Cohesion And Coupling

Lecture 43, v02

John G. Artus

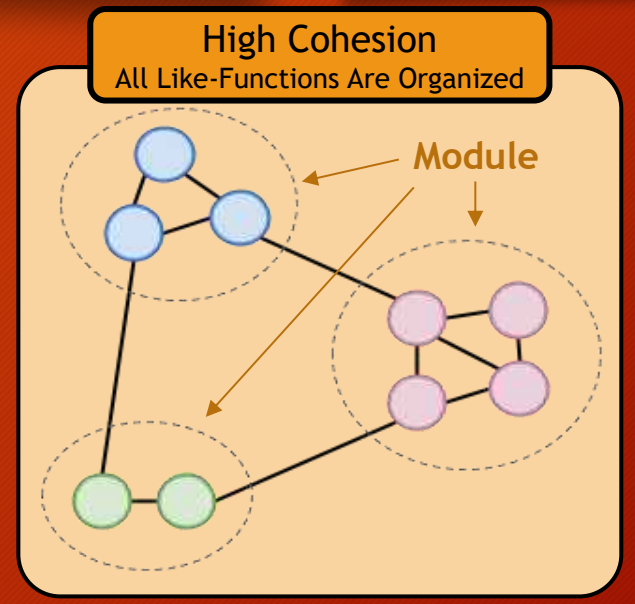
BSEE

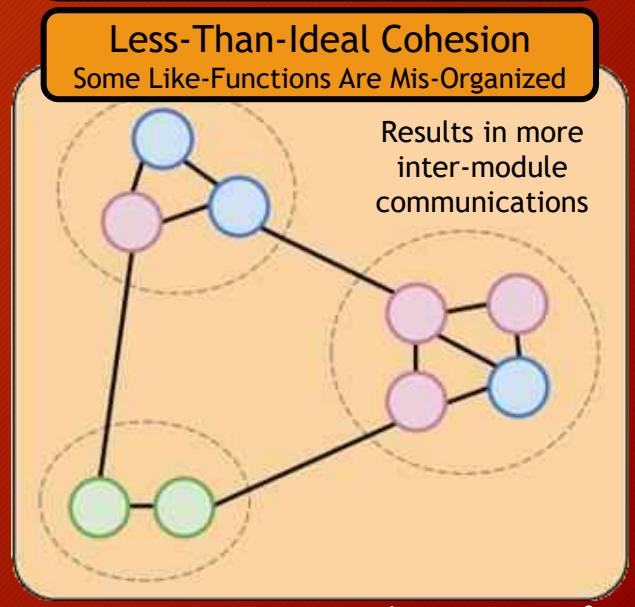
MSSE

INCOSE ESEP

Overview of Cohesion and Coupling

- Cohesion and coupling are common concepts in designing modular systems
  - Cohesion and coupling are defined in terms of a system and the inter-module and intra-module relationships among a system's functional components
- The terms cohesion and coupling are deliberately chosen to be ambiguous and abstract, because the concepts are abstract
  - They each manifest in some form, within any kind of system we might care to consider, in accordance with the skill of the architect to synthesize the individual, lower-level functions cohesively into the same module
- Generally, it is the goal of the architect to achieve high cohesion and loose coupling
  - High cohesion indicates modules whose functional components relate closely with each other in terms of them all working to fulfill common functional objectives of the parent module
    - Since the functional components are so closely related, most of the information needs will be satisfied through intra-module communications (within the same module)
  - Loose coupling indicates that there is less need for modules to communicate with each other in order to support the information needs of the module's functional components
    - This means that the number of intra-module communications (among different modules) is reduced, since highly cohesive functions within a module get most of what they need from within the module
    - The opposite is tight coupling in which modules are tightly coupled due to the poor placement of functional components within modules that require them to conduct extensive inter-module (module-to-module) communications to satisfy their information needs





Index

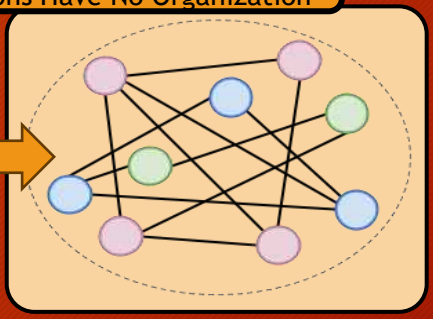
Cohesion

Overview of Cohesion

- Cohesion is the degree to which the functional components of a system support the functional objectives of a single functional "module"
  - A "module" can be considered to be the hierarchical functional parent of the functions grouped together
  - A module with high cohesion contains elements that are highly related to each other and united in their purpose
- Cohesion is an intra-module concept (within the same module)
- A highly-cohesive module is generally stable in the face of changes made to other modules
  - This is due to the fact that highly cohesive modules tend to be more independent of each other
  - Such greater independence tends to isolate one module from changes implemented in other modules
- High cohesion means that a module has a lesser number of concerns to deal with, since the module is tightly focused on its objective (it does one thing well) and its interfaces are simpler
- Low cohesion means that a module has a greater number of concerns to deal with (it tries to do too many things) due to the fact that its functions are varied in purpose, and/or it has more complex interfaces
- Highly cohesive modules tend to have a higher degree of intra-module coupling (internal to the module)
  - This is a natural result of high cohesion, in that the functions work cooperatively within the module to satisfy their functional concern
  - This is actually good, because it mean fewer inter-module communications are needed
- Less cohesive modules tend to have a higher degree of inter-module coupling (among multiple modules)
  - This is a natural result of low cohesion, in that the functions are poorly organized, and need to work with functions from other modules in order to satisfy their functional concern
  - This is not good, because it mean a greater number of inter-module communications are needed

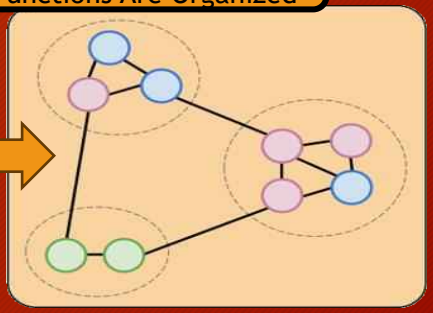
Low Cohesion
Like-Functions Have No Organization

To communicate, a large number of interfaces need to be established



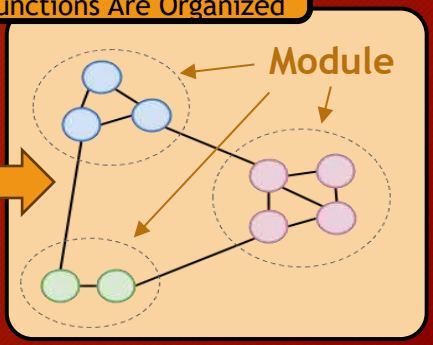
Moderate Cohesion
Most Like-Functions Are Organized

To communicate, some like-functional elements must do so using inter-module interfaces (across modules)



High Cohesion
All Like-Functions Are Organized

This is the ideal arrangement, but is not always achievable



Benefits of High Cohesion

- Simplifies modification
  - It is easier to make changes to a module's behavior since all the related functionality is resident within the module
  - This will keep the area of impact limited
  - Compare this to a design in which a certain functionality is spread across multiple modules, and each has to be changed to achieve the desired change in behavior
- Easier to test the module
  - Since such modules do not depend on other modules for their behavior, they are easier to test
- Changes are less prone to bugs
  - Alternatively, when you are making changes across modules, it is easier to make mistakes
- Facilitates reuse
  - Since such modules perform a single responsibility, they tend to be reused wherever there is such a need
- Leads to low coupling with other modules
  - Because the functionality contained in a single module is tightly contained, integration with other modules experiences fewer inter-module communications
- Reflects better quality of design
  - The design indicates that any functional components in a module that are not directly related to the main purpose have already been moved to some other module that better fits the functional purpose

Functional Analysis Versus Functional Synthesis

- Functional Analysis
  - At the beginning of a program, architects will likely not have a complete understanding of the detailed functionality the system needs to perform in order to deliver the desired behavior
  - A key operation that is performed during functional analysis of the system problem, is functional decomposition
  - Functional decomposition allows the architect to break down the proposed system functionality down to its elemental functional components
  - The purpose of this operation is to discover the detailed functionality that will be needed by the system to deliver the desired behavior

