Complex Systems

Lecture 03, v03

John G. Artus

BSEE

MSSE

INCOSE ESEP

© Copyright 2022-2023 John G. Artus

About This Courseware

- The majority of the material presented in this course is sourced from the textbook “System Architecture” by Edward Crawley, Bruce Cameron, and Daniel Selva
- I, John Artus, make no claim of ownership of the material sourced from this textbook
- I, John Artus, am using the material sourced from this textbook, and other indicated sources, as content for this courseware for educational purposes only
- This courseware lecture material has been sourced, interpreted, assembled, formatted, and copyrighted by John G. Artus for use in this educational context
- Anyone may freely access, and reuse this material in an educational context provided the copyright owner, John G. Artus, is recognized as the interpreter, assembler, and formatter of the source material used in the generation of this courseware, and provided that Edward Crawley, Bruce Cameron, and Daniel Selva are recognized as authors of the textbook “System Architecture” from which the majority of the content of this courseware has been sourced

© Copyright 2022-2023 John G. Artus

Complexity

- A complex system has many interrelated, interconnected, or interwoven entities and relationships
- Complexity is driven into systems by "asking more" of them
  - More function
  - More performance
  - More robustness
  - More flexibility
  - …
- It is also driven into systems by asking systems to work together and interconnect with increasing scale and scope
- Complex systems require a great deal of information to specify and describe
- Ways of measuring complexity
  - Based on the information content of the description of the system
  - Categorizing what the system does (a function-based approach)

© Copyright 2022-2023 John G. Artus

Complexity (continued)

- An idea closely related to complex is complicated
  - Takes into consideration the finite ability of the human to perceive and understand complexity
  - Complicated things have high apparent complexity
  - They are complicated because they stretch or overwhelm our ability to understand them
- Dealing with complexity is not new
  - Over the last century, systems and the context in which they operate have become more complex
  - This increasing complexity has strained our ability to comprehend systems
- Our job as architects is to train our minds to understand complex systems so that they do not appear to be complicated to us
- We should seek to build architectures that do not appear complicated to all the others who have to work on the system — designers, builders, operators
- The task of good architecture design can be summarized as follows
  - Build systems of the necessary level of complexity that are not complicated

© Copyright 2022-2023 John G. Artus

Decomposition

- Decomposition is the dividing of an entity into smaller pieces or constituents
  - It is one of the most powerful tools in our toolset for dealing with complexity
  - Break a problem down into smaller problems until each is tractable
- Sometimes defining decomposition is easy
  - When the system is made up of distinct elements
- Sometimes the system is modular
  - This suggests use of decomposition of the modular structure
- Sometimes the system is integral
  - Decomposition can then be somewhat arbitrary since there is no clear separation of components

© Copyright 2022-2023 John G. Artus

Decomposition (continued)

- The difficulty with breaking things apart is not in the breaking apart, but in the process of later bringing together the decomposed entities to build the whole
  - This process is often called integration
  - In the formal domain
    - Form aggregates
    - We have to worry about the physical/logical fit between the elements as they are brought together
  - In the functional domain
    - We decompose functions into more basic functions
    - In recombining the entities of function, we encounter emergence: the real challenge

© Copyright 2022-2023 John G. Artus

Hierarchy

- In a hierarchical system, the system entities belong to layers or tiers
  - Layers are ranked one above the other
- The top-most level of a system hierarchy is referred to as Level 0 (or Tier 0)
  - Below that, we refer to Level 1 (down), and Level 2 (down), and so on
- Hierarchy is can be used to understand and reason about complexity in systems
- What causes some tiers to be higher in the hierarchy than others?
  - They have more scope
    - Governors rank above mayors because they have more scope (a state is larger than a city)
  - They have more importance or performance
    - Black belts rank above brown belts because they perform to a higher standard and have greater skills
  - Their functions entail more responsibility
    - Presidents rank above vice presidents because they have more responsibility
- Hierarchy does not imply that a system element on a particular level is subordinate to any specific element at the next higher level
  - That observation only occurs when performing hierarchical decomposition

