

**2805ICT System and Software Design**  
**3815ICT Software Engineering**  
**7805ICT Principles of Software Engineering**

Lecture one: Software Engineering and Design Principles

Dr Larry Wen



# Contents

- About me
- About the course
- What is Software Engineering
- Software Development Life Cycle
- General Principles
- Cohesion & Coupling

# About me

- Name: Dr Larry Wen
- Office: N44 2.39 (Nathan campus)
- Phone: 55042
- Email: [l.wen@griffith.edu.au](mailto:l.wen@griffith.edu.au)
- Consultation time: Thursday 10am-12pm (in my office)
- Lecture time: (Week 1 – Week 11) 3pm-5pm
- Lab time: (Week 2 – Week 11) (10 times)
  - Nathan: Monday 11am-1pm(N79\_4.19) 2pm-4pm(N79\_4.18)
  - GC: Thursday 2pm-4pm(G06\_1.29) 4pm-6pm(G06\_1.29)
  - Online: Wednesday 1pm-3pm







# Course topics

1. Software Engineering and Design Principles
2. Requirement Engineering
3. UML and OO Design
4. Software Architecture
5. Software Architecture Documentation
6. Architectural Patterns
7. Design Patterns
8. Design Strategies
9. Software Testing and Behaviour Engineering
10. Other Design Diagrams
11. Formal Method and Model Checking



## Study activities

- Every teaching week (total 10 hours)
  - Two hour online lecture
  - Two hour online/on campus lab
  - Three hour self study (review and/or extra reading)
  - Three hours for assignment



## Assessment plan

- Three assessed labs (individual, 10% each)
- Three group assignments (group up to four students, 20% for the first two and 30% for the final assignment)
- Interview – Every student needs to schedule a 10-minutes interview with the workshop teacher in one of the last two labs. The interview results might affect your marks in the assessed labs and assignments.

# Course Schedule

Teaching Weeks	Tuesday Lecture Online 3pm-5pm	Monday Nathan Lab	Wednesday Online Lab	Thursday GC Lab	Assignment Due	
Week 1 18/7 - 22/7	Software Engineering and Design Principles	No Labs in this week				
Week 2 25/7 - 29/7	Requirement Engineering	Lab 1 Software Engineering				
Week 3 1/8 - 4/8	OO Design and UML	Lab 2 Requirement Engineering				
Week 4 8/8 - 12/8	Software Architecture	Lab 3 (Assessed Lab) UML and first stage assessment				
Trimester Break						
Week 5 22/8 - 26/8	Document Software Architecture	Lab 4 Software Architecture			Milestone 1 due on 26/8	
Week 6 29/8 - 2/9	Architectural Design Patterns	Lab 5 Architecture Documentation				
Week 7 5/9 - 9/9	Design Patterns	Lab 6 Architectural Design Patterns				
Week 8 12/9 -16/9	Design Tactics	Lab 7 (Assessed Lab) Design Patterns and second stage assessment				
Week 9 19/9 -23/9	Software Testing and Behaviour Engineering	Lab 8 Design Tactics			Milestone 2 due on 23/9	
Week 10 26/9 -30/9	Other Design Diagrams	Lab 9 Testing and BECIE (Interview)				
Week 11 3/10 - 7/10	Formal Methods	Lab 10 (Assessed Lab) Design Diagram, Formal Method and third stage assessment (Interview)				
Week 12 10/10 - 14/10	No Lecture and Labs, review week				Overall project due on 14/10	



## About the group

- A group may contain up to four students. One student group is possible.
- Mixed groups (students from different campus, taking different course codes) are allowed.
- By default, all members in one group will receive the same mark for the same submission. However, the assessor may adjust the marks if some member's contribution is too low or the quality is too poor.
- **Each group work only need to be submitted once by one student.** However, all students' ids and names must be clearly marked in the submission.





## More about the submission

- Plagiarism will not be tolerated, and will be prosecuted based on university's policy.
- All assessment items must be submitted on time. Unless has been approved before the submission deadline, marks will be deducted for later submission based on university's policy.



# Contents

- About me
- About the course
- **What is Software Engineering**
- Software Development Life Cycle
- General Principles
- Cohesion & Coupling



# What is Software

*Software is:*

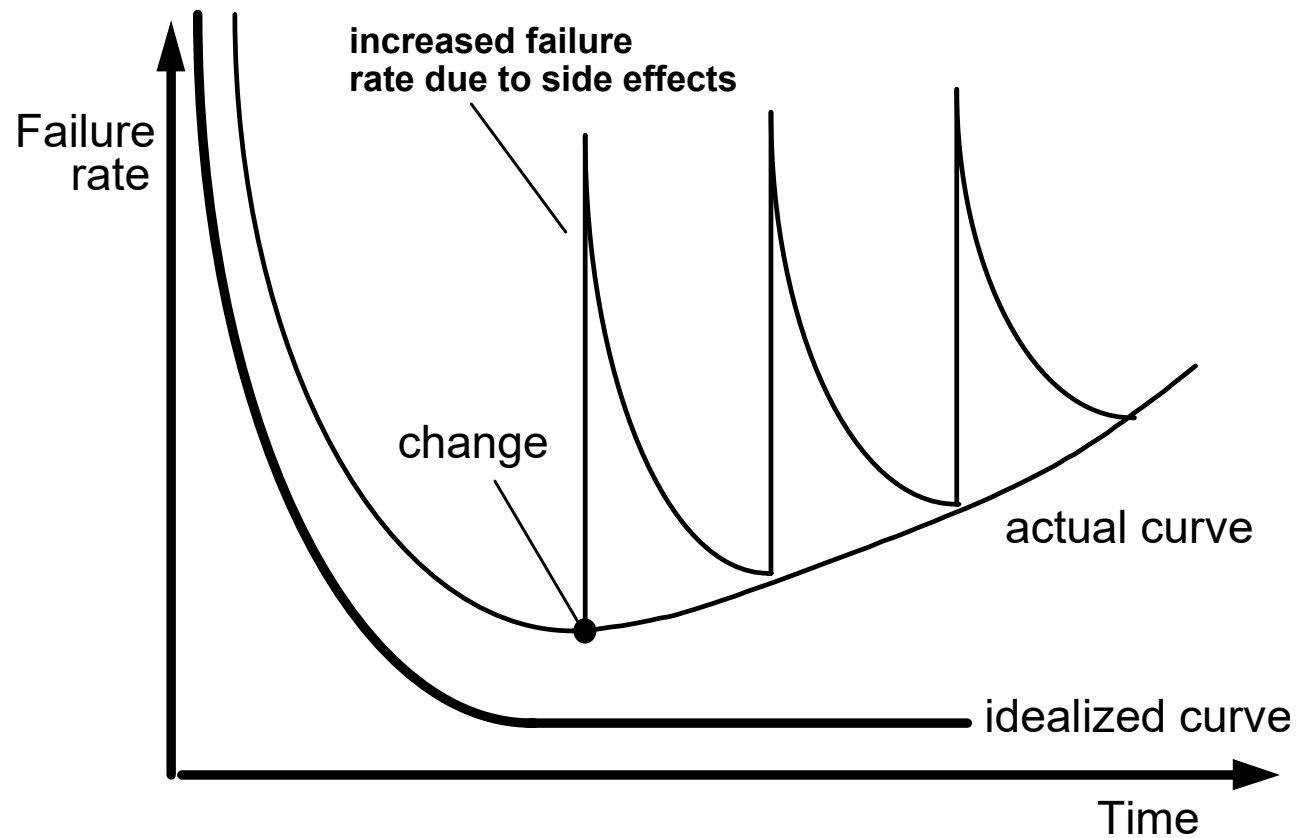
- (1) **instructions** (computer programs) that when executed provide desired features, function, and performance;*
- (2) **data (in) structures** that enable the programs to adequately manipulate information and*
- (3) **documentation** that describes the operation and use of the programs.*



