

# CHAPTER 1: INTRODUCTION TO SOFTWARE ENGINEERING DESIGN

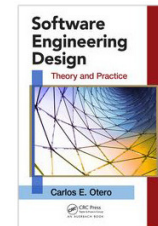
## SESSION I: MOTIVATION AND GENERAL DESIGN CONCEPTS

### *Software Engineering Design: Theory and Practice*

by Carlos E. Otero

Slides copyright © 2012 by Carlos E. Otero

*For non-profit educational use only*



May be reproduced only for student use when used in conjunction with *Software Engineering Design: Theory and Practice*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information must appear if these slides are posted on a website for student use.

## SESSION'S AGENDA

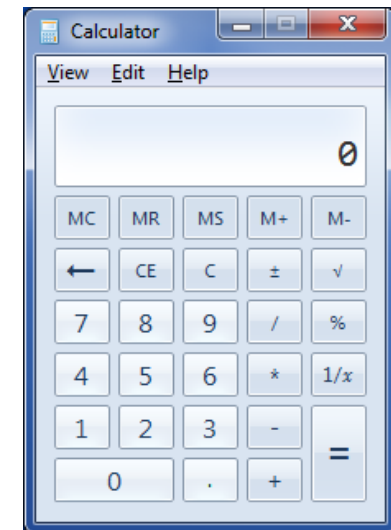
- What's it all about?
  - ✓ Motivation for software engineering design
  
- Engineering Design
  - ✓ Let's revisit the software engineering life-cycle
  - ✓ Context of software design
  
- Problem-solving
  - ✓ Problem-solving process
  - ✓ Types of problems
  - ✓ Types of thinking
  - ✓ Types of solution approaches

## MOTIVATION FOR SOFTWARE ENGINEERING DESIGN

- Let's go straight to the point, what is software engineering?
  - ✓ According to the IEEE [1], Software Engineering is:
    - (1) the application of a *systematic, disciplined, and quantifiable* approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
    - (2) The study of approaches as in (1)
- What do we mean by *systematic*? According to an online dictionary [2]:
  - ✓ Presented or formulated as a coherent body of ideas or principles
  - ✓ Methodical in procedure or plan
- What do we mean by *disciplined*? (From the word discipline [2]):
  - ✓ A rule or system of rules governing conduct or activity
- What do we mean by *quantifiable*? (From [2])
  - ✓ To determine, express, or measure the quantity of
- Given clear definition of these important terms, let's ask ourselves, *do we really need a systematic, disciplined, and quantifiable approach to developing software?* Let's look at some examples...

# MOTIVATION FOR SOFTWARE ENGINEERING DESIGN

- A software example: *The Mighty Calculator*
  - ✓ (1) Do we need software engineering to develop this product?
    - Do we need a systematic approach?
    - Do we need a disciplined approach?
    - Do we need a quantifiable approach?
  
- Stop and think about these questions! You may be wondering, well..., it depends!
  - ✓ (2) Is this my own pet project?
  - ✓ (3) Is somebody paying me for this product?
  - ✓ (4) Will other people use this product?
  - ✓ (5) Is there a time-frame for this product's development?
  - ✓ (6) Do I need to quantify the approach used?
  
- Depending on (2), (3), (4), (5), and (6) the answer to (1) may be: **yes, no, or maybe!!**
  
- Let's look at another example...



## MOTIVATION FOR SOFTWARE ENGINEERING DESIGN

- Without software, the Chevy Volt is Stuck in Neutral. According to GM [3],
  - ✓ “...the Chevy Volt is as much a software engineering accomplishment as it was a mechanical engineering challenge...”
  - ✓ “...the **software coordinates the flow of energy** and **provides drivers feedback**... Drivers can, for example, view charge status and **schedule battery charging from a smartphone**...”
  - ✓ “**Writing code is not really the challenge any more**; it’s managing the thousands of moving pieces in a project and using that material, such as requirements, code, and models, for future projects”
  - ✓ “We’re now at the point where software control strategies and controls are the gating factor of the vehicles we make”
- Do we need software engineering here?
  - ✓ Let’s put things into perspective to (hopefully) understand what it takes to develop this product (see video on next slide...)



## MOTIVATION FOR SOFTWARE ENGINEERING DESIGN

- Without software, the **Chevy Volt** is Stuck in Neutral (cont) [4]...