© Copyright 2022-2023 John G. Artus

Hierarchical Decomposition

- Often, decomposition and hierarchy are combined into a multilevel or hierarchical decomposition: a decomposition with more than two levels
- Organizing a hierarchical system in this way appeals to our desire to group things in sets of seven +/– two
- System classification scheme:
  - Simple systems
    - A system that can be completely described by a one-level decomposition diagram
    - There are generally no more than seven +/– two elements of form at Level 1
    - When you get to these elements, they are more or less atomic parts
  - Systems of medium complexity
    - A system can be represented by a two-level decomposition diagram
    - There are no more than (seven +/- two) parts — that is, no more than about 81 entities
  - Systems of complex complexity, or "complex" systems
    - A complex system has the same representational diagram as a medium-complexity system
    - At the second level (down) there are still only abstractions of things below
    - These are significantly harder to analyze, and more common

© Copyright 2022-2023 John G. Artus

Hierarchical Decomposition (continued)

- One rarely sees a single decomposition diagram more than two levels deep
- There are two reasons why such drawings are not frequently used
  - A three-level diagram could have as many as (seven +/– two) elements at the third level
    - Or as many as 729
    - This is far more than humans can routinely process
  - When we look down into the organization of a system, we find that we know the Level 1 elements fairly well
    - We more or less comprehend things on the next level (Level 2)
    - Beyond that (Level 3 and lower) our understanding and appreciation of the system organization is much less clear

© Copyright 2022-2023 John G. Artus

Two Levels (Tiers) of Decomposition

1 element at Tier 0

7 elements at Tier 1

49 elements at Tier 2

If fully populated, the number of entities quickly becomes overwhelming

Hierarchical Decomposition (continued)

- The designation of Level 0 as “the system” is somewhat arbitrary and depends on the viewpoint of the architect
- There are dozens of words to designate the layers below “the system”
  - modules, assemblies, sub-assemblies, functions, racks, online replacement units (ORUs), routines, committees, task forces, units, components, sub-components, parts, segments, sections, chapters, and so on
- Unfortunately, there is little consensus about how these terms are applied
  - One person’s assembly is another person’s component
- There are relatively fewer words for the levels above “the system”
  - People speak of systems of systems (SoS), family of systems (FoS), complexes, collections, etc
  - If needed, we will call these Level 1 (up) and Level 2 (up)

© Copyright 2022-2023 John G. Artus

Atomic Parts

- The term “atomic part” does not have an exact definition
- In mechanical systems, atomic parts are those that cannot easily be “taken apart”
- In information systems, the definition of “atomic part” is even fuzzier
  - A useful test is to call something an atomic part when
    - It possesses a fundamental semantic meaning (as does a word or an instruction)
    - It is a unit of data or information
  - These words, instructions, or data units would include details, of course
  - Because all information is an abstraction, defining meaningful abstractions of abstractions is necessarily more fuzzy
- Call it a part if it loses meaning when you take it apart

© Copyright 2022-2023 John G. Artus

Special Logical Relationships

- Class/Instance Relationship
  - More commonly used in software than in hardware
  - A class is a construct that describes the general features of something, and an instance is a specific occurrence of the class
  - The instance is often referred to as the instantiation of the class
  - The idea of class/instance is quite common
- The Specialization Relationship
  - Describes the connection between a general object and a set of more specific objects
  - Used extensively in design
  - Akin to the concept of inheritance in object-oriented programming
    - A class can be created starting from a more general class
    - From which some attributes and functionality are inherited
    - And to which new attributes and functionality are added
- Iteration
  - Iteration is commonly and very explicitly used in software engineering
  - Occurs when a process is performed several times <u>at the same level</u> of a hierarchy
  - Occurs when a routine or function uses itself at the same level
  - Occurs when entities or relationships are used in a self-similar way
- Recursion
  - Recurrence is commonly and very explicitly used in software engineering
  - Occurs when a process or object uses itself within the whole
  - Occurs when a routine or function uses itself
  - Occurs when an approach used at one level of a system is <u>used again at lower levels</u> within the system
  - Occurs when entities or relationships are used in a self-similar way