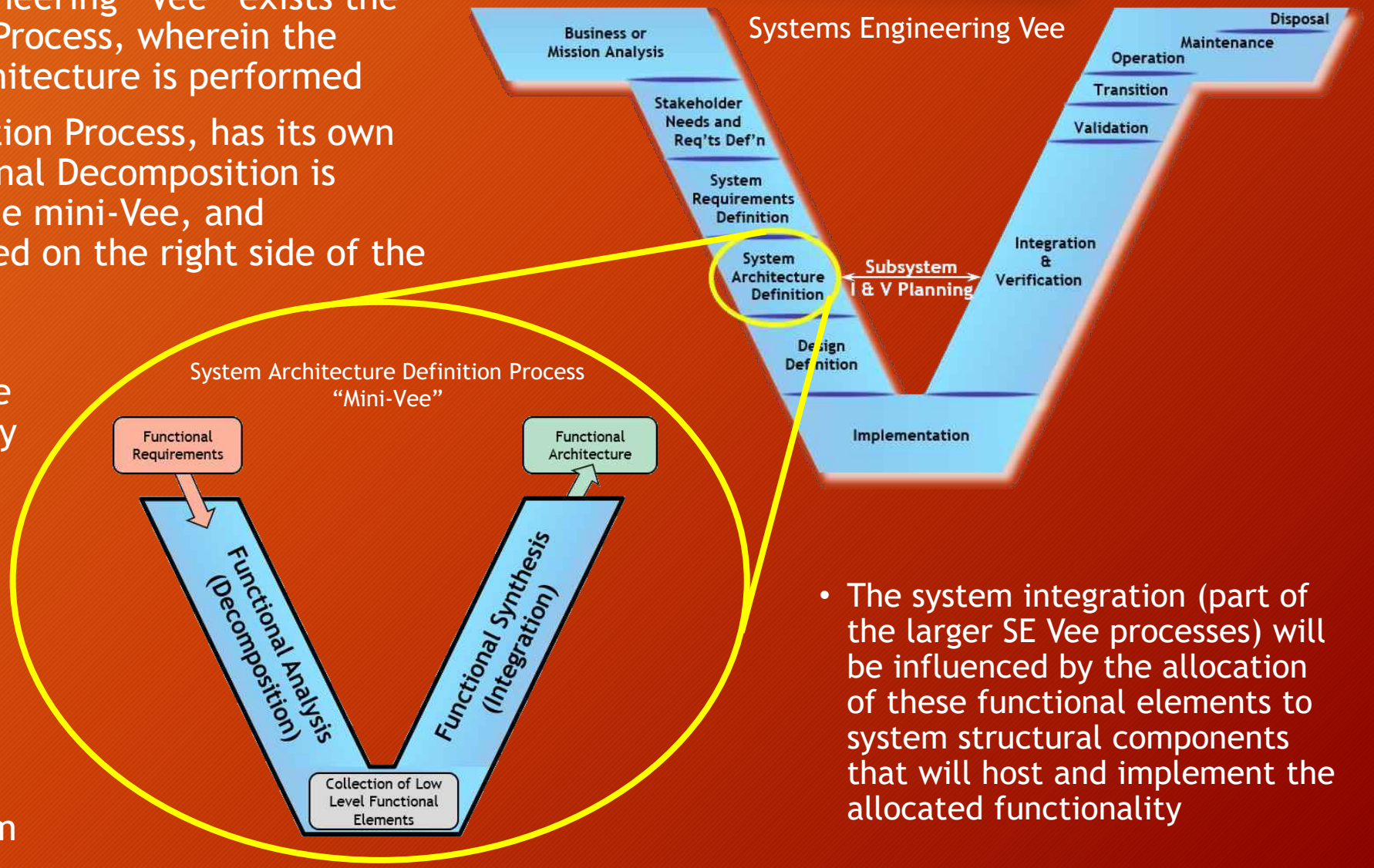
Identifying the Needed Functional Elements of the System

- Functional Synthesis
  - Now that this detailed functionality has been exposed, it is the architect's job to then assemble the functional pieces in a way that delivers behavior in the most effective and efficient way – this is functional synthesis (synthesizing the pieces together)
  - The result is a new functional hierarchy of synthesis, which could be quite different from the functional hierarchy of analysis
  - Cohesion (and therefore coupling) is established during functional synthesis
  - The challenge for the architect is to determine which of all the functional components have a common purpose that makes them likely candidates to form a cohesive module

Devising the Solution by Arranging the Functional Elements so as to exhibit High Cohesion and Loose Coupling

Functional Analysis and Synthesis within the Sys Arch Def'n Process

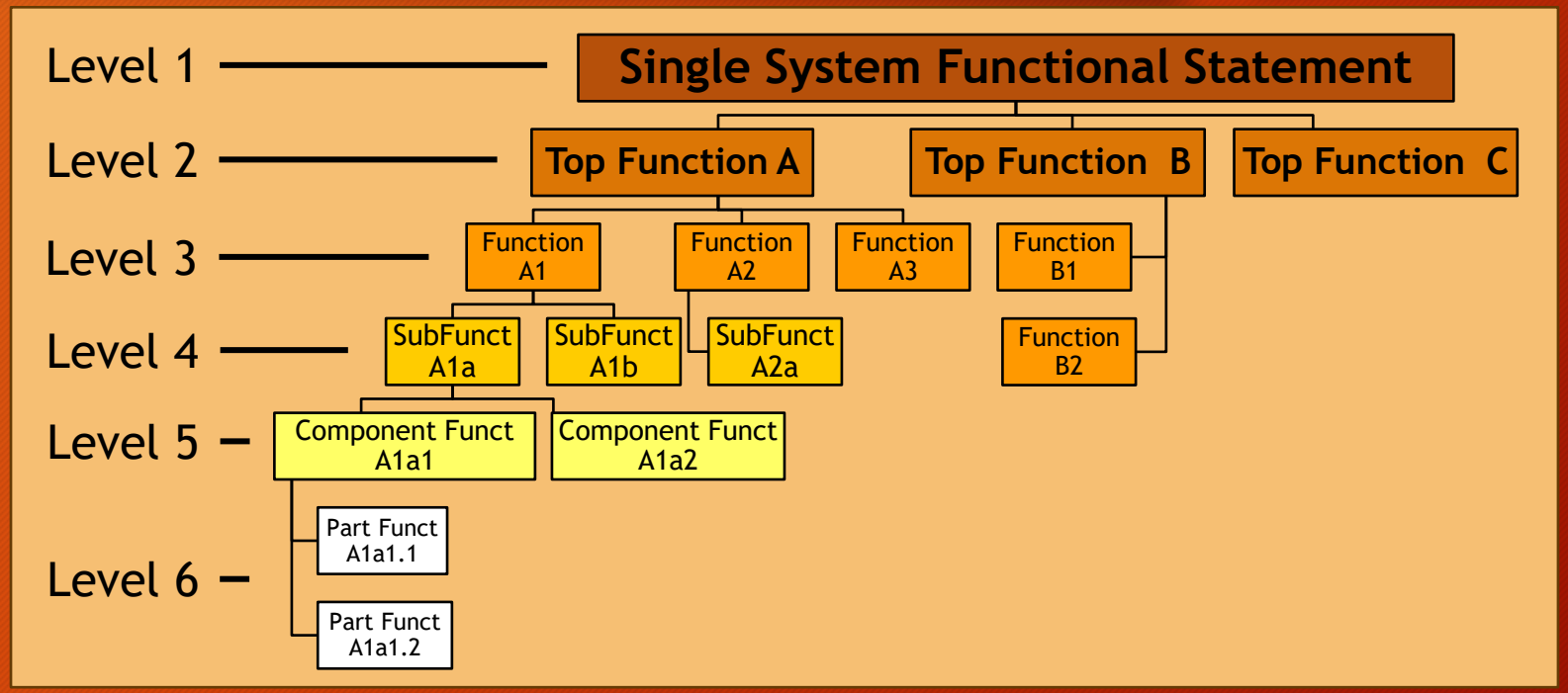
- Within the overall Systems Engineering "Vee" exists the System Architecture Definition Process, wherein the development of the system architecture is performed
- The System Architecture Definition Process, has its own little mini-Vee in which Functional Decomposition is performed on the left side of the mini-Vee, and Functional Synthesis is performed on the right side of the mini-Vee
- The architect employs experience and skill to integrate the functional elements in a way so as to promote high cohesion and loose coupling
- The end result of this integration activity is not the system as a whole
  - That integration activity is part of the larger SE Vee
- The end result of this integration activity is the system functional architecture



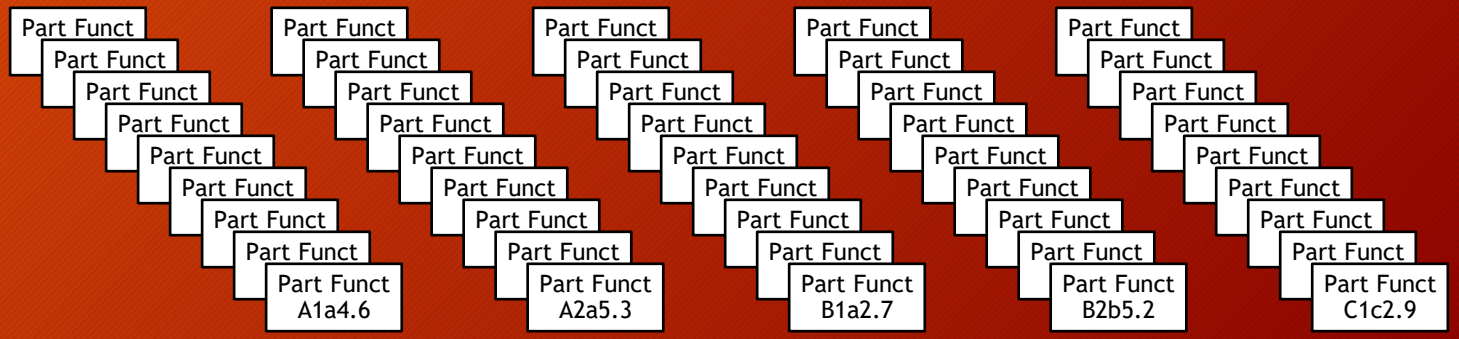
- The system integration (part of the larger SE Vee processes) will be influenced by the allocation of these functional elements to system structural components that will host and implement the allocated functionality

The Result of Functional Decomposition During Functional Analysis

- During functional analysis, functional decomposition is used as a tool to decompose the top-level functionality of the system (as known) to understand the detailed functions that are needed to deliver the final system behavior to the stakeholder
- At the end of the process we do not yet have a system
  - All we have is a set of detailed functions
  - They have not yet been assembled into a working system
  - That is the objective of Functional Synthesis
- The architect may very well discover that detailed functionality that supports one top-level function, may have a great deal in common with lower-level functions that support different top-level functions
  - The challenge is to smartly combine lower-level functions that have significant commonality to form modules
  - This is where the architect's skill comes into play

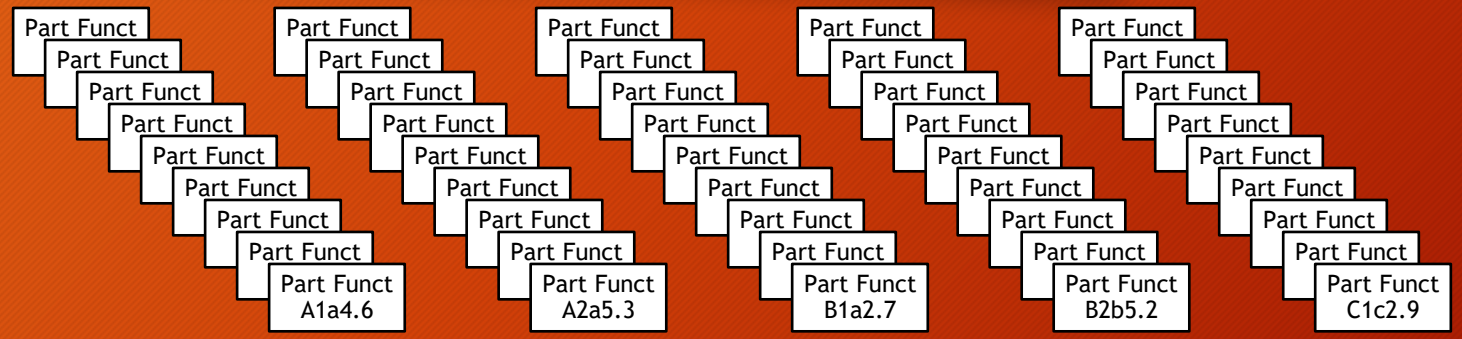


The result of functional decomposition could be MANY low-level functions (maybe not all at the same level as shown)