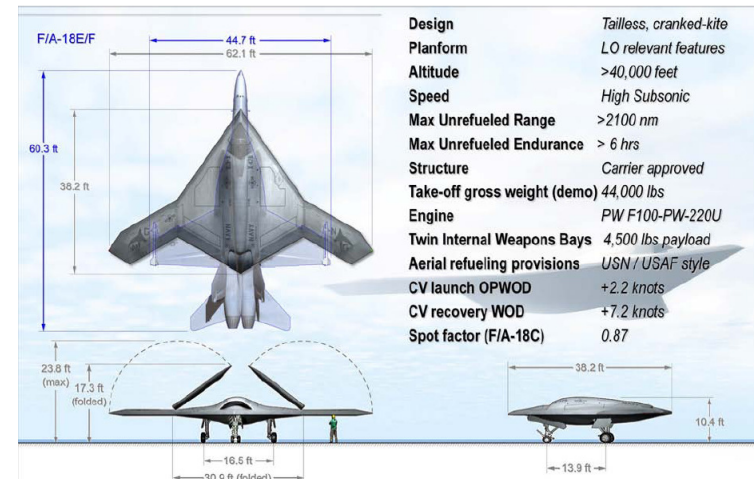
- If embedded youtube control fails, use the browser to navigate to:
  - <http://www.youtube.com/watch?v=CjjASGV36mw>

- Let's look at one final example...

# MOTIVATION FOR SOFTWARE ENGINEERING DESIGN

- The X-47B is a smart, autonomous, computer-controlled unmanned aircraft that takes off, flies a **software-controlled mission**, then returns to base in response to mouse clicks from its mission operator. The **mission operator** monitors the X47B air vehicle's operation, but **does not actively "fly" it via remote control as is the case for other unmanned systems** currently in operation [5].

If embedded youtube control fails, use the browser to navigate to:  
<http://www.youtube.com/v/kBrVRTVNph0>



- Do we need software engineering here?
  - ✓ Ok, it's becoming evident that not all software systems are equal!
  - ✓ Some are more complex than others and software failure may result in catastrophic events, including loss of human life!
- From this point forward, **our discussion will focus on these types of large-scale software systems**, as opposed to systems similar to the Mighty Calculator...

# ENGINEERING SOFTWARE

- Hopefully, by now, you are convinced that a *systematic, disciplined, and quantifiable* approach is needed to build certain types of software systems; that is, software engineering is necessary to build some (if not all) software products. *But how do we engineer software?*
- Luckily, a formal process for engineering software has been known for quite some time. This process supports a *systematic, disciplined, and quantifiable* approach to software product development, and includes the following phases:

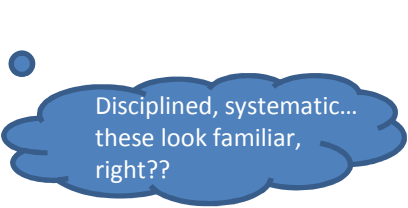
Phase	Description
Requirements	Initial stage in the software development life-cycle where requirements are elicited, analyzed, specified, and validated.
Design	The requirement's specification is used to create the software design, which includes its architecture and detailed design.
Construction	Relies on the requirements' specification, the software architecture, and detailed design to implement the solution using a programming language. A great deal of design can also occur at this phase.
Test	Ensures that the software behaves correctly and that it meets the specified requirements.
Maintenance	Modifies software after delivery to correct faults, improve performance, or adapt it for a different environment.

- Of **importance** to this course is the *design phase*, where requirements are used to create a blueprint of the software to be constructed. Before delving deep into software engineering design, let's look at the general concept of engineering design...



## ENGINEERING DESIGN

- Design is not a new concept conceived by software engineers. Design is used in all other engineering disciplines, e.g., electrical, mechanical, civil, etc. Dym and Little define engineering design as [6]:
  - ✓ A systematic, **intelligent process** in which designers **generate, evaluate and specify designs** for devices, systems or processes whose form(s) and function(s) **achieve clients' objectives and users' needs** while satisfying a specified set of **constraints**.
- Design is a lengthy and complex process which requires significant investment in time and effort. So, *why conduct design in engineering disciplines?*
  - ✓ Using commonsense, complex products (e.g., bridges, airplanes, watercrafts, medical devices, safety-critical software systems, etc.) are hard to create, costly to change, and when built carelessly, can harm human life.
  - ✓ Design allows teams to organize in **disciplined** manner and provides a **systematic** approach to carefully ensure that products are built to meet their specification.
    - Remember the definition of software engineering??



Disciplined, systematic...  
these look familiar,  
right??

## ENGINEERING PROBLEM SOLVING

- In the previous slide, engineering design was defined as an intelligent process... what do we mean by intelligent process?
  - ✓ We refer to it as intelligent process because a great deal of problem-solving occurs during design.
  - ✓ During the design process, engineers are constantly engaging in problem-solving activities.
    - Their work requires them to identify, evaluate, and proposed solutions to complex problems.
    - Some problems encountered by engineers have never been solved before!
  