© Copyright 2022-2023 John G. Artus

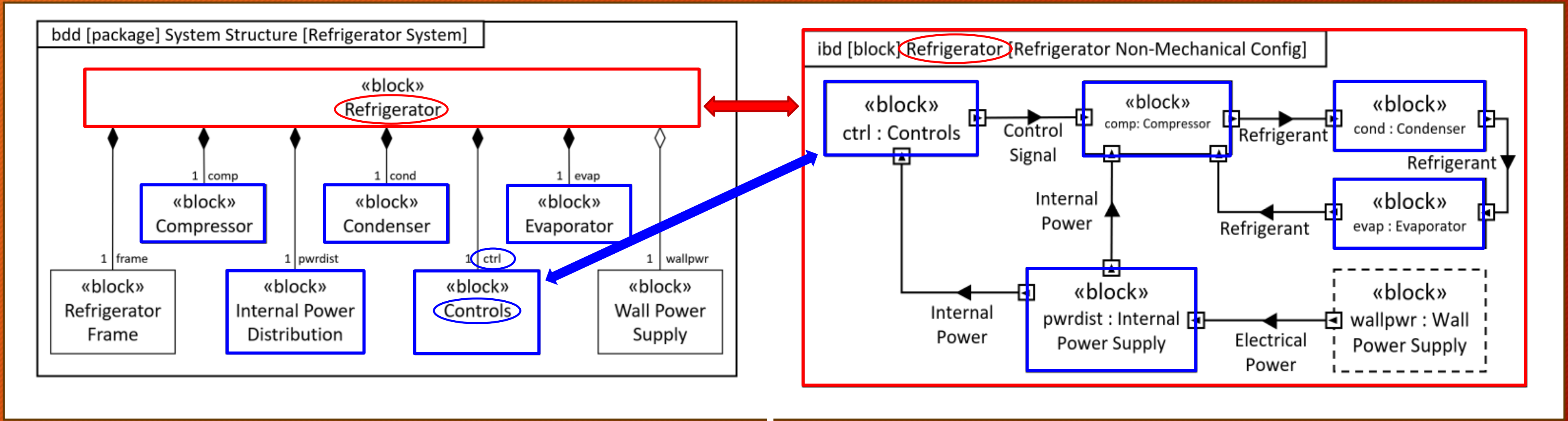
Class/Instance Relationships

A class is a construct that describes the general features of something, and an instance is a specific occurrence of a class

The instance is often referred to as the instantiation of the class

The idea of class / instance is quite common

In Systems Engineering the most common example is that of the definition of a block (class) and description of a particular usage of a block (an intance of that class)