# Software is special

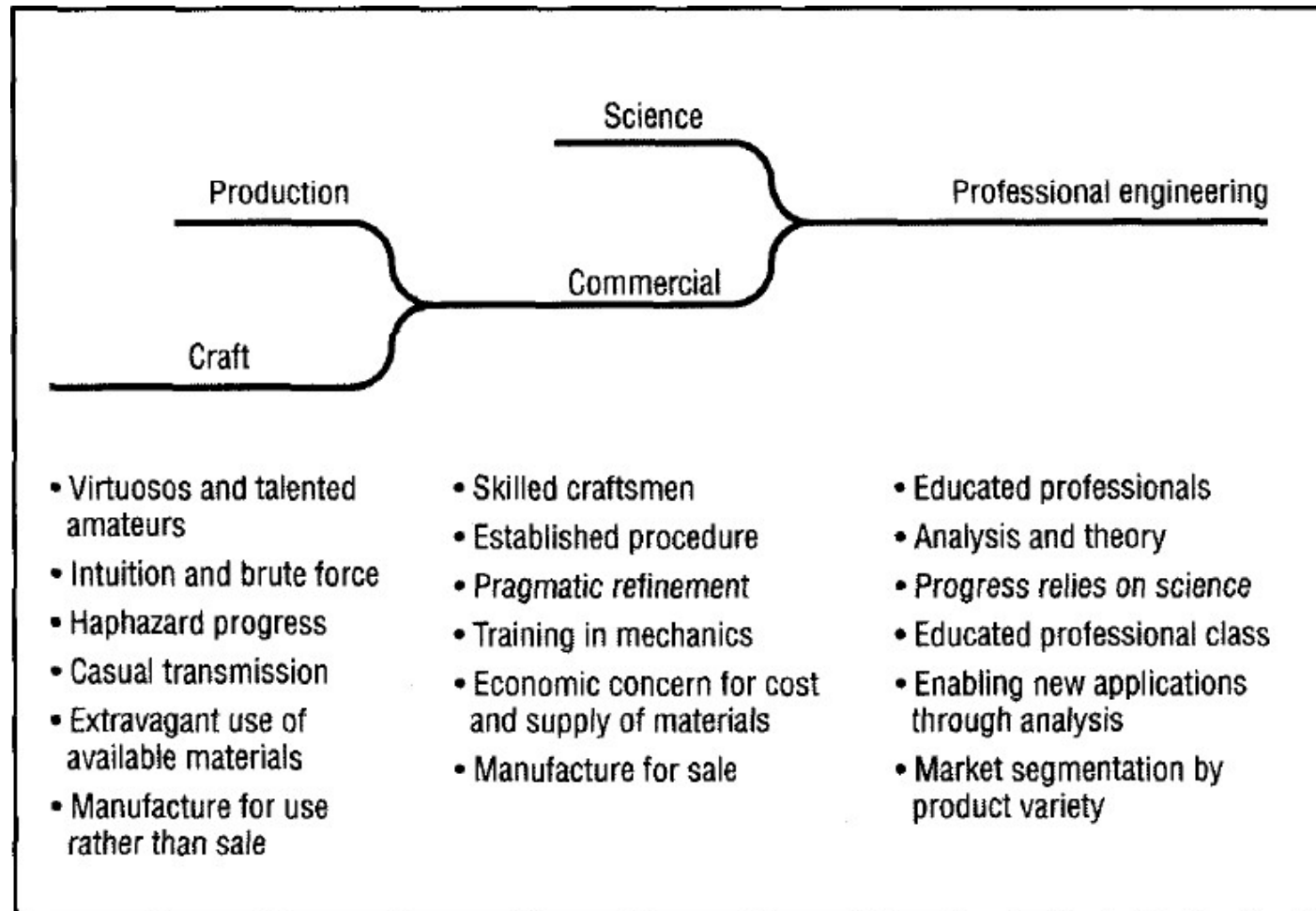
- *Developed or engineered, not manufactured*
- *Software doesn't "**wear out.**"*
- *custom-built.*