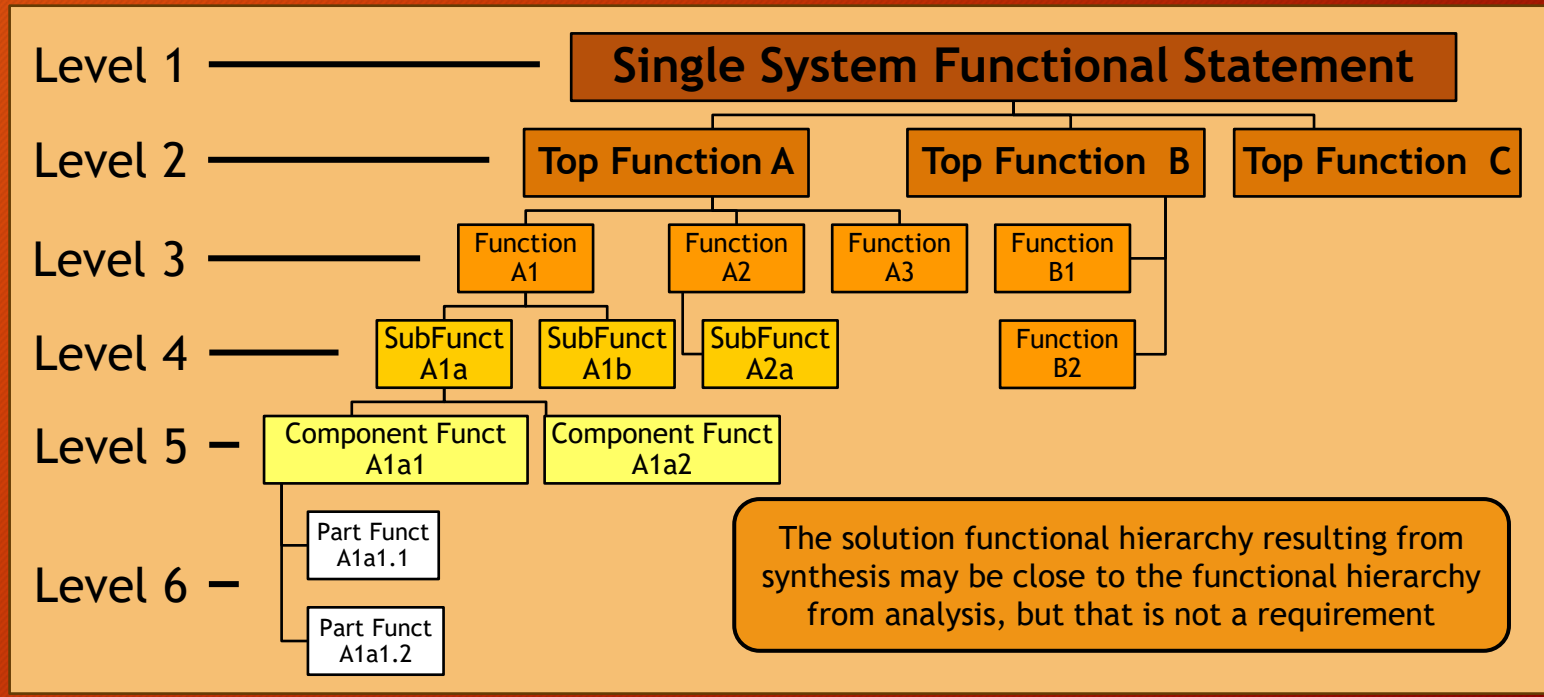


Functional Synthesis Results in the System Functional Hierarchy

- The full set of low-level functions (which might not all be at the same level) are used to then synthesize (assemble) the final functional hierarchy that represents the functional architecture of the solution
- The solution hierarchy may be somewhat similar to the hierarchy coming out of the analysis process, but it does not have any such requirement placed upon it
- The challenge is to smartly combine lower-level functions that have significant commonality to form modules
- This is where the architect's skill comes into play
- The hierarchy of these modules at multiple levels constitutes the final functional hierarchy of the system solution



The job is to identify commonality among all these functional components, and combine them into modules at multiple levels



The solution functional hierarchy resulting from synthesis may be close to the functional hierarchy from analysis, but that is not a requirement

How to Achieve High Cohesion

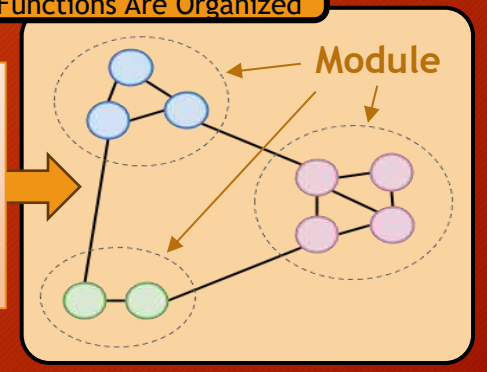
- The desired top-level functionality of the system should be clearly defined in the top-level functional requirements
- Establish top-level functions that clearly meet those requirements
- Decompose the established top-level functions down to levels that reveal the basic functions that need to be combined to adequately support the top-level functional requirements
- Arrange the decomposed functions into functional modules that are focused on achieving a particular functional goal
  - As such a hierarchy is developed, it is critical to ensure that all functional components of a module implement functionality that supports the same functionality goal of the module (that all functional components are functionally related)
  - The combining of functional components into a module should result in a focus of the module on performing a single task
  - Aim to minimize interaction of the subject module with other modules of the system at the same and other levels
- Continue such organization to group like-purposed modules together into larger modules, into a multi-level hierarchy
  - While there may be close similarities between this synthesized functional hierarchy and the hierarchy resulting from functional decomposition during system analysis, there is no requirement for them to be identical
  - If the job is done right, there will likely be some differences indicating that a deeper understanding of the functional needs of the system has been achieved

Taking Modularization Too Far

- Good cohesion means that a module has few responsibilities (it does one thing well) and it has a simple interface
- So, you might think it a good thing to continue identifying modules to the point that every module is represented by a single function (the module does one simple thing and one thing only)
- Taking modularization to this extent only makes matters worse
  - Cohesion is actually reduced since the system functions have been broken up and scattered to a large set of unrelated modules
  - Breaking modules into the smallest possible elements will separate related concepts and will lead to very low cohesion
  - Such a degree of cohesion will inevitably increase the coupling between modules to the maximum, since inter-module communications will have to increase tremendously
- Cohesion is not something you can create automatically
  - Cohesion is a process of discovery during synthesis of the architectural components
  - Part of the art of architecture is to determine what is the right amount of modularization needed to provide the highest cohesion and loosest coupling
  - It is difficult to reliably measured cohesion since determining how closely individual functions relate to each other in fulfilling a functional requirement is a judgement call made by the architect

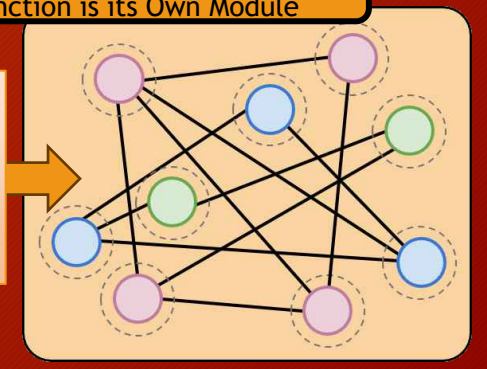
High Cohesion
All Like-Functions Are Organized

Finding the best level of "modularization" requires skill acquired over time spent trying, failing, and improving



Modularization Taken Too Far
Each Function is its Own Module

Attempting to push modularization too far can result in functionality spread too thin - not the ideal situation

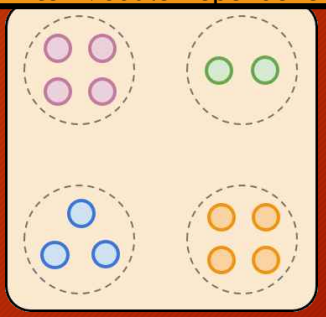


Coupling

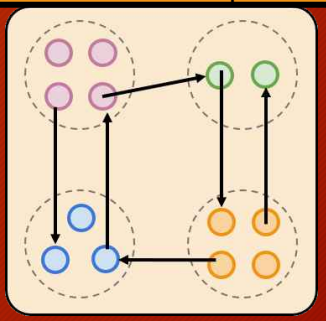
Overview of Coupling

- Coupling is the degree of interdependence between functional modules
- Uncoupled modules have no interdependence at all
  - It is impossible to construct an uncoupled system that has any practical functional behavior
- Modules with loose coupling work mostly independently of each other
  - Normally, the goal is to achieve a system with loose coupling
- Two modules are tightly coupled if they share many inter-module connections
  - A tightly coupled system can deliver functional behavior, at a higher cost of production and maintenance
- Modules with a single, well-defined purpose are easier to understand, implement, and maintain
  - The purpose of such modules is clear to understand
- Generally, coupling is a affected by cohesion
  - If high cohesion is achieve, it is likely that loose coupling will be the result