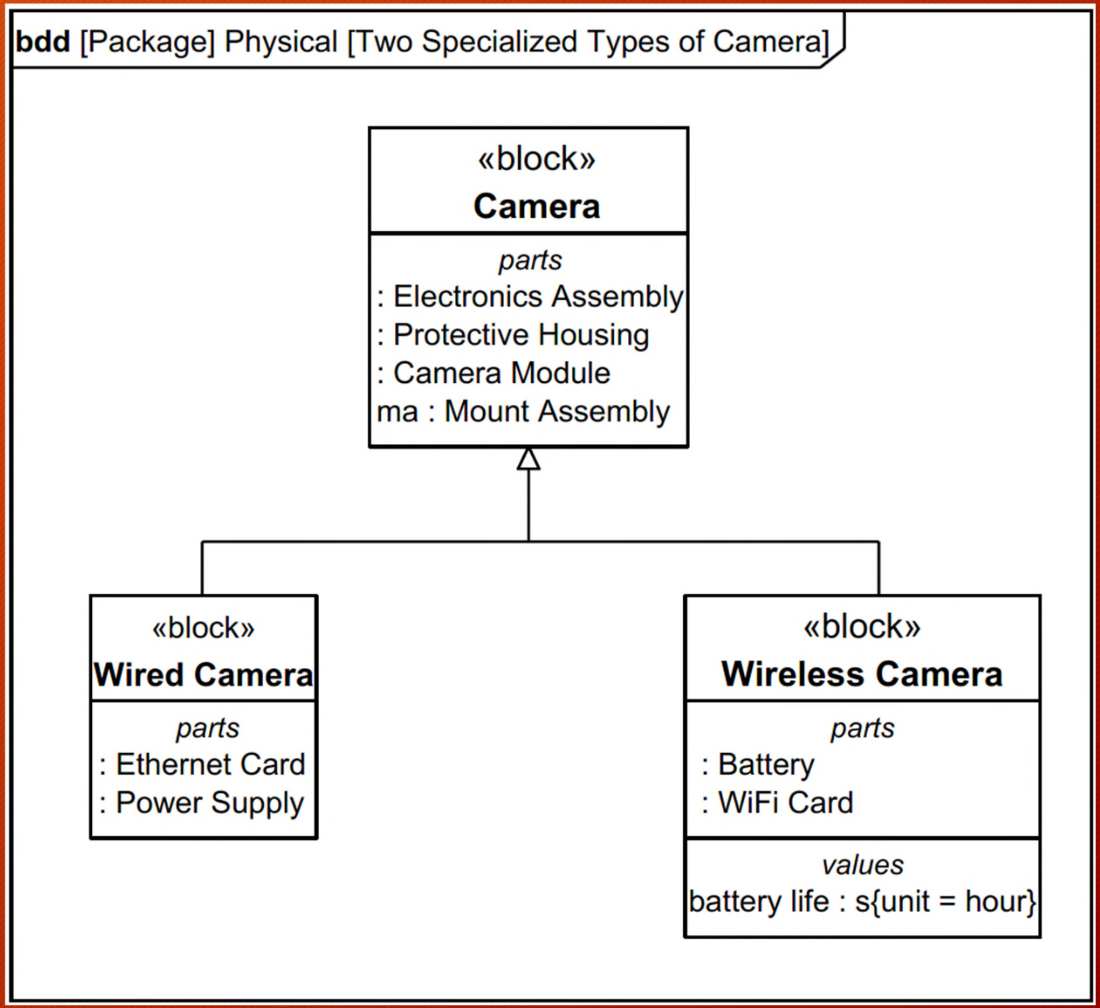
- Serves like class definition in software
- Defines the properties of block types (properties are not shown in the above example)
- Used to “type” the usage of blocks in an IBD
- The example above shows the decomposition of the “whole” Refrigerator block into its component “parts” (represented by the black diamond)
- This BDD describes the “definition” of a “Refrigerator”
  - If any of the components defined here are missing in a deployment, then the system does not fit this definition of Refrigerator