# Wear vs. Deterioration





# What is Engineering





# Properties of Engineering

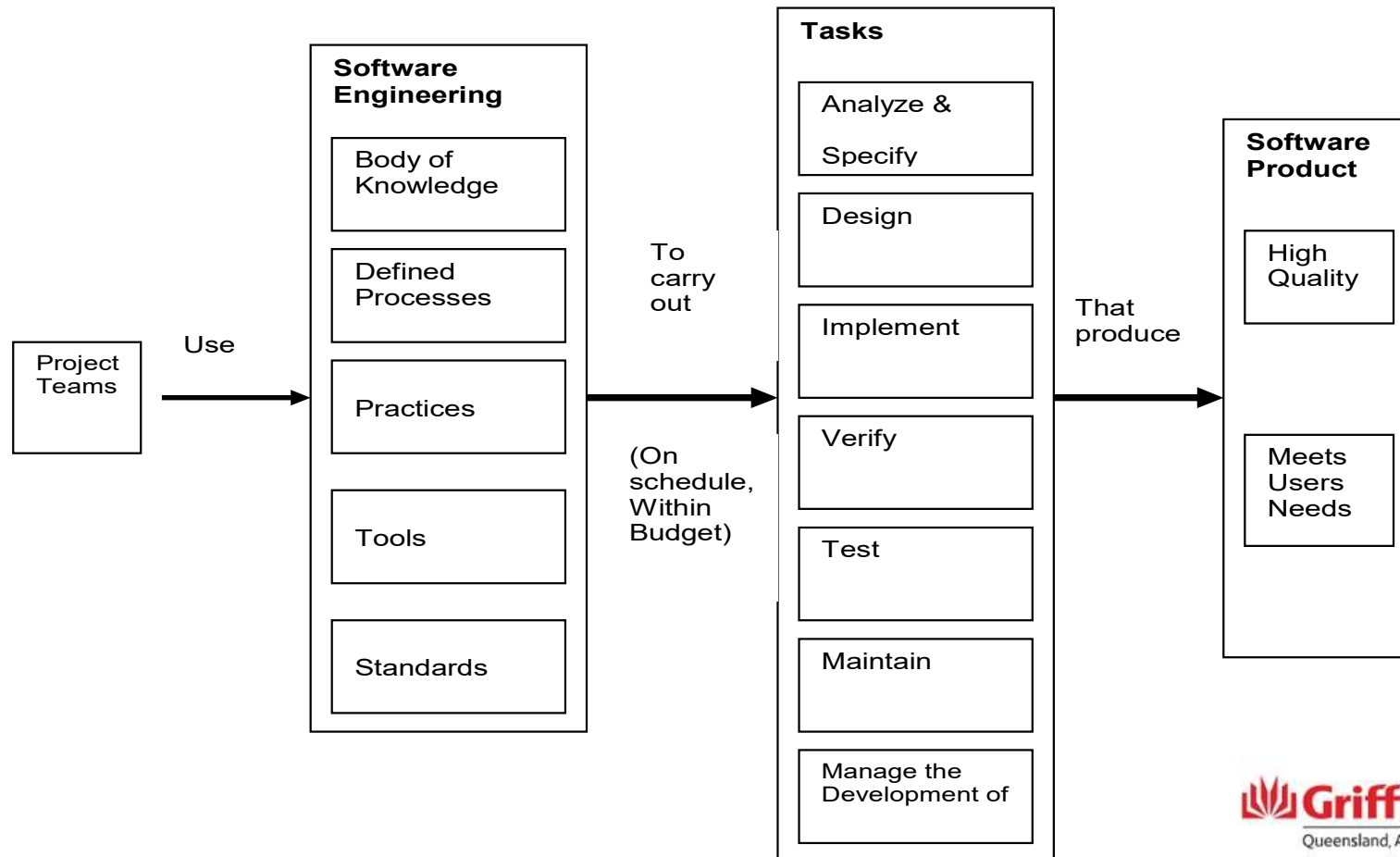
- Large scale
- Quality
- Commercial (market)
- Cost effective (profit)
- Practical
- Management and human issue
- Standards and regulation



# Software Engineering

- IEEE definition
  - the application of a **systematic, disciplined, quantifiable** approach to the **development, operation, and maintenance** of software, and
  - the **study** of these approaches; that is, the application of engineering to software.

# What is Software Engineering






# Kipling's Six Friends

- What?
- Why?
- When?
- How?
- Which?
- Who?





# Factors Affecting the Quality of a Software System

- **Complexity:**

- The system is so complex that **no single programmer** can understand it anymore
- The introduction of one **bug fix** causes another bug

- **Change:**

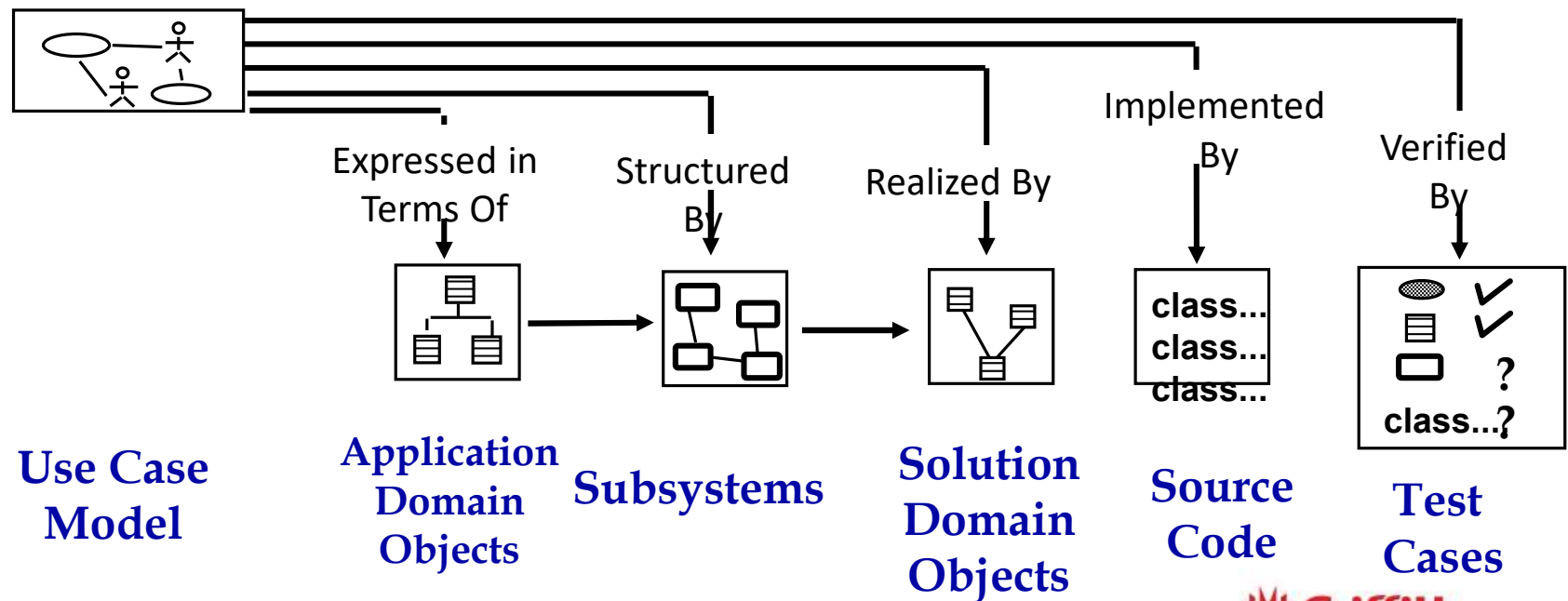
- The “Entropy” of a software system increases with each change: Each implemented change **erodes the structure** of the system which makes the next change even more expensive (“Second Law of Software Dynamics”).
- As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base. (**eventually a system will die**)



