Software Engineering CS3003

Lecture 6: Software Complexity

The reality of software use

- The Standish Group in 2002 reported that for software, generally:
  - 45% of features were never used
  - 19% used rarely
  - 16% sometimes
  - only 20% were used frequently or always

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.
—C. A. R. Hoare

source: wikipedia

Structure of this lecture

This lecture will explore:

- "Wicked" problems
- What is cohesion and levels of good/bad cohesion
- What is coupling and levels of good/bad coupling
- Talk about the coursework
- Links to previous lectures

Lecture schedule

Week	Lecture Topic	Lecturer	Week Commencing
1	Introducing the module and Software Engineering	Steve Counsell	20th Sept.
2	Software maintenance and Evolution	Steve Counsell	27th Sept.
3	Software metrics	Steve Counsell	4th Oct.
4	Software structure, refactoring and code smells	Steve Counsell	11th Oct.
5	Test-driven development	Giuseppe Destefanis	18th Oct.
6	Software complexity **Coursework released Tues 26th Oct.**	Steve Counsell	25th Oct.
7	**ASK week**	**N/A**	**1st Nov**
8	Software fault-proneness	Steve Counsell	8th Nov.
9	Clean code	Steve Counsell	15th Nov.
10	Human factors in software engineering	Giuseppe Destefanis	22th Nov.
11	SE techniques applied in action	Steve Counsell	29th Dec.
12	Guest Lecture (tba) **Coursework hand-in 6th December**	Guest Lecture	6th Dec.