- Serves like an object instance in software
- Describes the use of one or more of the block types described in a BDD
  - The IBD frame represents the “whole”
  - In this case, the frame is the Refrigerator
- The blocks inside the IBD frame represent the Refrigerator’s “parts” in one particular configuration (“usage” – meaning how they are arranged, or connected together)
- There could be other arrangements or configurations
- Parts indicate what “type” they are
  - **Ctrl : Controls** – says “ctrl is of type Controls”
- Parts inherit all the properties of their type

© Copyright 2022-2023 John G. Artus

Specialization Relationship

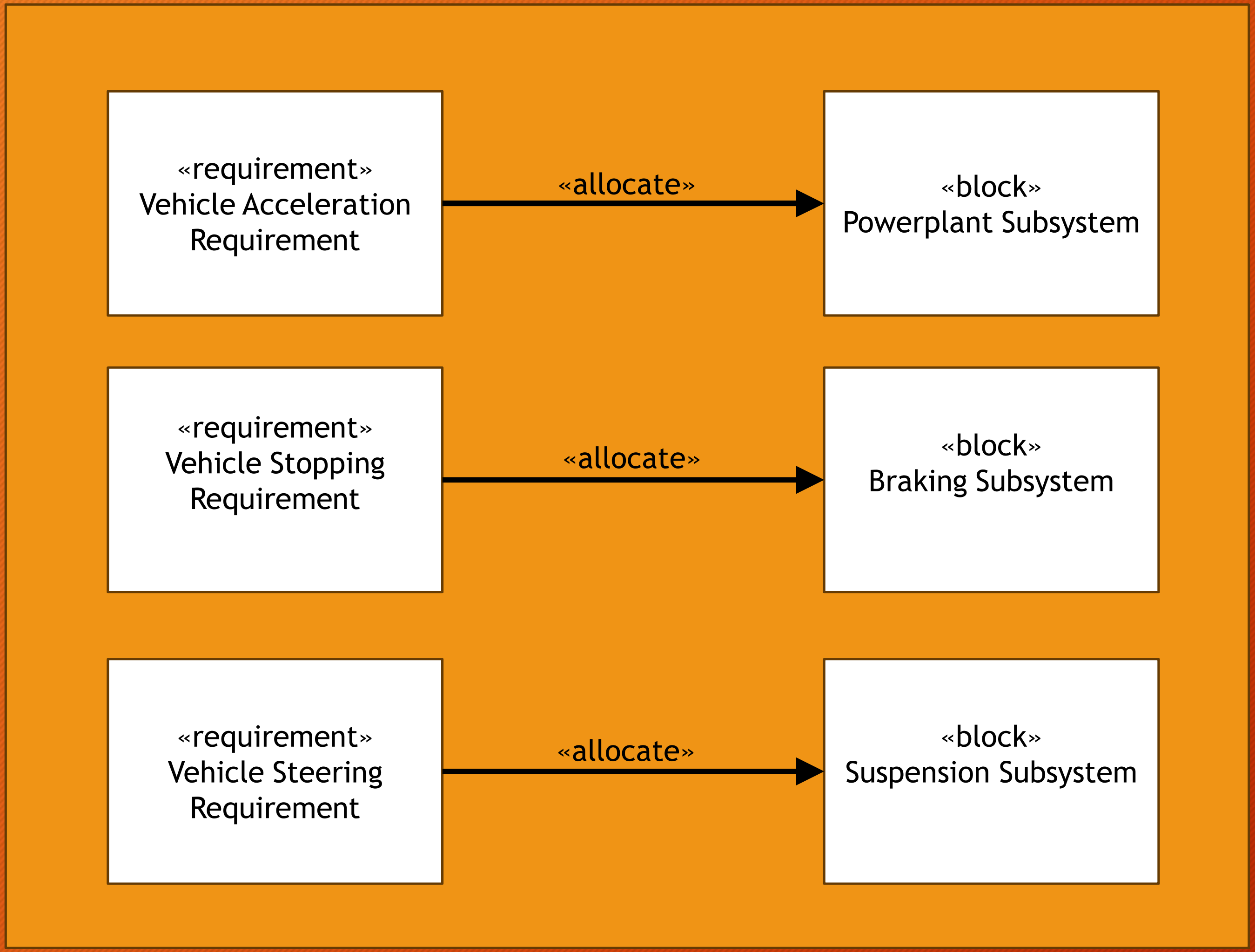
- Describes the connection between a general object and a set of more specific objects
- Used extensively in design
- Akin to the concept of inheritance in object-oriented programming
  - A class can be created starting from a more general class
  - From which some attributes and functionality are inherited
- In this example, the superclass "Camera" is specialized into two subclasses
  - The "Wired Camera" class adds features, such as Ethernet Card, that specialize the subclass apart from the more general class "Camera"
  - Equivalently, the "Wireless Camera" defines parts, such as WiFi Card, that are unique to that subclass of Camera



