossimitz, G. (2000) "Teaching System Dynamics and Systems Thinking in Austria and Germany" The 18th International Conference of the System Dynamics Society (System Dynamics 2000)

Parallel and Distributed Systems: Testing and Debugging (PADTAD), (2004) http://www.haifa.il.ibm.com/Workshops/TestingAndDebugging/index.html, 2/12/2004

Schön, D.A. (1983) *The Reflective Practitioner How Professionals Think in Action*. Basic Books, 1983

Schön, D.A. (1987) *Educating the Reflective Practitioner*. Jossey-Bass

von Bertalanffy, L. (1976) *General System Theory; Foundations, Development, Applications*. (Revised Edition). George Braziller

Weiner, N. (1965) *Cybernetics: or the Control and Communication in the Animal and the Machine*. 2nd Edition, The MIT Press


## Biographic Information


Dr. Michael VANHILST. Dr. VanHilst is an assistant professor at Florida Atlantic University. He received the BS and Master of City Planning degrees from MIT in 1970's, and MS and PhD degrees in Computer Science from the University of Washington in the 1990's. He is a member of the Secure Systems Research Group at Florida Atlantic University (http://www.cse.fau.edu/~security).