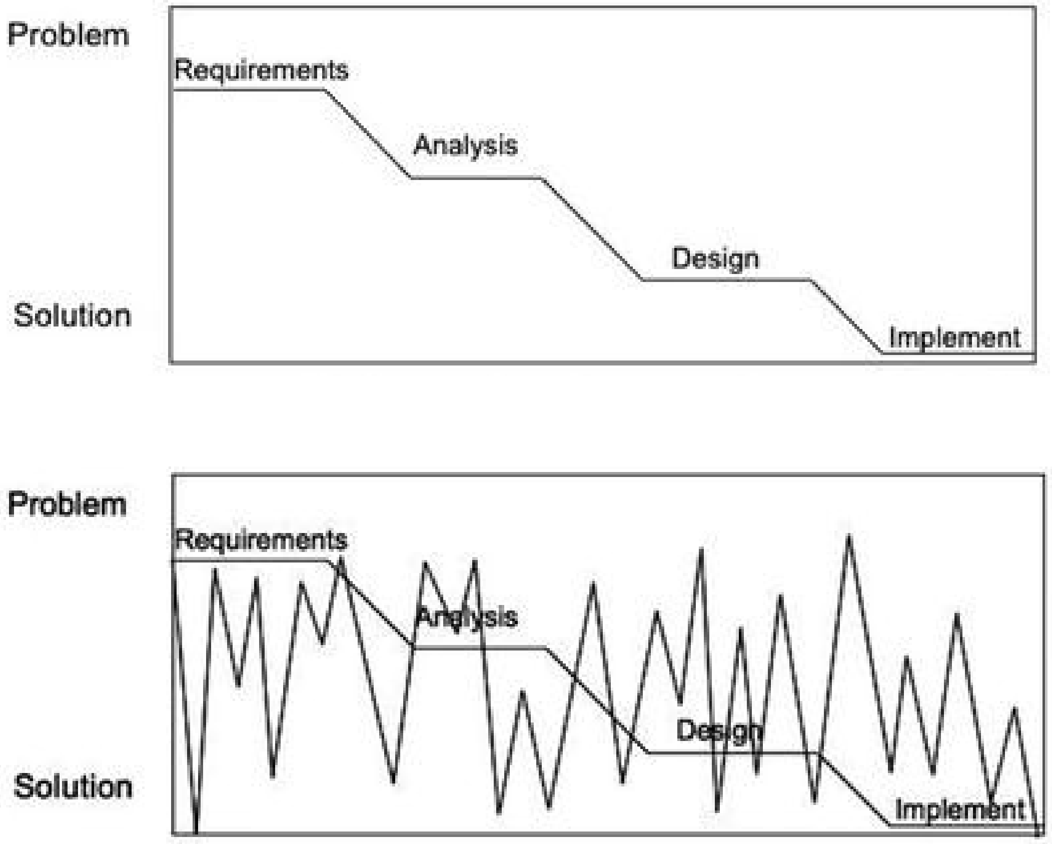
Lab schedule

Week	Labs	Week Commencing
1	**No labs**	20th Sept.
2	Lab (Introduction)	27th Sept.
3	Lab	4th Oct.
4	Lab	11th Oct.
5	Lab	18th Oct.
6	**No lab**	25th Oct.
7	**ASK week**	**1st Nov.**
8	Lab	8th Nov.
9	Catch-up Lab	15th Nov.
10	Work on coursework (no Lab)	22nd Nov.
11	Work on coursework (no Lab)	29th Nov.
12	**No lab**	6th Dec.

“Wicked” problems

- A **wicked problem** is a problem that is difficult or impossible to solve because of
  - Incomplete, contradictory and changing requirements that are often difficult to recognize
- Characterised by (according to Conklin):
  - A wicked problem is not understood until after the formulation of a solution.
  - Solutions to wicked problems are not right or wrong
    - Just better or worse
  - Every wicked problem is essentially novel and unique
  - Every solution to a wicked problem is a "one shot operation.
    - There is no opportunity to learn by “trial and error”

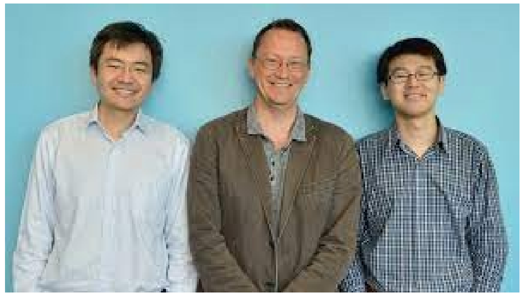
Software implementation wicked problem



The jagged line indicates the designer's work moving from the problem to a solution prototype and back again

Characteristics of Good Design

- Class independence
  - High cohesion
  - Low coupling
- Fault prevention and fault tolerance
  - Automatic program repair (APR) at Facebook
    - Professor Mark Harman and the team (UCL also)
- Design for change
- A shallow and long bathtub curve