- Because of the large number of problem-solving activities present during design, a formal discussion on problem-solving is required.
  - ✓ To become a good designer, engineers must be good problem solvers.
  - ✓ To become a good problem solver... well, that requires lots of time and effort, but we can at least set up a framework for solving problems throughout the design phase. Let's examine the fundamentals of problem solving...

# ENGINEERING PROBLEM SOLVING

- In a general sense, problem-solving during design occurs in three different states:

- ✓ Initial State
- ✓ Operational State
- ✓ Goal State

- Initial State

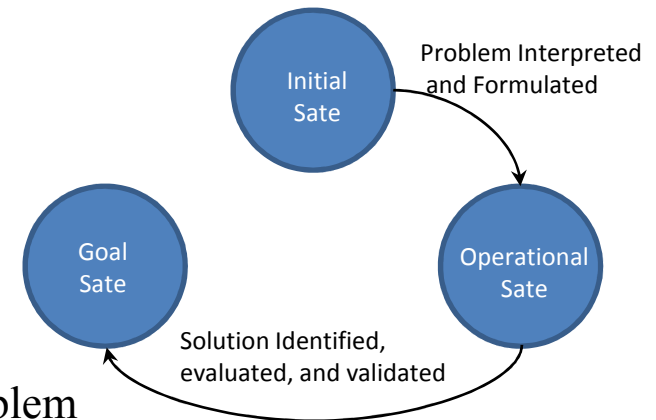
- ✓ Problems are formulated and interpreted
- ✓ In some cases, understanding the problem is a problem itself.

- Operational State

- ✓ Once problem is understood, operational state begins
- ✓ Thinking about the problem and viable solutions come to light

- Goal State

- ✓ Once an appropriate solution is identified, evaluated, and validated, goal state is reached
- ✓ Final solution found, marking the end of the problem-solving process



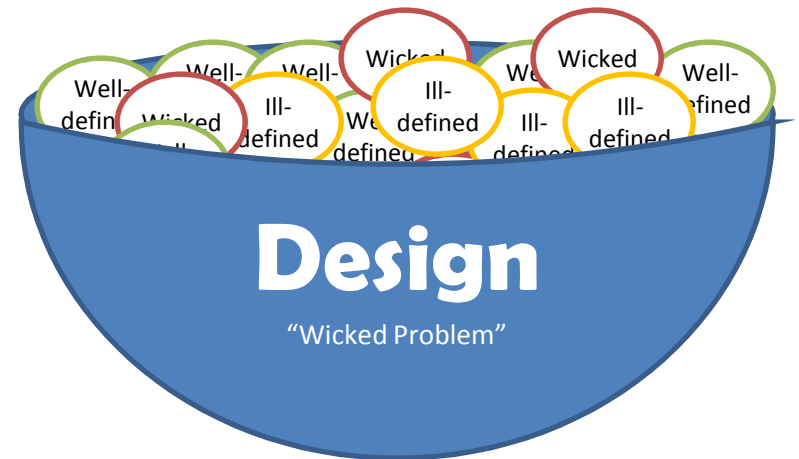
## ENGINEERING PROBLEM SOLVING – INITIAL STATE

- Moving from state-to-state (i.e., initial→operational→goal) is not as simple as it sounds. During the Initial State of problem solving, it becomes evident that not all problems are the same; they vary in size, complexity, and the amount of time required to reach the goal state. Problems can be classified as:
  - ✓ Well-defined
  - ✓ Ill-defined
  - ✓ Wicked Problem
  
- *Well-defined* problems have clear defined goals and their constraints are well understood.
  - ✓ This makes scoping the problem, proposing a solution, and arriving at a the solution easier than with other types of problems.
  
- *Ill-defined* problems are ambiguous with undefined goals.
  - ✓ They require more time and effort to clarify and interpret the problem
  - ✓ Sometimes, these problems can be transformed into well-defined problems.
  - ✓ Once formulated, we can move on to propose and arrive at a solution.

## ENGINEERING PROBLEM SOLVING – INITIAL STATE

- *Wicked-problems* are problems where no single formulation exists.
  - ✓ Many acceptable formulations can take place with no definite solution
  - ✓ Solutions are not deemed correct or incorrect, but good or bad
  - ✓ They require more time and effort to propose and arrive at “some” solution
  - ✓ The problem is formulated after it has been solved!

- Example of wicked problems include:
  - ✓ Healthcare
  - ✓ Global climate change
  - ✓ Border security
  - ✓ Design, more to the point, software design
    - Design a secure system
    - Design a usable system
    - Design a maintainable system



- Design is a wicked problem, and during the design process, many well-defined, ill-defined, and wicked sub-problems are encountered!