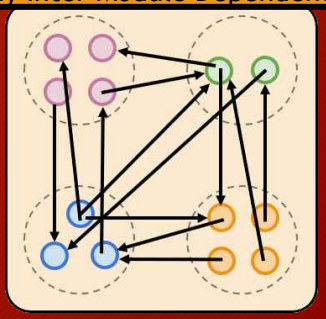
Uncoupled
No Inter-Module Dependencies



Loosely Coupled
Some Inter-Module Dependencies



Tightly Coupled
Many Inter-Module Dependencies



Advantages of Loose Coupling

- Loosely coupled modules are easier to develop
  - Since they are independent of each other, they can developed and tested in parallel
  - Such modules can be independently built and deployed, significantly reducing the deployment time
- Overall, loose coupling reflects a higher quality of design
  - Aim for designing components that are as independent as possible
- Coupling affects how the system is maintained
  - Effectively, coupling is about how changing one component requires a change in another component (dependence)
  - The more inter-module dependencies there are, the more expensive it will be to implement a change in functionality

Disadvantages of Tight Coupling

- Tightly coupled components are difficult to change (impacting extensibility and scalability)
  - The developer needs to understand multiple modules and how they relate to each other
  - You need to be careful in making changes to all the modules consistently
  - This makes the system more error-prone
  - Also, there is a need to build, test, and deploy each changed module, further increasing the development effort
- Tightly coupled modules are also difficult to test
  - Testing a single module is difficult since it heavily depends on other modules
- Integration tests are also difficult to set up
  - Overall the tests are fragile since a change in any one module could break the tests
  - Debugging such modules is also complex because it needs all the dependent modules operating concurrently
- A tightly coupled module is less likely to be reused
  - Because of interconnected dependencies, such a module does not perform anything useful on its own
  - Hence, it rarely fits the purpose for someone else to reuse
  - Also pulling it as a dependency is difficult since it brings in other dependent modules along with it
- Alternatively, under some circumstances, tight coupling has advantages
  - In special circumstances involving one-of-a-kind, customized applications
  - That may require very close association of multiple, unrelated modules for some special purpose, such as tight packing of modules in extremely limited physical space

How to Achieve Loose Coupling

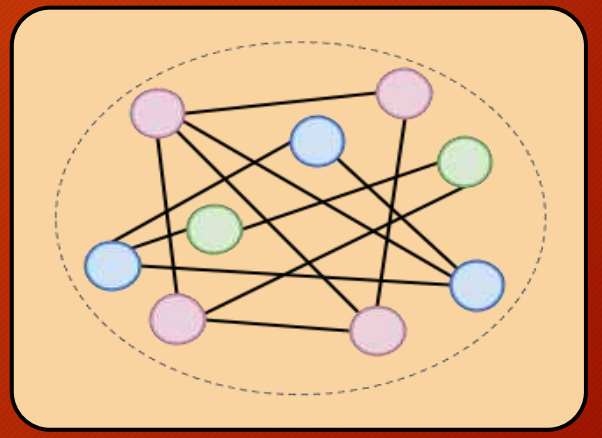
- Modules should be as independent as possible
  - Modules should communicate only via small, well-defined (“narrow”) interfaces
  - The more one module has to “know” about another module, the higher is the coupling between them
- Looking at different kinds of systems and the way they manifest internal coupling, the variations of this pattern are endless
  - It's always the case that if you can reduce the coupling between a system's modules, you can reduce the overall complexity of the system accordingly
- Among Systems Engineering best practices, as stated in many standards, it is key to minimize the coupling between modules in order to master the product complexity
  - Good coupling means that the system organization is clean and the dependencies are easy to understand
  - Alternatively, bad coupling means that the system architecture is an entangled mess, and the dependencies are hard to understand
  - To achieve such minimization, a well-known method consists of using Coupling Matrices (also called N2 diagrams) and then reorganize them to identify architectures with minimal coupling
  - The coupling between modules concerns the dependencies between the modules
  - These dependencies between the logical modules mainly stem from the functional interfaces
  - First start by examining the functional dependencies
    - Use them to develop a coupling matrix (N2 diagram)
    - Examine the interdependencies of modules within the system organization as revealed by the N2 diagram
    - Implement a re-allocation of functions to modules that minimizes coupling
- It is impossible to achieve full decoupling without damaging cohesion

Cohesion and Coupling in Architecture Patterns

Architectural Schemes Exemplifying Cohesion and Coupling

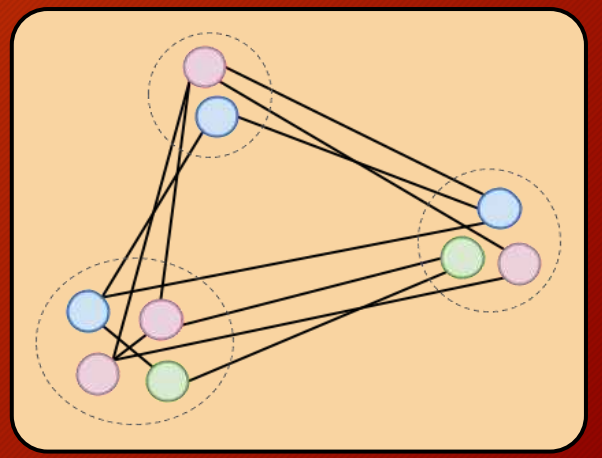
The following four figures on this slide and the next show the same elements with the same dependencies

- In each of the four figures, the system elements are organized differently
- Related domain concepts are represented with the same color



Organization Scheme: <u>Low Cohesion, Tight Coupling</u>

- System elements have no explicit boundaries
- There is no real fore-thought of promoting cohesion of any kind (low cohesion)
  - Any occurrence of cohesion is purely coincidental
- Interfaces are established as the need for them is identified
  - As a result, interfaces crisscross throughout the system (tight coupling)
- Such an architecture is also known as a “Big Ball of Mud”



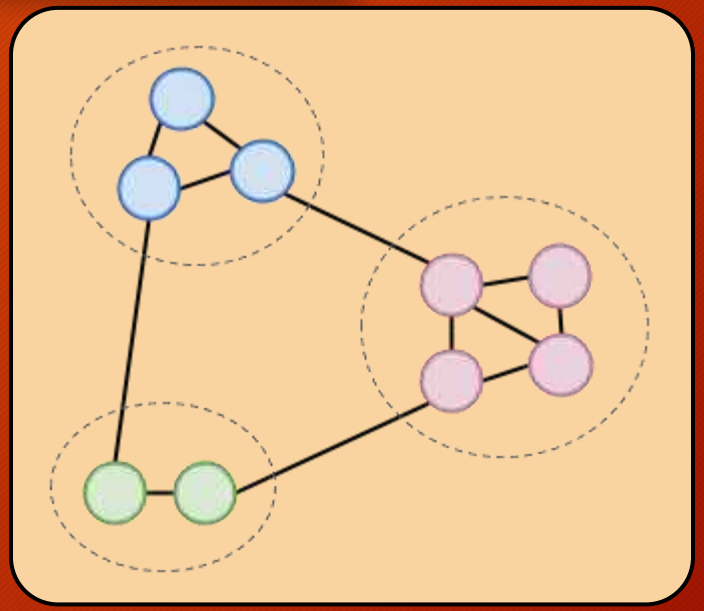
Organization Scheme: <u>High Cohesion, Tight Coupling</u>

- System elements are organized into modules with lots of dependencies between them
- Although the modules are highly cohesive, they are cohesive by the wrong principle
  - In this case, system elements are organized by something other than a domain relationship
  - They are NOT with regard to fulfilling any functional goal of the parent subsystem
- A typical example is an organization in accordance with the principle of separation of concerns, as shown in a Layered Architecture
  - Layered Architecture is a design pattern that that organizes a system into a set of horizontal layers and thus promotes clean separation of concerns
  - In this architecture, each layer has a specific responsibility and communicates only with adjacent layers

Architectural Schemes Exemplifying Cohesion and Coupling

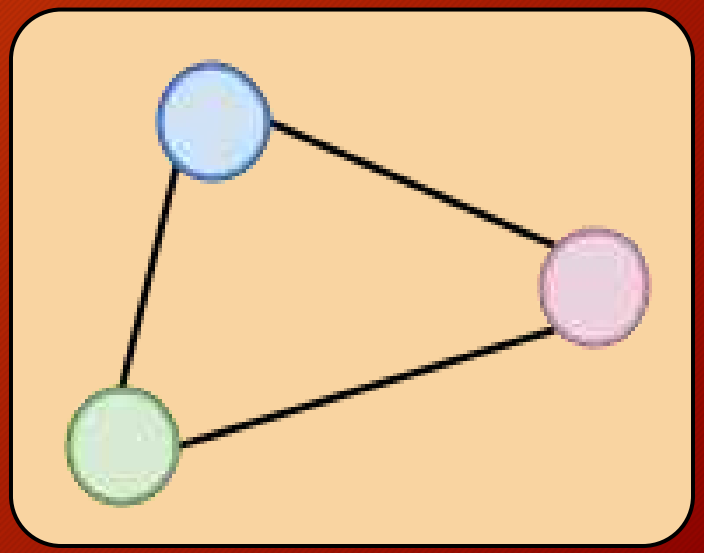
Organization Scheme: High Cohesion, Loose Coupling

- This system is organized according to the principle of functionality
  - What matters are the functions that a business domain needs to implement in order to succeed
- The domain defines the functional abstractions with a stable purpose that drives the cohesion of system elements
- This is the main idea of the Domain-Driven Design
  - Domain-Driven Design (DDD) is an approach that emphasizes system designs that reflect a focus on functionality that addresses the concerns of the business domain
- The focus is on enhancing the cohesion of domain functionality
- An example of this kind of approach is provided in the next few slides
  - The domain focus is on providing a multi-layered air defense against threatening aircraft



Organization Scheme: Low Cohesion, Loose Coupling

- Systems with this organization scheme are the result of incorrect understanding of the domain and applying purely technical approaches to decouple the modules in an arbitrary way
- In this scheme, interfaces exist everywhere with no abstraction representing any kind of domain purpose
- Is it even possible to have such an architecture?
- Unfortunately, it is; and it is actually pretty common



Cohesion Versus Coupling

Cohesion Versus Coupling

- Cohesion and coupling are related to each other
  - Each can affect the level of the other
- Generally, high cohesion correlates with loose coupling
  - A module having its elements tightly related to each other and serving a single purpose would rarely interact and depend on other modules
  - Thus, the system will have loose coupling with other modules
- Similarly, tight coupling could be a sign of low cohesion
  - Modules could be heavily dependent on each other due to the elements spread across the multiple modules
  - Such extensive interdependencies will result in low cohesion