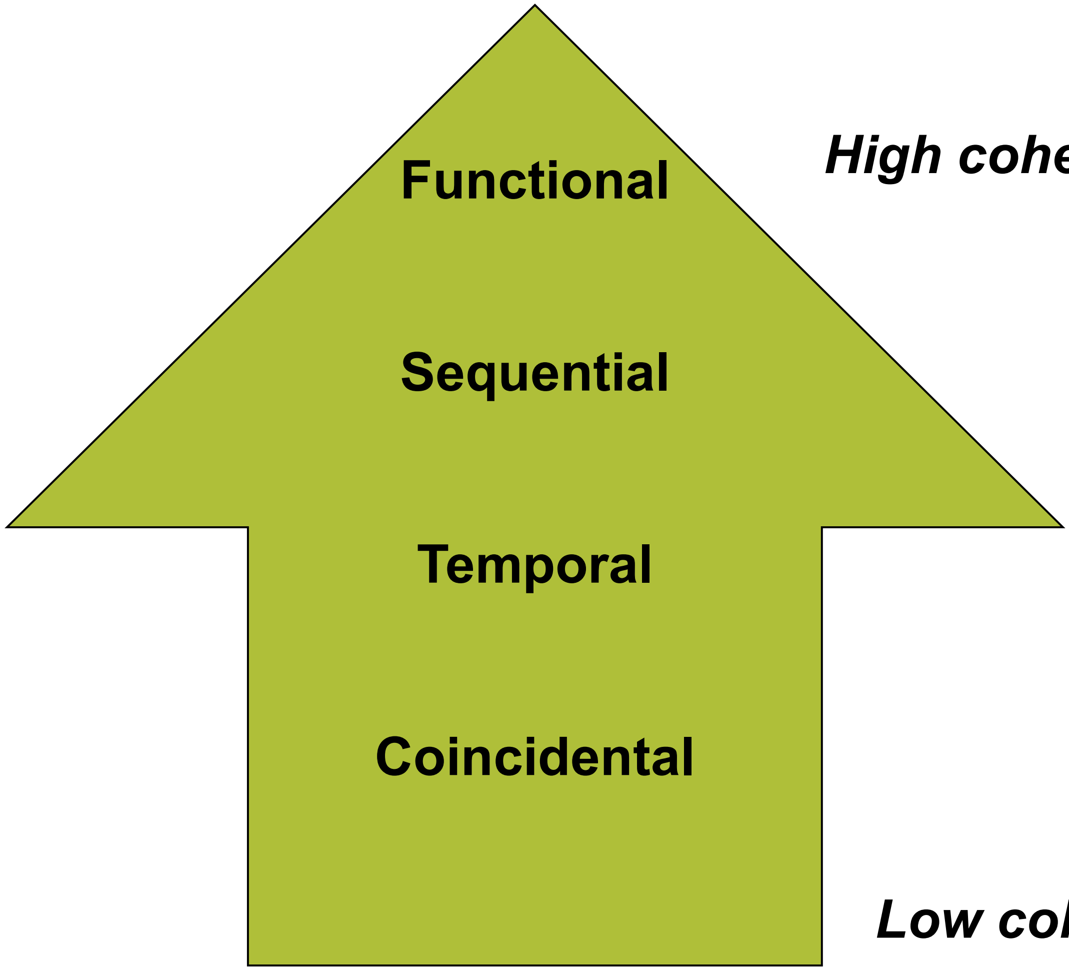


Source: Facebook

Cohesion

- Definition
  - The degree to which all methods of a class are directed towards a single task
  - The degree to which all responsibilities of a single class are related
- Methods with which a class is constructed are what matters:
  - All methods of the class are directed toward and essential for performing the same task and that is what developers should strive for at all times

Cohesion types



High cohesion

Low cohesion

Functional Cohesion

- Def: Every essential method required for the computation is contained in the class (and only that)
- Every method in the class is essential to the computation
  - Ideal situation
- What is a functionally cohesive class?
  - One that not only performs the task for which it was designed but
    - it performs only that function and nothing else
- Example: a class that assigns a seat to airline passenger and does nothing else

Sequential Cohesion

- Def: The output of one part (e.g., method) is the input to another part (e.g. method).
- *Data flows* between parts
- Example: a class which reads data from a file, formats the data and then processes the data
- Another example….

Sequential cohesion (cont.)

- Extract some raw data about a student
  - Registration details, for example
- Format that raw data
- Validate fields in formatted data
- Return formatted cross-validated data
- Each method has to be executed in turn
  - That is why it is called ‘sequential’ cohesion

Temporal Cohesion

- Def: Methods are related by timing involved
- Example: An error logger which
  - Closes all open files then creates an error log, then notifies user
- So, lots of different activities occur one after the other
- This type of cohesion differs from sequential cohesion because data is not necessarily being passed around in Temporal cohesion

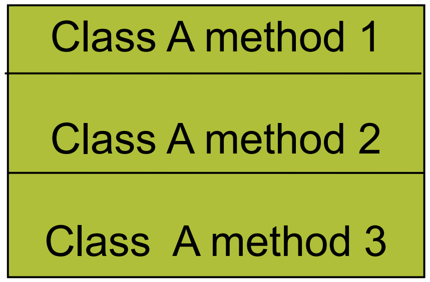
Coincidental Cohesion

- Def: Methods of the class are unrelated (unrelated functions, processes, or data)
- Methods of the class are only related by their location in source code
- Accidental
- Worst form
- Large class code smell often arise when classes become bloated with methods that are unrelated
  - Refactor if that is happening

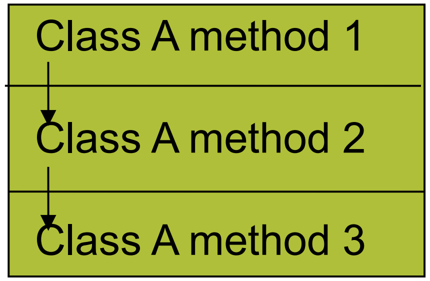
Example of coincidental cohesion

1. Print next line
2. Reverse string of characters in second argument
3. Send someone an automated Birthday Message
4. Add 7 to 5th argument
5. Request cinema ticket details