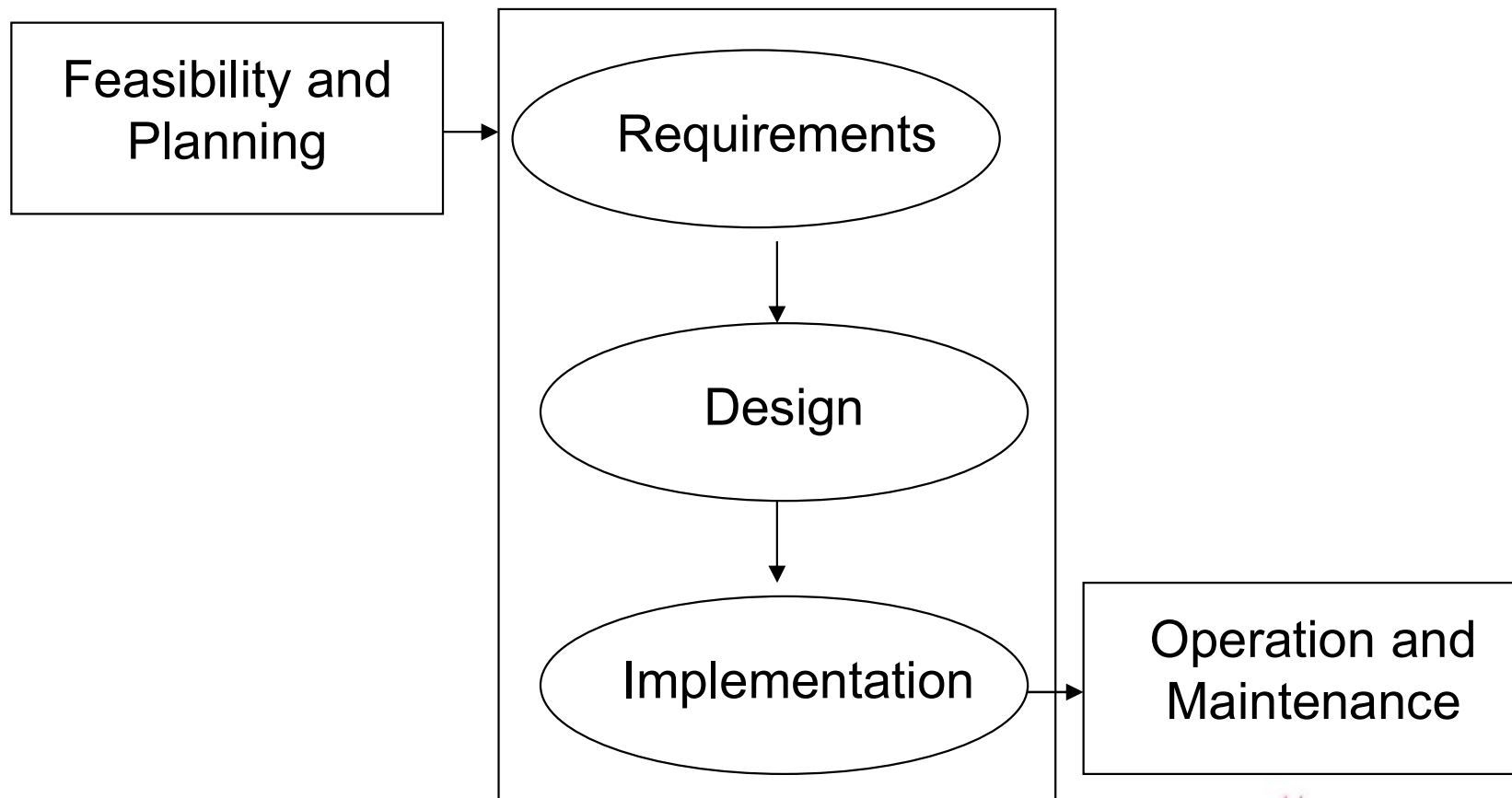
# Contents

- About me
- About the course
- What is Software Engineering
- **Software Development Life Cycle**
- General Principles
- Cohesion & Coupling