	Cohesion	Coupling
1	In cohesion, a component focuses on a performing single function	In coupling, components are linked to other components
2	Cohesion is the degree to which the elements *inside* a component belong together	Coupling is the degree of interdependence *between* the components
3	Cohesion shows the component's relative *functional strength*	Coupling shows the relative *independence* between the components
4	A component with high cohesion contains elements that are tightly related to each other and united in their purpose	Two components have high coupling (or tight coupling) if they are closely connected and dependent on each other
5	A component is said to have low cohesion if it contains unrelated elements	Components with low coupling among them work mostly independently of each other
6	Highly cohesive components reflect higher quality of design	Loose coupling reflects the higher quality of design
7	Cohesion shows the component's relative functional strength	Coupling shows the relative independence between the components

Summary

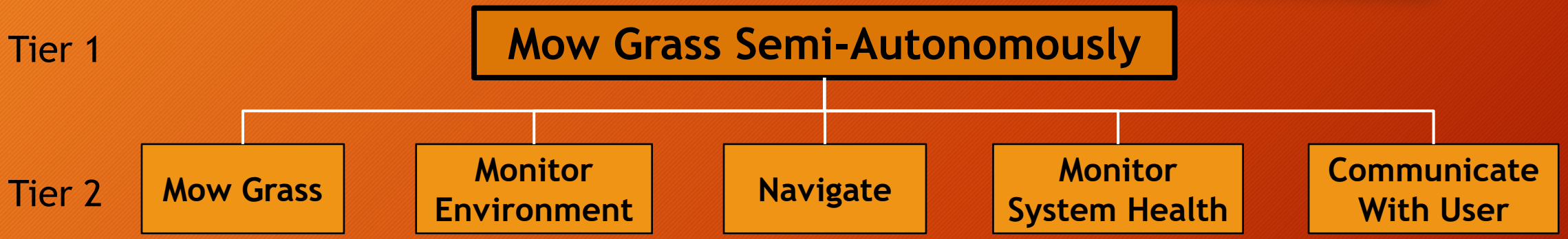
- Cohesion does not depend on the number of connections between elements, which is what coupling is all about
  - Cohesion is more about common functional purpose
  - Cohesive modules tend to have a higher degree of coupling *within* the module, which is expected and normal
- High cohesion and loose coupling help us reduce accidental complexity and create modules with well-defined boundaries
- In practice, measures of coupling and cohesion are always interdependent
  - As loose coupling is driven by high cohesion, we should strive for high cohesion in the first place
  - That is, changing one variable always changes the other; it is impossible to optimize one without degrading the other
  - For example, modifying a system to improve the cohesion of its modules is a good thing to do
    - But it generally results in having smaller modules than we had before and therefore more of them
    - Adding more modules to a system tends to increase its overall coupling
  - Conversely, if we try to improve (reduce) the coupling of a system by merging small modules together into larger ones
    - We would succeed in reducing the number of dependencies between modules
    - But, the new modules will necessarily have to have more responsibilities than the previous ones, and thus poorer cohesion

Summary (continued)

- Note that it is never possible to actually have perfect decoupling within a system
  - That is only achieved when none of the components interact with each other
  - That's not really a system, just an inert collection of components
  - Coupling is to a certain extent a good thing and a completely necessary thing, and in fact it's the thing that enables the functionality of the system
  - It's only when it gets out of hand and begins to hamper productivity that it becomes a problem
- In the case of cohesion, on the other hand, it is possible to achieve perfection
  - For example, a class that contains just a single method is a perfectly cohesive component of its system
  - However, we're not going to want to take things to that extreme in practice
    - Because of the surge in coupling it would incur
    - A system made up of a large number of tiny components will tend to manifest a great amount of coupling among them
- In a sense, coupling is a manifestation of what a system does, and cohesion is a manifestation of what the system doesn't do

Example: Semi-Autonomous Lawn Mowing System

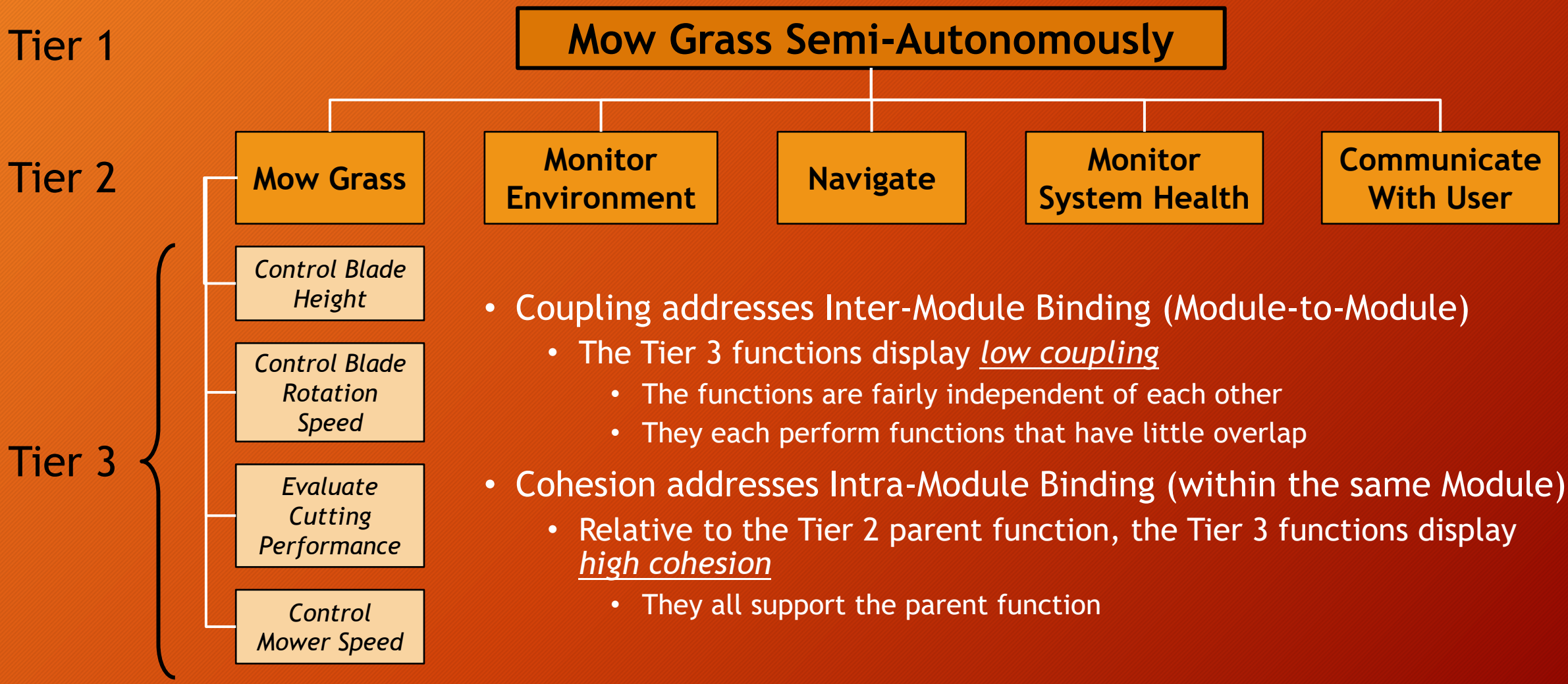
Cohesion and Coupling



- Cohesion addresses Intra-Module Binding (within the same Module)
  - Relative to Tier 1, the Tier 2 functions display *high cohesion*
    - They all support the parent function
- Coupling addresses Inter-Module Binding (Module-to-Module)
  - The Tier 2 functions display *low coupling*
    - The functions are fairly independent of each other
    - They each perform functions that have little overlap
    - Therefore, the amount of inter-module interaction (coupling) is reduced

Model adapted from https://www.mdpi.com/2624-7402/1/3/33/htm

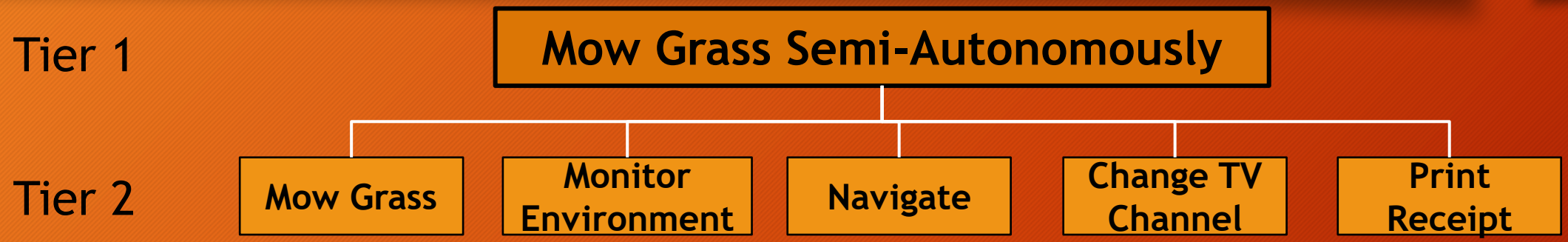
Cohesion and Coupling



- Coupling addresses Inter-Module Binding (Module-to-Module)
  - The Tier 3 functions display *low coupling*
    - The functions are fairly independent of each other
    - They each perform functions that have little overlap
- Cohesion addresses Intra-Module Binding (within the same Module)
  - Relative to the Tier 2 parent function, the Tier 3 functions display *high cohesion*
    - They all support the parent function