Examples of Cohesion

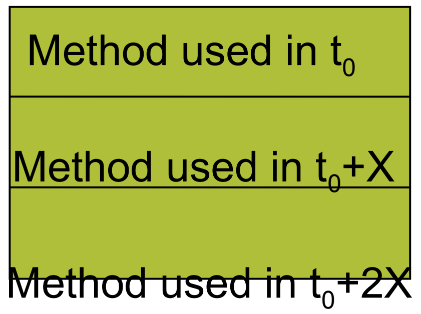


Functional
Sequential with complete, related methods

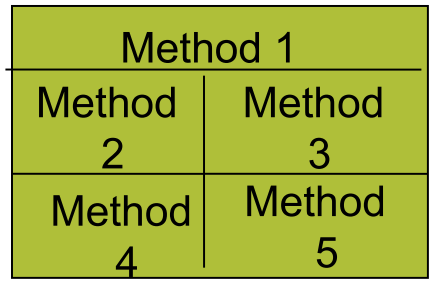


Sequential
Output of one is input to another (i.e., data is being passed from one to the next)

Examples of Cohesion



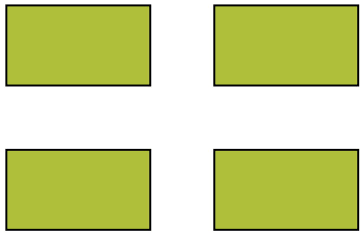
Temporal
Methods related by time

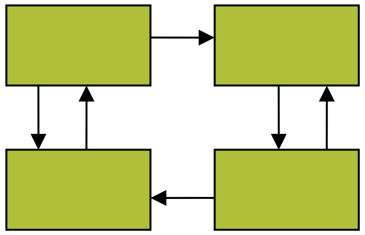


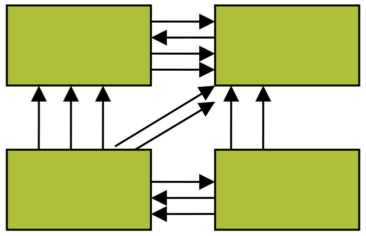
Coincidental
Methods of class are unrelated

Coupling

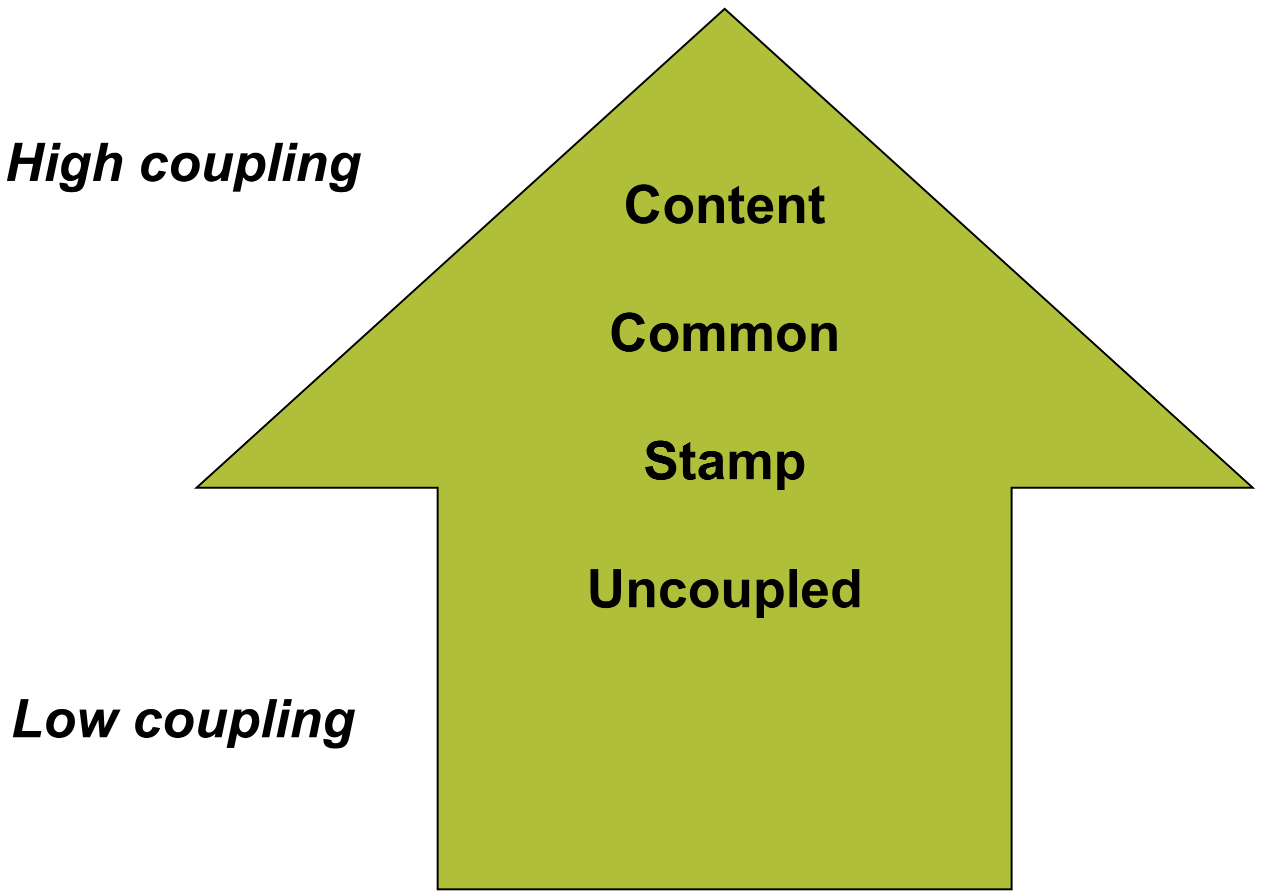
- The degree of dependence between classes
- The amount of interactions among classes