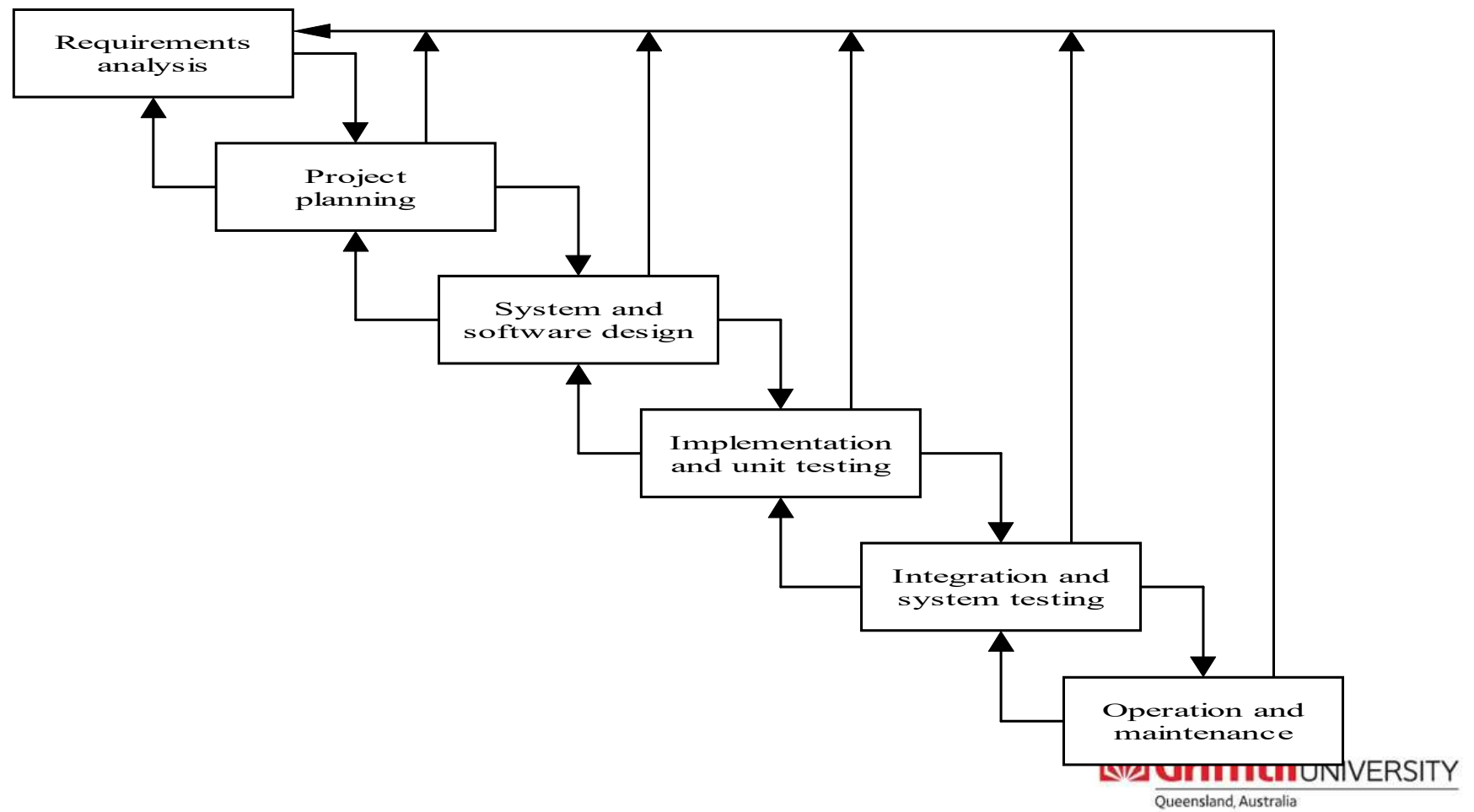
# Common Software Lifecycle Activities



# Simplified Software Process

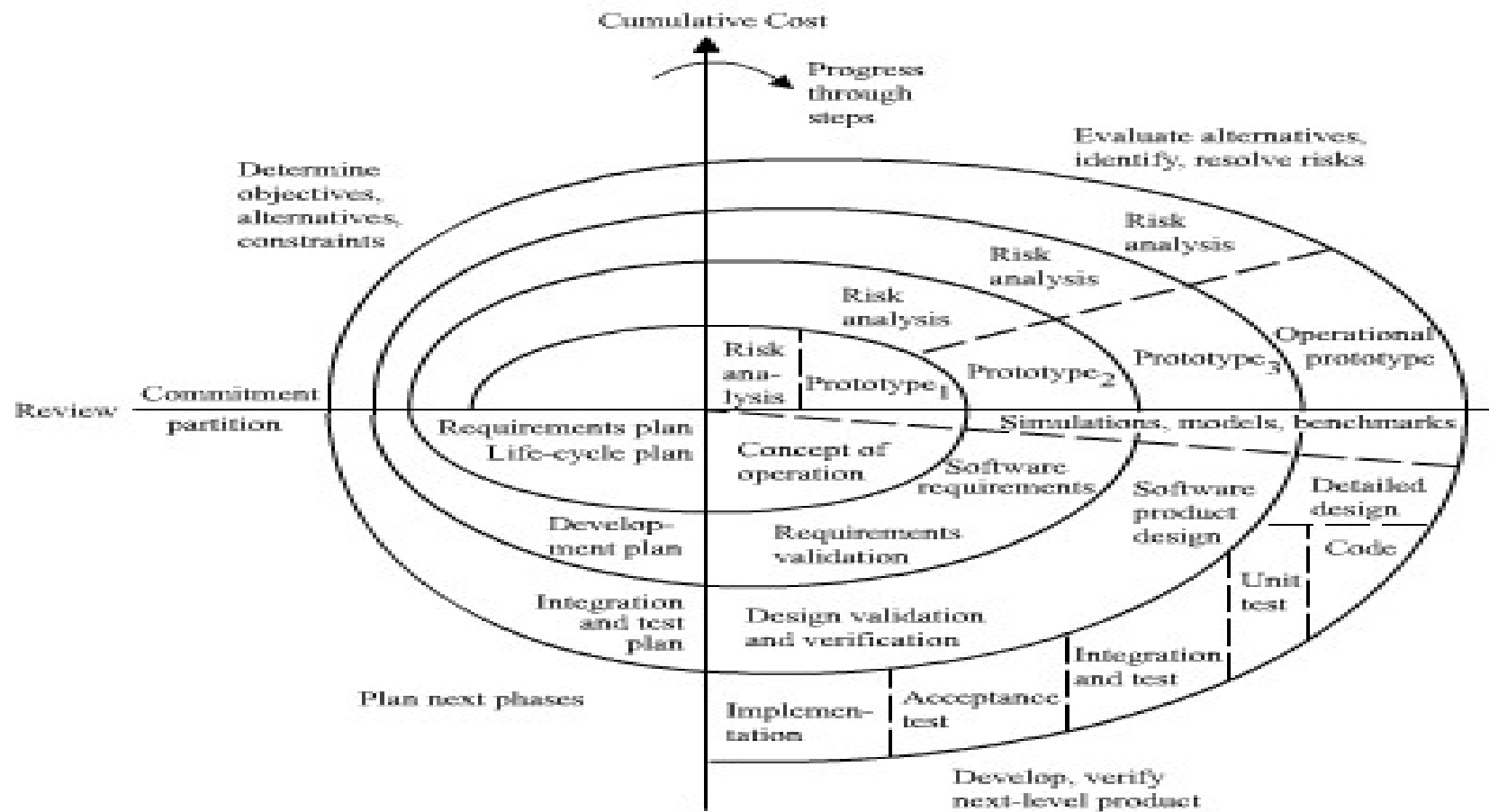


# Waterfall Model





# Spiral Model



**The Spiral Model**



## Other Lifecycle Model

- Prototyping
- Iteration
- V model
- W model
- Agile approaches



# Contents

- About me
- About the course
- What is Software Engineering
- Software Development Life Cycle
- **General Principles**
- Cohesion & Coupling



# General Principles

- Software Engineering
  - Different from other engineering disciplines such as civil or mechanical, there is no one set of “universal principles” in software engineering that is agreed upon by everyone.
  - Neither is there any “law” of software engineering such as Newton’s law of motion in physics.
    - There are, however, several that are well received and respected.
      - Boehm’s Principle
      - Davis’s Principles
      - Royce’s Principles
      - Wasserman’s Concepts



## Seven Basic Principles of Software Engineering (Boehm, 1983)

1. manage using a phased life-cycle plan.
2. perform continuous validation.
3. maintain disciplined product control.
4. use modern programming practices.
5. maintain clear accountability for results.
6. use better and fewer people.
7. maintain a commitment to improve the process.





# Davis's 15 Principles (1994)

1. Make quality number 1.
2. High-quality software is possible.
3. Give products to customers early.
4. Determine the problem before writing the requirements.
5. Evaluate design alternatives.
6. Use an appropriate process model.
7. Use different languages for different phases.
8. Minimize intellectual distances.
9. Put techniques before tools.
10. Get it right before you make it faster.
11. Inspect code.
12. Good management is more important than good technology.