Model adapted from https://www.mdpi.com/2624-7402/1/3/33/htm

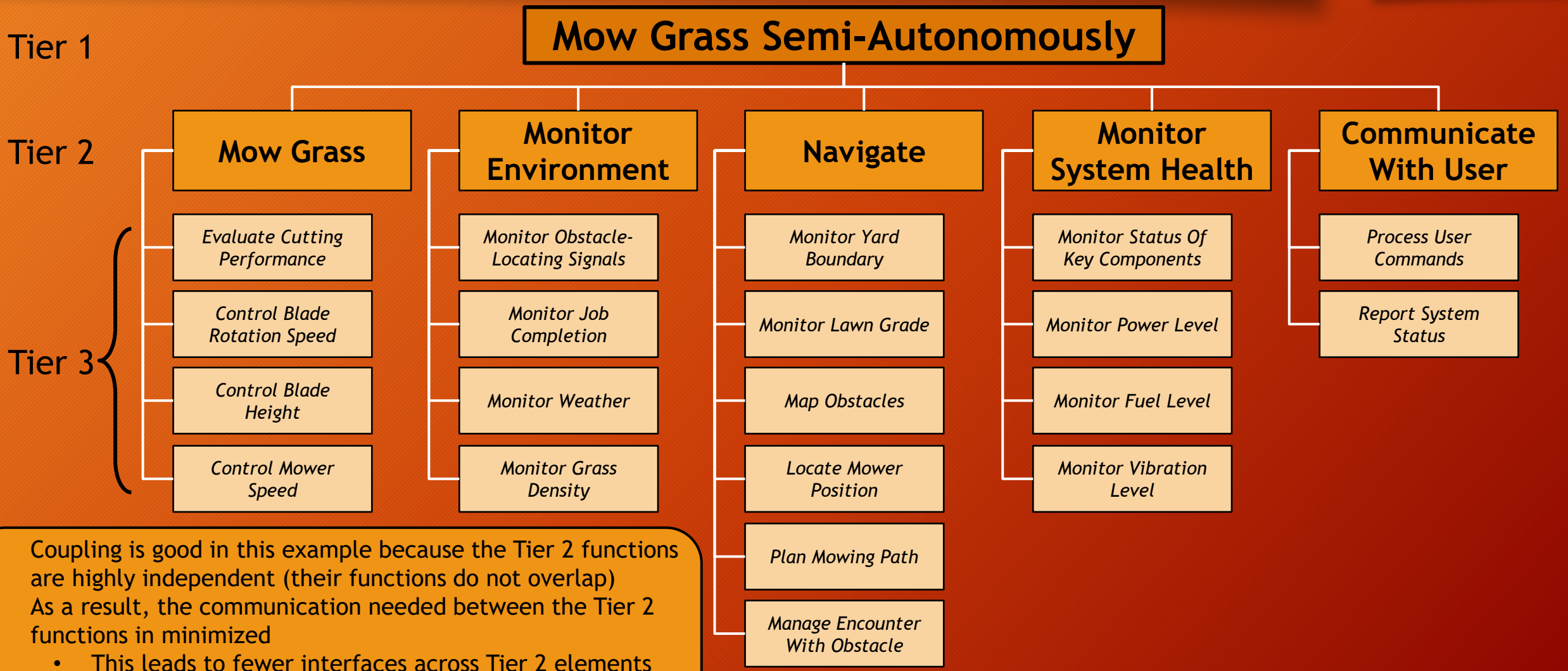
Poor Cohesion



- Do these Tier 2 functions exhibit high cohesion?
  - No, they are not all supporting the goals of the parent function

Model adapted from https://www.mdpi.com/2624-7402/1/3/33/htm

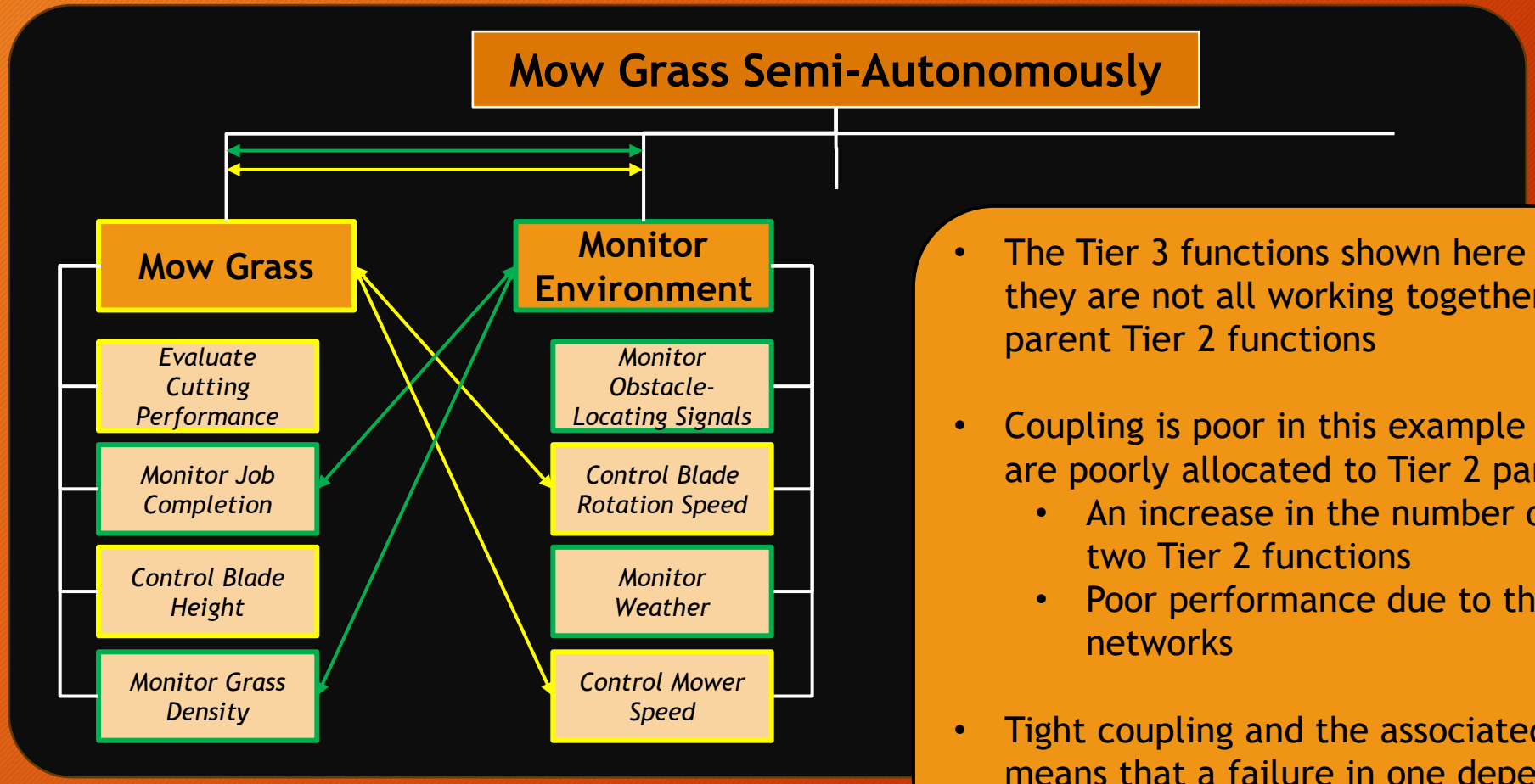
Good (Loose) Coupling



- Coupling is good in this example because the Tier 2 functions are highly independent (their functions do not overlap)
- As a result, the communication needed between the Tier 2 functions in minimized
  - This leads to fewer interfaces across Tier 2 elements
- There is also very little communication needed between the Tier 3 functions across Tier 2

Model adapted from https://www.mdpi.com/2624-7402/1/3/33/htm

Low Cohesion, Poor (Tight) Coupling



- The Tier 3 functions shown here exhibit low cohesion because they are not all working together to fulfill the goals of their parent Tier 2 functions
- Coupling is poor in this example because some Tier 3 functions are poorly allocated to Tier 2 parent functions, leading to
  - An increase in the number of interfaces between these two Tier 2 functions
  - Poor performance due to the increased data traffic across networks
- Tight coupling and the associated interdependence of functions means that a failure in one dependent function can severely impact another dependent function

Model adapted from https://www.mdpi.com/2624-7402/1/3/33/htm

Example: Multi-Layer Air Defense System

Components of the Layered Air Defense Architecture

Component elements are provided in the numbers required to provide the desired air volume coverage

Lowest Layer

Mobile Command

Anti-Aircraft Artillery (AAA)

Middle Layer

Regional Command Center

Surface-To-Air Missile (SAM)

Upper Layer

Theater Command Center

Strategic Missile System (SMS)

This example is presented by describing the system structural components
This is done due to the fact that the functionality of the system closely resembles the structural identity of its components

Battlefield Layout of Defense Equipment Configuration

- Shown is a completely notional and fictional air defense configuration
- Additional equipment such as radar sensors not shown to avoid overcomplicating the story

Upper Level Weapon Control
Upper-to-Middle Command Link
Middle Level Weapon Control
Middle-to-Lower Command Link
Lower Level Weapon Control

Incoming Air Threat

Hierarchical Configuration of Three-Layer Structure

HIGH COHESION – All Subordinate Units Support ONLY their Parent Unit
LOOSE COUPLING – There are NO cross-organizational communications

Top Layer

Middle Layer

Lower Layer

- In a truly centralized configuration such as this, EVERYONE is bound to follow their Commander's instructions
- NOONE takes any independent action
- The Commanders direct the engagement by issuing commands to subordinates
- This is an effective arrangement until a commander or a command link is taken out
- At this point, the disconnected elements are useless, as they cannot make decisions on their own

For those curious, this is only one simplified view of a complete system
See next slide for a larger perspective

Additional Components of the Layered Air Defense Architecture

Component elements are provided in the numbers required to provide the desired air volume coverage

Lowest Layer

Mobile Command

Short-Range Radar (AAA)