No dependencies


Lowly coupled
some dependencies


Highly coupled
many dependencies

Coupling types



Content Coupling

- Def: One class modifies the parts of another
- Example:
  - Class directly modifies another’s data
  - Violates encapsulation

22

Example

Part of a program handles lookup for customer

When customer is not found, class adds customer by directly modifying the contents of the data structure containing customer data being looked after by another class.

Common Coupling

- Def: More than one class shares data such as global data structures
  - But not in the form of a database where integrity of the data can be protected
- Usually a poor design choice because
  - Lack of clear responsibility for the data
  - Difficult to determine all the classes that affect a data element (reduces maintainability)
  - Difficult to reuse classes

Example

A process control class maintains current data about state of operation (e.g., nuclear reactor software):

- Gets data from multiple places
- Supplies data to multiple places
- Many classes write directly to global data store
- Many classes read directly from a global data store

Stamp Coupling

- Def: A class passes a data structure to another class that does not have access to the entire structure
- Requires second class to know how to manipulate the data structure (e.g., needs to know about implementation)

Example

Customer Billing System

- The print class of a customer billing system accepts **part of the huge customer data structure** as an argument
- It then parses it, and…
- ….then prints the name, address, and billing information.

Uncoupled

- Completely uncoupled classes are not systems
- Systems are made of interacting classes

Consequences of high and low coupling

- High coupling
  - Classes are difficult to understand in isolation
  - Changes in class ripple to others
  - Classes are difficult to reuse
    - Need to include all coupled classes
    - Difficult to understand
- Low coupling
  - Poor cohesion
  - Poor code understandability
    - Difficult maintenance

Questions

- **Question 1:** The **cohesion** of a class is dependent on the developer writing classes well. How would you ensure that the classes you wrote as a developer were as cohesive as possible?
- **Question 2:** Describe two types of cohesion and two types of coupling providing an example for each. What are the problems of having classes with low cohesion and excessively high coupling?
- **Question 3:** Why do you think it is important to measure the complexity of code?
- **Question 4:** “Coupling in OO systems is a good thing and the more there is of it the better”. Discuss this statement.

Reading for the week

- https://ducmanhphan.github.io/2019-03-23-Coupling-and-Cohension-in-OOP/
- A comparison and evaluation of variants in the coupling between objects metric. J. Syst. Softw. 151: 120-132 (2019)
    - My paper!!! With colleagues from South Bank
- Standish Group 2002:
    - http://www.featuredrivendevelopment.com/node/614
- https://www.open.edu/openlearn/science-maths-technology/approaches-software-development/content-section-1.5.4

Reading for the week (cont.)

- https://alldifferences.net/difference-between-cohesion-and-coupling/
- https://wiki.c2.com/?CouplingAndCohesion