## ENGINEERING PROBLEM SOLVING – INITIAL STATE

- Let's use a bird's eye view and tackle *Design* as a problem. During the *Initial State*, we classify it as a *Wicked Problem*. Why does this make sense?
  - ✓ Consider the following task: *Develop a secure software system*.
    - What is the goal?
      - *That's easy, to design and develop a secure system!*
    - What is the problem? More specifically, can we formulate this problem in a way that can be solved?
      - Stop and think about this!
  - ✓ Consider the following task: *Develop a usable system*.
    - What is the goal?
      - *Easy again, to design and develop an "easy to use" system.*
    - Can we formulate this problem in a way that can be solved?
      - *We can certainly try, but you will find that most of the time, the problem can only be formulated after many iterations between customer and designer and only after an acceptable solution has been found!*
- Designers spend a great deal of time formulating acceptable versions of these wicked problems! That is, acceptable to stakeholders, including customers, managers, etc.
  - ✓ For example, an attempt to problem formulation for developing a secure software system may include creating an authentication (e.g., username and password) mechanism. Although this may be an acceptable problem formulation for the project's stakeholders, it clearly does not solve the security problem!

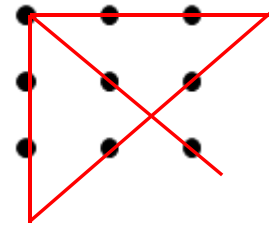
## ENGINEERING PROBLEM SOLVING – OPERATIONAL STATE

- During the Operational State of problem solving, it becomes evident that not all problems can be tackled the same way. Different techniques for problem-solving can be employed, including:
  - ✓ Divide and conquer
  - ✓ Reusing solutions, e.g., patterns.
- Many times, problem-solving requires a “think outside the box” mentality. What exactly does this mean?
  - ✓ In a nutshell, it means that we need to think in ways that deviate from conventional wisdom.
- For example, try solving the popular nine-dot puzzle problem using the figure to the right and the following requirements:
  - ✓ Draw four straight lines to connect all dots.
  - ✓ The pencil cannot be lifted from the paper once the line-drawing process begins.
  - ✓ No lines can be retraced.
  - ✓ **A line must pass through every dot.**



## ENGINEERING PROBLEM SOLVING – OPERATIONAL STATE

- During the Operational State of problem solving, it is natural to perceive and approach problems in certain ways, based on our current knowledge.
  - ✓ For example, the nine-dot problem resembles a square, therefore it is hard for some people to operate outside of the assumption that lines should begin and end on a dot.
  - ✓ This *functional fixedness* limits the ability to find solutions based on objects having a different function from their usual ones.



- To increase the chance of overcoming *functional fixedness*, problems need to be attempted several times and considered from many different *viewpoints* and unusual angles.
  - ✓ Don't forget about this... this advice will come in handy during the software architecture activity!



## **ENGINEERING PROBLEM SOLVING – OPERATIONAL STATE**

- During the Operational State of problem solving, different types of thinking takes place.
  - ✓ Convergent thinking
  - ✓ Divergent thinking
  
- Convergent thinking
  - ✓ Type of thinking that seeks to find one single solution to a problem, e.g., a math problem.
  
- Divergent thinking
  - ✓ Type of thinking that seeks to find multiple solutions to a problem.
  - ✓ This type of thinking is associated with originality, inventiveness, and flexibility.

## ENGINEERING PROBLEM SOLVING – OPERATIONAL STATE

- During the Operational State of problem solving, different problem-solving methods take place.
  - ✓ Algorithms
  - ✓ Heuristics
  
- Algorithm
  - ✓ Step-by-step procedure for finding the correct solution to a given problem.
  - ✓ Do not normally involve subjective decisions or rely on intuition or creativity to find solutions.
  - ✓ Examples:
    - Bubble sort, Merge sort, etc.
  
- Heuristic
  - ✓ Rules of thumb (or procedure) that may or may not lead to the solution of a problem.
  - ✓ In some cases, they can lead to optimal solutions, in others, they can lead to solutions that are far from optimal, or no solution at all!
  - ✓ Heuristics are advantageous when tackling tough problems where an efficient algorithm may not exist.