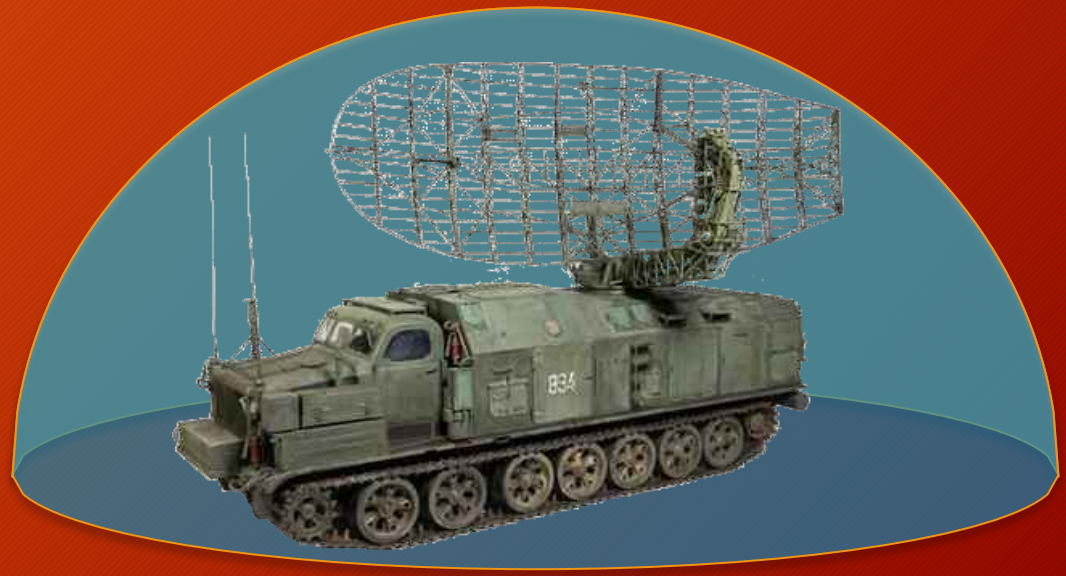
Middle Layer

Regional Command Center

Medium-Range Radar (SAM)

Upper Layer

Theater Command Center



Long Range Radar (SMS)

A Complete Air Defense System Would Have Many More Elements

- The Air Defense System example shown is grossly oversimplified to keep the detail under control and the content understandable
- An actual such system would require many, many more elements to provide realistic capability
- This includes:
  - The weapon systems already addressed, PLUS ...
  - Sensor systems needed to detect and track the incoming air threat
  - Air-based air defense systems (aircraft assigned to attack the incoming air threat)
  - Electronic Warfare Countermeasures equipment to jam the enemy's sensors
  - Electronic Warfare Counter-Countermeasures equipment to jam the enemy's jamming equipment
  - Communication units to connect all the components indicated
  - The number of all these elements discussed needed to provide complete coverage of the defendable area

This never-ending counter-counter-countermeasures cat-and-mouse game has been ongoing in modern warfare ever since electronics entered the battlefield in WW II

- In additional, further complications in a real system arise due to
  - Many, many different versions of the basic equipment due to improvements made to the systems over time
  - Reduced performance levels due to equipment requiring tuning or maintenance
- This all being said to ensure that the student does not get the wrong impression that these kinds of systems are simple, clean, and fully-functional all the time (they are not)

Measuring Cohesion

Measuring Cohesion

- The degree of cohesion of a module can be expressed by the following formula:

$$coh(m,k) = \frac{N_m^k}{N^k + (N_m - N_m^k)}$$

The "key" is some aspect of system functionality which forms the basis for establishing functional commonality among functional elements
Functional elements of the same "key" should ideally form functional modules

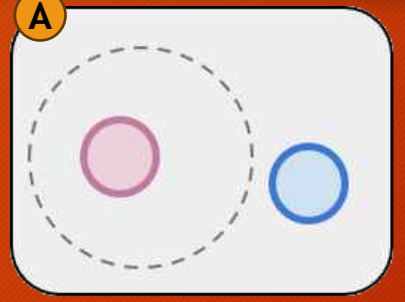
  - where
    - coh is the measure of cohesion as a function of
      - m, the module under investigation
      - k, the "key" or functional commonality of interest among the functional elements
    - N_m^k is the number of functional elements cohesive by the key within the module
    - N^k is the total number of functional elements in the entire system cohesive by the key
    - $N_m - N_m^k$ is the number of functional elements not cohesive by the key inside the module
- Obviously, the maximal cohesion of a module is 1
- This is what we strive for

Examples of Measuring Cohesion

$$coh(m,k) = \frac{N_m^k}{N^k + (N_m - N_m^k)}$$

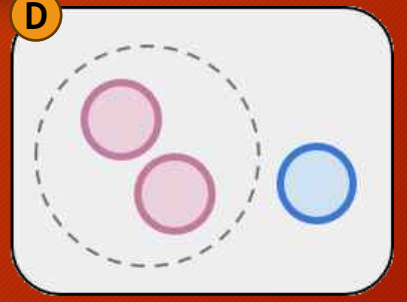
- Here are examples of usage of the formula for determining functional cohesion of a module of interest
- In these examples, the module under consideration is shown by the dashed circle
- In example A, the cohesion of the module is 1, although it is not recommended to have functional modules that comprise only a single functional element
- In example B, a functional element of the same key exists outside of the module of interest, indicating that the module is less than ideally cohesive
- In example C, the condition is worse since a functional element of a foreign key exists within the module of interest
- In example D, we have achieve a more ideal solution, since all functional elements of a like key exist within a single module
- In example E, cohesion is again compromised by the existence of a functional element of a foreign key within the module of interest
- In example F, cohesion is the worst of the examples since not only does a functional element of a foreign key exist within the module of interest, but also, a functional element of the desired key exists outside of the module of interest

A



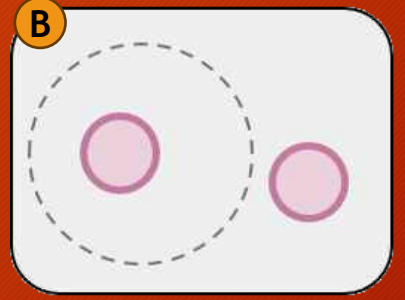
1 / (1 + 1 - 1) = 1

D



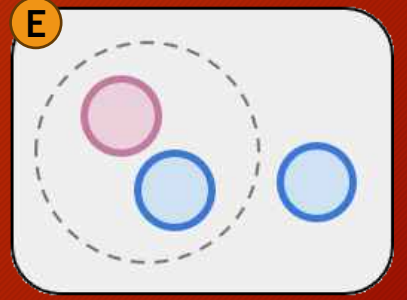
2 / (2 + 2 - 2) = 1

B



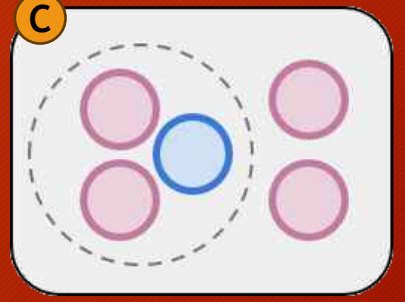
1 / (2 + 1 - 1) = 0.5

E



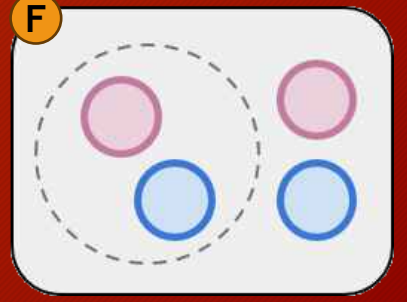
1 / (1 + 2 - 1) = 0.5

C



2 / (4 + 3 - 2) = 0.4

F



1 / (2 + 2 - 1) = 0.33...

Measuring Coupling and Instability

Measuring Coupling

- Coupling metrics help engineers determine the complexity of their architecture based on the dependencies between modules
- These metrics reveal how the modules are connected, the strength of their dependencies and the stability of the overall design
- Tight coupling makes it difficult to alter a single module without causing direct changes to others
- Updates and tests are all complicated by tight coupling
- A system becomes loosely coupled if individual modules require little or no knowledge of the surrounding modules to operate
- As such, teams can build systems with modular components by using functional modules that can act and evolve independently
- This measurement technique is not absolute, but can be used as a tool to measure the relative degree of coupling between alternative design approaches

The Fenton and Melton Metric

- The Fenton and Melton metric is commonly used to measure the degree of coupling, C, between modules a and b:

$$C(a, b) = i + n / (n + 1)$$

- The variable n represents the actual number of direct interconnections that exist between modules a and b
- The variable i indicates the highest level of coupling type that exists between modules a and b
  - Engineers can determine i by examining each of those modules and identifying the tightest dependency relationship, with 0 representing the lowest level of dependency and 5 representing the highest (see next slide)

Dependency Scale

- Fenton and Melton identified the following scale for measuring coupling

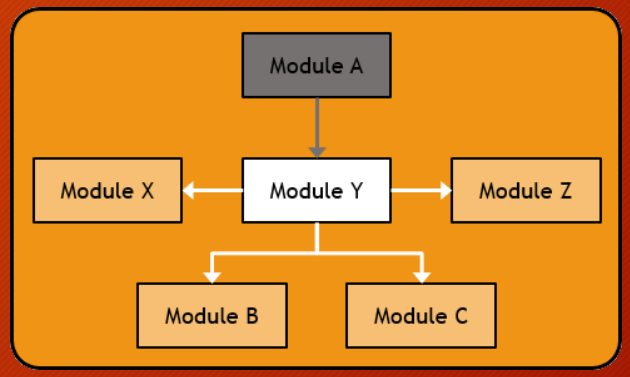
Measure of Coupling (i)	Coupling Type	Meaning for Software	Meaning for Systems
5	Content Coupling	Module X refers to the inside of Module Y It branches into, changes data, or alters a statement in Module Y	Module X affects the functional behavior of Module Y
4	Common Coupling	Modules X and Y refer to the same global data This type of coupling is undesirable because if the format of the global data needs to be changed then all common coupled modules must also be changed	Modules X and Y refer to the same global data
3	Control Coupling	Module X passes a parameter to Module Y with the intention of controlling its behavior	Module X passes a parameter to Module Y with the intention of controlling its behavior
2	Stamp Coupling	Modules X and Y accept the same record type as a parameter This type of coupling may manufacture an interdependency between otherwise unrelated modules	Modules X and Y accept the same record type as a parameter This type of coupling does not constitute a strong interdependency
1	Data Coupling	Modules X and Y communicate by parameters, each one being either a single data element or a homogeneous set of data items that do not incorporate any control element This type of coupling is necessary for any communication between modules	Modules X and Y communicate by parameters This type of coupling is necessary for any communication between modules
0	No Coupling	Modules X and Y have no communication whatsoever They are totally independent	Modules X and Y have no communication whatsoever They are totally independent