"A Practical Guide to SysML" by Friedenthal, Moore, and Steiner

© Copyright 2022-2023 John G. Artus

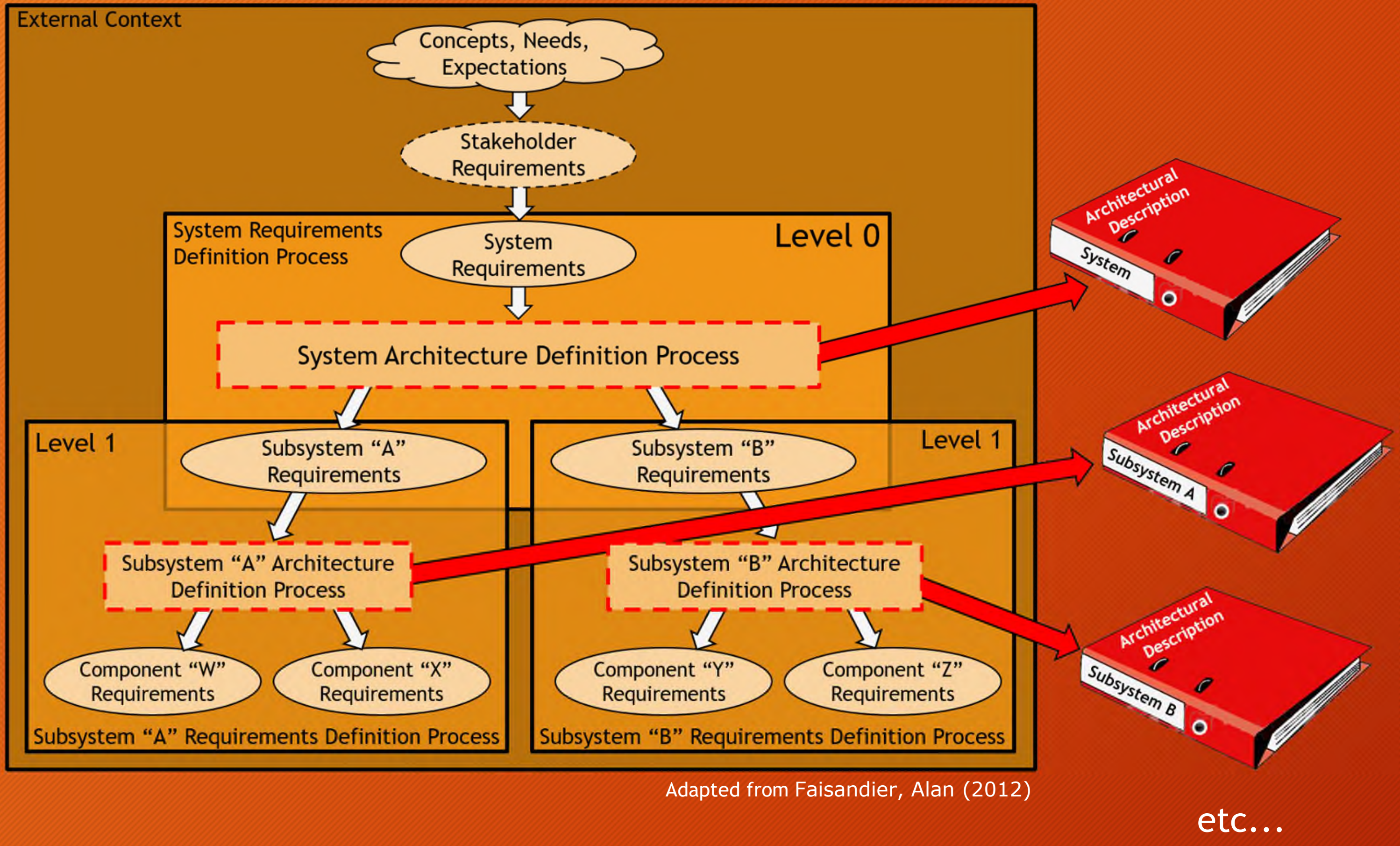
Iteration



- Occurs when a process is used at one level of a system and then is used again, repeatably, at the same level within the system
- In the example shown, the same process is used multiple times to allocate a system requirement to be implemented by a specific system component within the same level of the system hierarchy

© Copyright 2022-2023 John G. Artus

Recursion



Adapted from Faisandier, Alan (2012)

- Occurs when an approach used at one level of a system is used again at other levels within the system
- In the example shown, the Architecture Definition Process is repeated at each level of the system hierarchy, as well as within each element of a given level of the hieararchy

© Copyright 2022-2023 John G. Artus

Reasoning Through Complex Systems

v02

- Top-down Approach
  - Refers to the direction in which you approach a system
  - Start from the goals of a system and proceed to the concept and then the high-level architecture
  - Then develop the architecture in increasing detail until you reach the smallest entities of interest to you
  - This method follows the "left-hand side" of the systems engineering VEE model
- Bottom-up Approach
  - Think about the artifacts, capabilities, or services that are available in the lowest-level entities
  - Build upward from them, predicting emergence
- Outer-In Approach
  - Start at both top and bottom
  - Work toward the middle
- Middle-Out
  - Start at some arbitrary point in the system hierarchy
  - Try to reason one or two levels up or down
- There is really no top or bottom to truly complex systems
  - In reality we end up applying a middle-out approach

Good architects should be able to apply all of these approaches

© Copyright 2022-2023 John G. Artus

Zig-Zagging

- Zigzagging
  - When reasoning about a system, alternate between reasoning in the form domain and reasoning in the function domain
  - Start in one domain
  - Work as long as is practical
  - Then switch to the other domain
- This alternating pattern of thinking continues down through all levels of the system design

© Copyright 2022-2023 John G. Artus

References

1. Crawley, Edward; Cameron, Bruce; Selva, Daniel (2016). *System Architecture: Strategy and Product Development for Complex Systems*, Pearson Higher Education Inc, Hoboken, NJ
2. S. Friedenthal. A. Moore, R. Steiner. (2019). *A Practical Guide to SysML, The Systems Modeling Language, Third edition*, Elsevier Inc, Waltham, MA
3. Forsberg, Kevin; Mooz, Hal; Cotterman, Howard (2005). *Visualizing Project Management: Models and Frameworks for Mastering Complex Systems*, 3rd Edition, John Wiley & Sons, Inc., Hoboken, New Jersey

© Copyright 2022-2023 John G. Artus