13. People are the key to success.
14. Follow with care.
15. Take responsibility.



# Royce's More “Modern” Set of Principles

1. Base the process on an **architecture first** approach.
2. **Establish iterative process – address risk early.**
3. Emphasize component-based development to reduce effort.
4. Establish change management.
5. Use round-trip engineering – a form of iterative process.
6. Use model-based and machine-processable notations for design.
7. Establish process for quality control and project assessment.
8. Use approach that allows artifacts to be demonstrated early.
9. Plan to have incremental releases.
10. Establish a configurable process to suit the needs.



# Wasserman's Fundamental Concepts(1996)

1. Abstraction (a form of simplification)
2. Analysis and design methods and notation
3. User interface prototyping
4. Modularity and architecture
5. Reuse
6. Life cycle and process
7. Metrics
8. Tools and integrated environment



# Agile Manifesto

- **Individuals** and interactions over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- Responding to **change** over following a plan



# Contents

- About me
- About the course
- What is Software Engineering
- Software Development Life Cycle
- General Principles
- **Cohesion & Coupling**



# Walling in & Walling out

- Object oriented analysis and design is essential about building systems with high degrees of cohesion (within) and low amount of coupling (without).
- **Cohesion** – The degree to which elements of a component or object inter-depend.
- **Coupling** - The degree to which components or objects depend on one another.
  - Generally loose coupling is desirable for good software engineering.
- Finding a balance between what to wall into an object, component or module is the essence of design.