Measuring Instability

- Instability is a metric used to measure the relative susceptibility of a component to breaking changes. Instability indicates that a software class, package, subsystem or other given module will be significantly impacted by changes elsewhere
- Instability is calculated according to the following formula, where I represents instability:
  - I = Ce / (Ce + Ca)
- To accurately calculate instability, engineers must also delineate the nature of a module's dependency relationships by measuring two major types of coupling
  - Efferent Coupling
  - Afferent Coupling

Efferent Coupling and Afferent Coupling

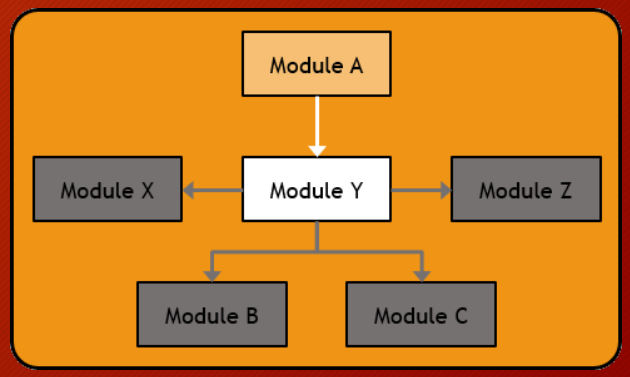
- Efferent Coupling
  - Efferent coupling is a measure of the number of other modules that a given module (Module Y in the example) depends on to operate
  - A sizable level of efferent coupling indicates that a module may be difficult to observe, reuse, test, and maintain
  - For instance, it's unlikely that engineers will be able to update a module with a great deal of efferent coupling without also updating each module it depends on
  - In most cases, engineers will want to decompose these dependencies, which will help instill the single functional responsibility principle needed to loosen coupling



Arrows indicate a dependency

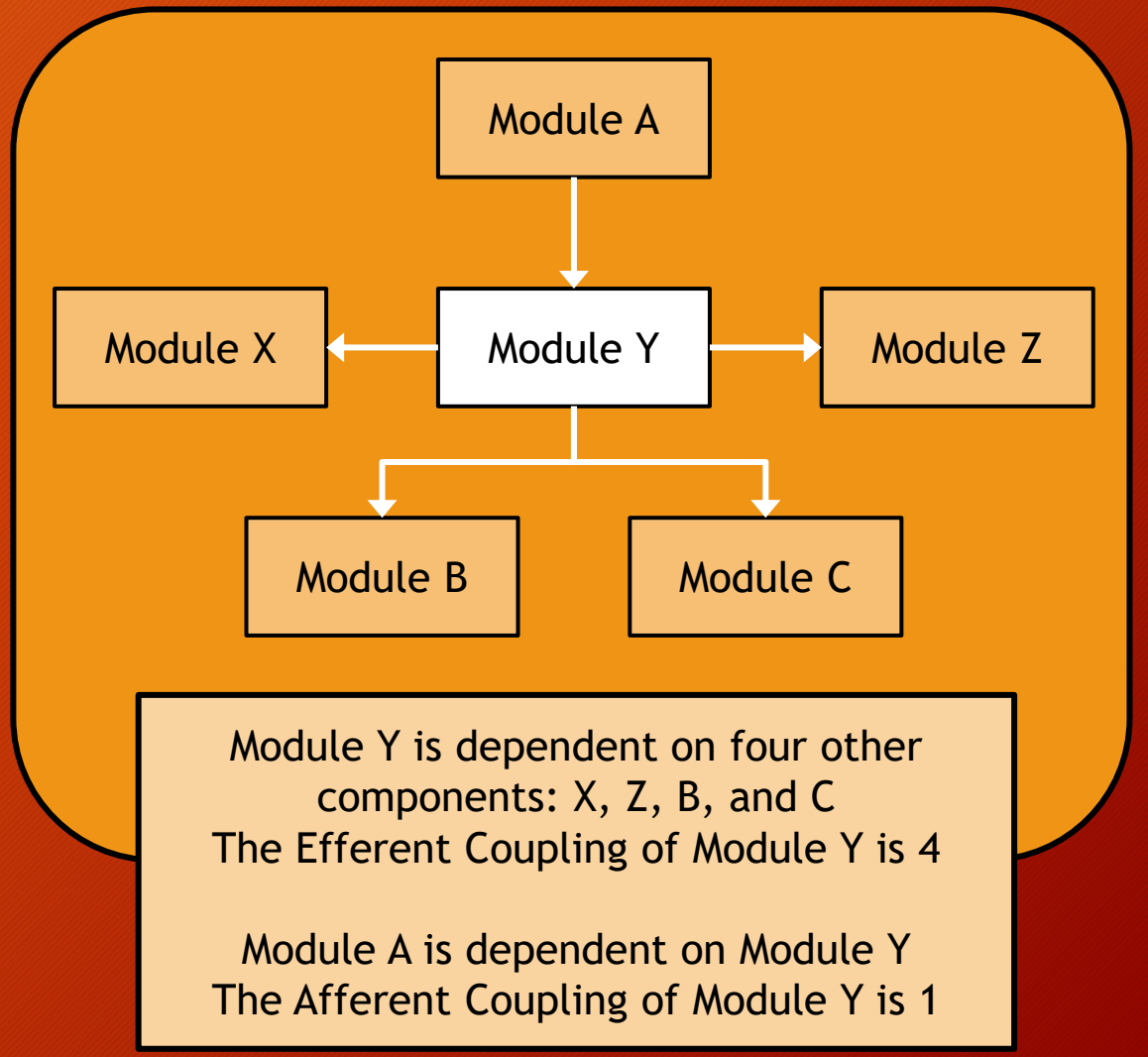
Module at tail end of arrow(s) depends on module(s) at head end

- Afferent Coupling
  - Afferent coupling measures the number of other modules that are dependent on a given module (Module Y in the example)
  - While it still may make a module difficult to change, high afferent coupling is not necessarily a bad thing, as it can invariably occur in certain areas of a system
  - However, it is a poor architecture practice to maintain extensive afferent coupling across a large number of system functions
  - Ideally, a module with bundles of afferent coupling should remain small and take on as few functional responsibilities (requirements) as possible



Efferent and Afferent Coupling Example

- Here, the efferent coupling (Ce) and the afferent coupling (Ca) of Module A, is illustrated
- As the diagram shows, Module Y is dependent on four other modules: X, Z, B, and C
- Meanwhile, there is a single module - Module A - that depends on Module Y
- In this case, the efferent coupling of Module Y has a value of 4, while the value of its afferent coupling is 1



Instability Example

- The instability formula will produce a value for I that ranges from 0 to 1
- Modules are considered to be generally stable when this calculation places the value of I closer to 0
- Modules with tighter coupling will usually produce a value closer to 1, which indicates that it is highly susceptible to changes that cause the module to fail
- Using the example class from earlier, we can calculate the instability of Module A by:
  - $I = 4 / (4 + 1)$
- Hence, the instability of Module A is 0.8, indicating that it is particularly susceptible to changes that could cause the module to fail
- While the module is not completely incapable of independent changes, it still maintains enough dependencies to prohibit modularity

References

1. Pagade, G (2021). Difference Between Cohesion and Coupling. Retrieved from https://www.baeldung.com/cs/cohesion-vs-coupling
2. Nelson, J (2014). Coupling And Cohesion: Overview. Retrieved from https://github.com/jn123456789/coupling_and_cohesion
3. Java T Point (2022). Coupling and Cohesion. Retrieved from https://www.javatpoint.com/software-engineering-coupling-and-cohesion
4. Samares Engineering (2020). Coupling optimization of logical architecture using genetic algorithm. Retrieved from https://www.samares-engineering.com/en/2020/07/31/part-5-coupling-optimization-of-logical-architecture-using-genetic-algorithm/
5. Sandoval, K (2020). The Difference Between Tight Coupling and Loose Coupling. Retrieved from https://nordicapis.com/the-difference-between-tight-coupling-and-loose-coupling/#:~:text=Tight%20Coupling%20is%20the%20idea,interface%2C%20or%20a%20specific%20frontend.&text=In%20a%20loosely%20coupled%20system,are%20detached%20from%20each%20other

References (continued)

6. Tulka, T (2020). How Cohesion and Coupling Correlate. Retrieved from https://blog.ttulka.com/how-cohesion-and-coupling-correlate/
7. Patterson, A., Yang, Y., Norris, W. (2019). Development of User-Integrated Semi-Autonomous Lawn Mowing Systems: A Systems Engineering Perspective and Proposed Architecture. Retrieved from https://www.mdpi.com/2624-7402/1/3/33
8. Kanjilal, J. (2020) The Basics of Software Coupling Metrics and Concepts. Retrieved from: https://www.techtarget.com/searchapparchitecture/tip/The-basics-of-software-coupling-metrics-and-concept
9. Fenton, N., and Melton, A. (1990) Deriving Structurally Based Software Measures. Retrieved from: https://sci-hub.st/10.1016/0164-1212(90)90038-n

Image Sources

- AAA System
  - http://www.armaco.bg/userfiles/images/ARMAAGZU23-n.png
- Regional Defense System
  - https://www.ausairpower.net/PVO-S/5P73-Launcher-Deployed-MiroslavGyurosi-1S.jpg
- Long Range Defense System
  - https://en.euractiv.eu/wp-content/uploads/sites/2/2022/12/ree.jpg
- Mobile Comms Vehicle
  - https://api.army.mil/e2/c/images/2018/07/26/525190/original.jpg
- Regional Command Center
  - https://api.army.mil/e2/c/images/2015/09/11/408981/original.jpg
- Theater Command Center
  - https://www.afcea.org/signal-media/crown-jewel-program-modernizes-air-operations
- Terrain
  - https://www.aiviagroup.com/mapping-steep-terrain-and-dense-vegetation/
- Air Threat Aircraft
  - https://www.popsci.com/wp-content/uploads/2021/04/21/6557886.jpeg
- Long Range Radar
  - https://mavink.com/explore/1S12-Long-Track
- Medium-Range Radar
  - https://www.falcon-lounge.com/falcon-bms-essentials/threats-guide/surface-to-air-missile-sa-3/
- Short-Range Radar
  - https://www.pinterest.com/pin/159314905555255037/