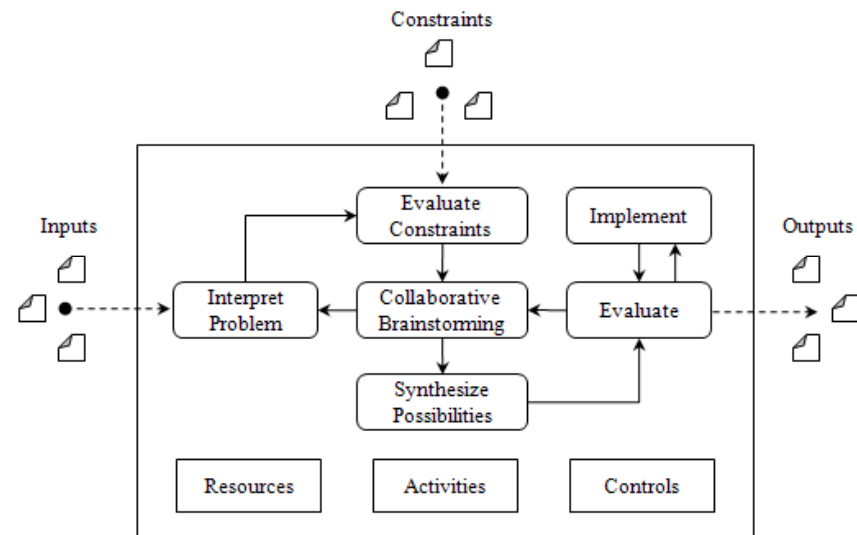
# ENGINEERING PROBLEM SOLVING – HOLISTIC APPROACH

- We can use everything discussed so far to come up with a holistic problem-solving framework that can help us tackle problems of any size during the design process. The framework consists of the following tasks:

- ✓ Interpreting the problem
- ✓ Evaluating the constraints
- ✓ Collaborative brainstorming
- ✓ Synthesizing the possibilities
- ✓ Evaluating the solution
- ✓ Implementing the solution

- In addition, we must consider:

- ✓ Resources
  - Means by which activities are performed, e.g., people, software, hardware.
- ✓ Activities
  - Internal tasks determined by the development organization that must be followed when solving problems, e.g., architecture, detailed design, status reports, etc..
- ✓ Controls
  - Internal constraints set by the development organization so that problem solutions aligns well with organizational goals and processes, e.g., processes, permitted tools, etc.



## ENGINEERING PROBLEM SOLVING – HOLISTIC APPROACH

- Interpreting the problem
  - ✓ Performed during the Initial State of problem-solving
  - ✓ Problem classification, e.g., well-defined, ill-defined, wicked.
- Evaluate constraints
  - ✓ Identify external constraints to set bounds on the solution landscape.
- Collaborative brainstorming
  - ✓ Performed during the Operational State of problem-solving.
  - ✓ Think about different solutions to the problem, i.e., divergent thinking.
- Synthesize the possibilities
  - ✓ Performed during the Operational State of problem-solving.
  - ✓ Switch to convergent thinking to seek one acceptable solution.
- Evaluate solution
  - ✓ Performed during the Operational State of problem-solving.
  - ✓ Flaws in the solution may trigger a transition back to the collaborative brainstorming activity.

## SUMMARY OF ENGINEERING DESIGN AND PROBLEM-SOLVING

- We've provided general information about engineering design.
  - ✓ Motivation for software engineering design
  - ✓ General engineering concepts
  - ✓ General problem-solving concepts
  
- In the next session, we will focus our efforts to explain engineering design in the software domain, i.e., software engineering design. Specifically, we will cover:
  - ✓ A focused discussion on software engineering design
  - ✓ Software design challenges
  - ✓ Software design process
    - Software architecture
    - Detailed design
    - Construction design
    - HCI design

## REFERENCES

- [1] IEEE. “IEEE Standard Glossary of Software Engineering Terminology.” IEEE, 1990.  
<http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=2238>, Retrieved on August 23, 2012.
- [2] <http://www.merriam-webster.com/dictionary/systematic>
- [3] [http://news.cnet.com/8301-11128\\_3-20021346-54.html](http://news.cnet.com/8301-11128_3-20021346-54.html), Retrieved on August 23, 2012.
- [4] [http://www.youtube.com/watch?v=CjjASGV36mw&feature=player\\_embedded](http://www.youtube.com/watch?v=CjjASGV36mw&feature=player_embedded), Retrieved on August 23, 2012.
- [5] [http://www.as.northropgrumman.com/products/nucasx47b/assets/X-47B\\_Navy\\_UCAS\\_FactSheet.pdf](http://www.as.northropgrumman.com/products/nucasx47b/assets/X-47B_Navy_UCAS_FactSheet.pdf), Retrieved on August 23, 2012.
- [6] Dym, Clive L., AND Patrick Little. Engineering Design: A Project-Based Introduction. Hoboken, NJ: Wiley, 2008.