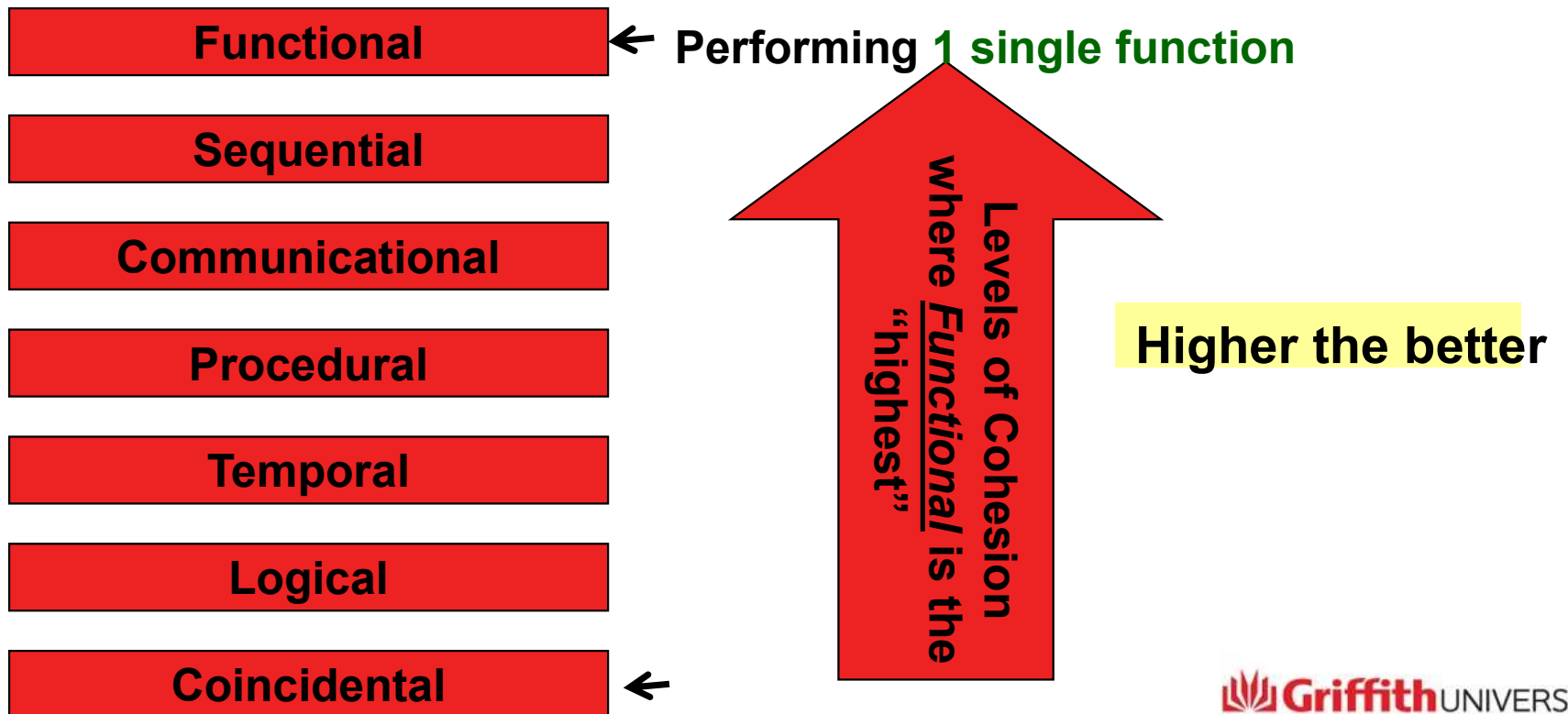


## A Little “Deeper” on Good Design Attributes

- Easy to:
  - Understand
  - Change
  - Reuse
  - Test
  - Integrate
  - Implement/Code
- Believe that we can get many of these “easy to’s” if we consider:
  - Cohesion
  - Coupling

# Cohesion

- **Cohesion** of a unit, of a module, of an object, or a component addresses the attribute of “degree of relatedness” within that unit, module, object, or component.





# Types of Cohesion

- **Functional Cohesion**

- A and B are part of a single functional task. This is very good reason for them to be contained in the same procedure.

- **Sequential Cohesion**

- Module A outputs some data which forms the input to B. This is the reason for them to be contained in the same procedure.

- **Procedural Cohesion**

- Procedural Cohesion occurs in modules whose instructions although accomplish different tasks yet have been combined because there is a specific order in which the tasks are to be completed.



## Types of Cohesion (cont.)

- **Temporal Cohesion**

- Module exhibits temporal cohesion when it contains tasks that are related by the fact that all tasks must be executed in the same time-span.

- **Logical Cohesion**

- Logical cohesion occurs in modules that contain instructions that appear to be related because they fall into the same logical class of functions.

- **Coincidental Cohesion**

- Coincidental cohesion exists in modules that contain instructions that have little or no relationship to one another.

# Coupling

- Coupling addresses the attribute of “degree of interdependence” between software units, modules, or components.

Content Coupling

← Accessing the internal data or procedural information

Common Coupling

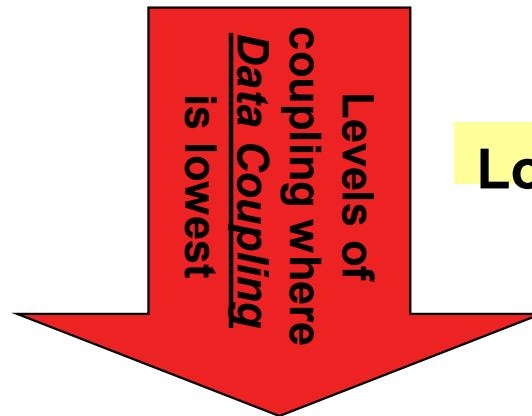
Control Coupling

Stamp Coupling

Data Coupling

No Coupling

← Ideal, but not practical

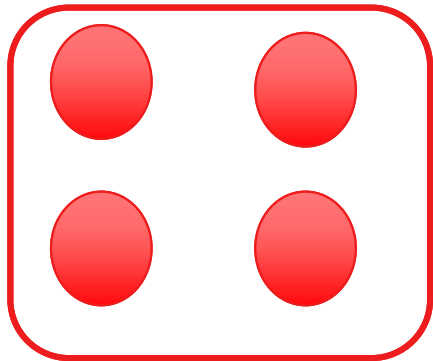


Lower the better

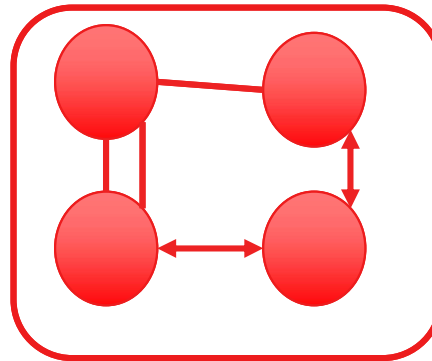
← Passing only the necessary information

# Module Coupling

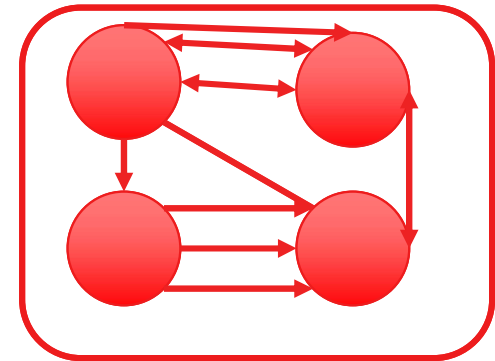
- Coupling is the measure of the degree of interdependence between modules.



No coupling



loosely coupled  
Some dependencies



Highly coupled  
Many dependencies





# Type of Coupling

- **Data coupling**
  - The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data. Other than communicating through data, the two modules are independent.
- **Stamp coupling**
  - occurs between module A and B when complete data structure is passed from one module to another.



## Types of Coupling (cont)

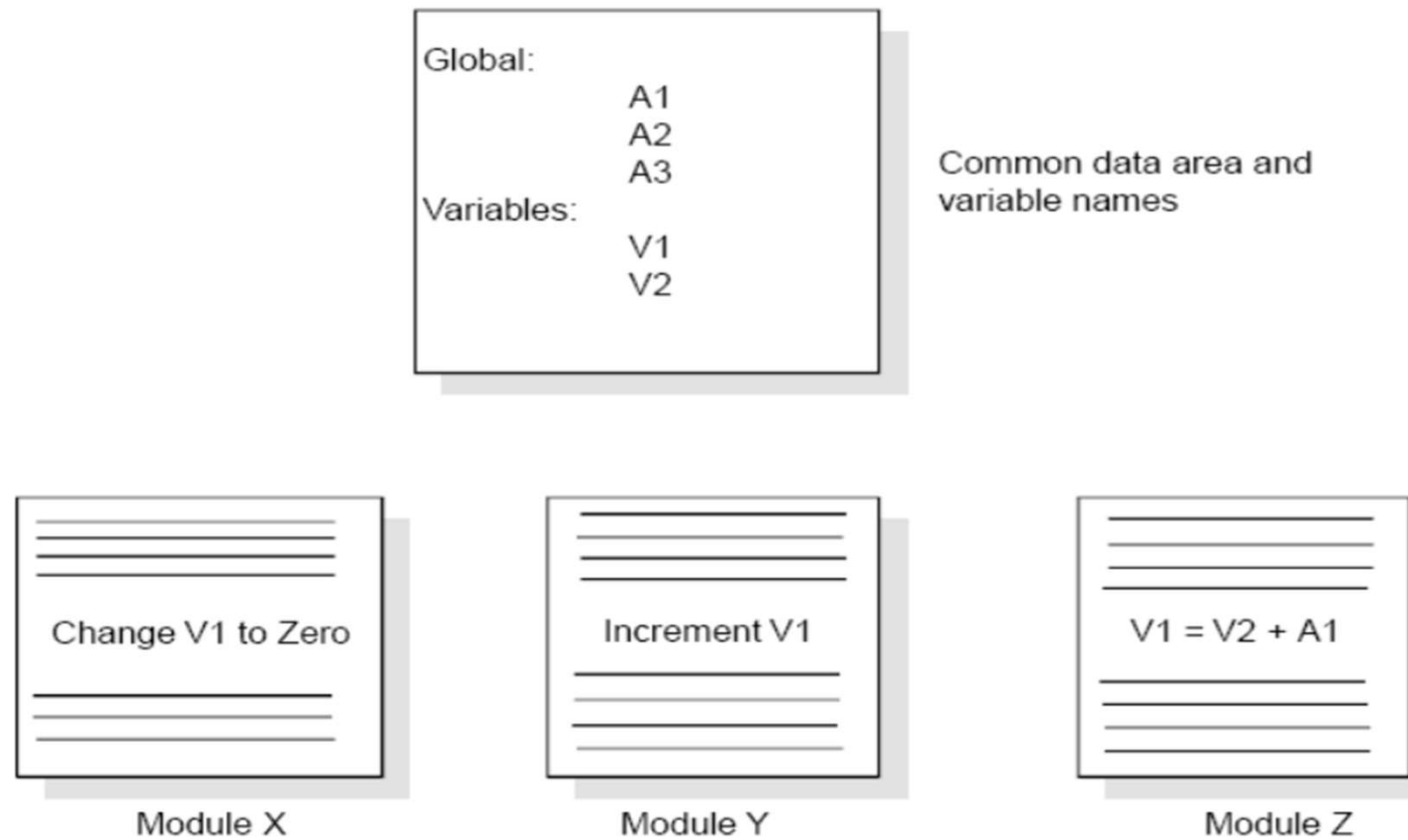
- **Control coupling**

- Module A and B are said to be control coupled if they communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

- **Common coupling**

- With common coupling, module A and module B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of changes.

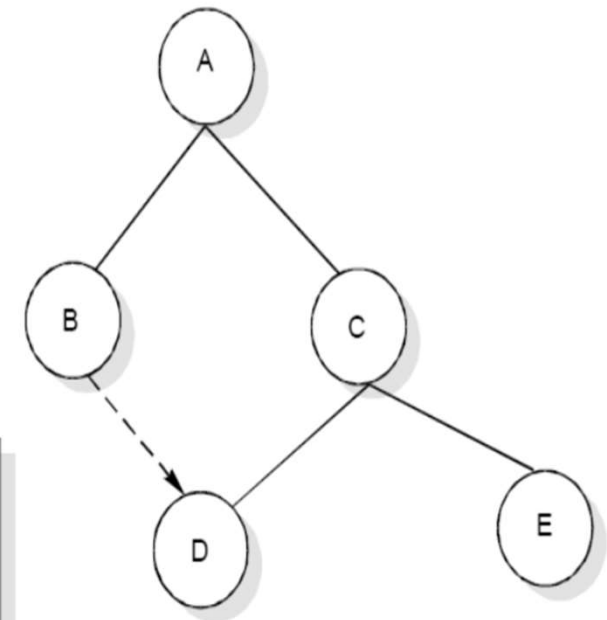
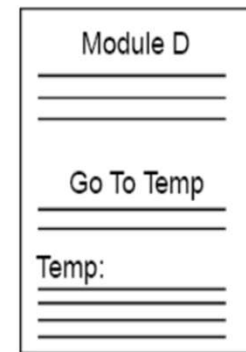
# Example of Common Coupling



# Types of Coupling (cont)

- **Content coupling**

- Content coupling occurs when module A changes data of module B or when control is passed from one module to the middle of another.
- Module B branches into D, even though D is supposed to be under the control of C.





# Law of Demeter

- An object should **send messages to only** the following kinds of objects:
  - The object itself
  - The object's attributes (instance variables)
  - The parameters of the methods in the object
  - Any object created by a method in the object
  - Any object returned from a call to one of the methods of the object
  - Any object in any collection that is one of the above categories
- Discussion: Law of Demeter and cohesion & coupling



# Questions



**THANK YOU**