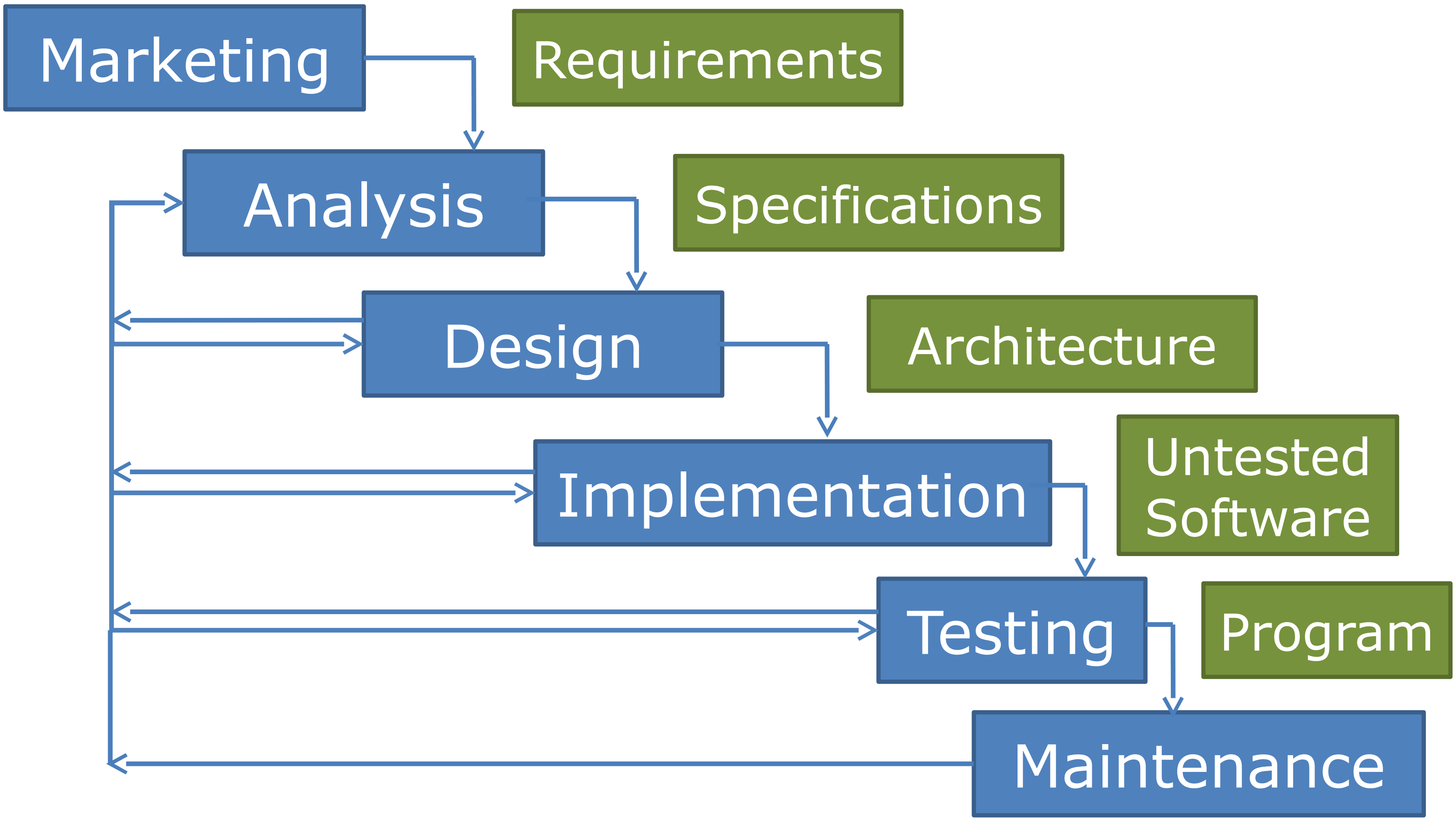
Software Engineering

Requirements + Specifications

Software Engineering

- The term *software engineering* refers to the study of software development on large scales.
- Few programs these days are written by lone, individual programmers.
  - Instead, programs are often written by large teams who must coordinate their efforts.

The Waterfall Model

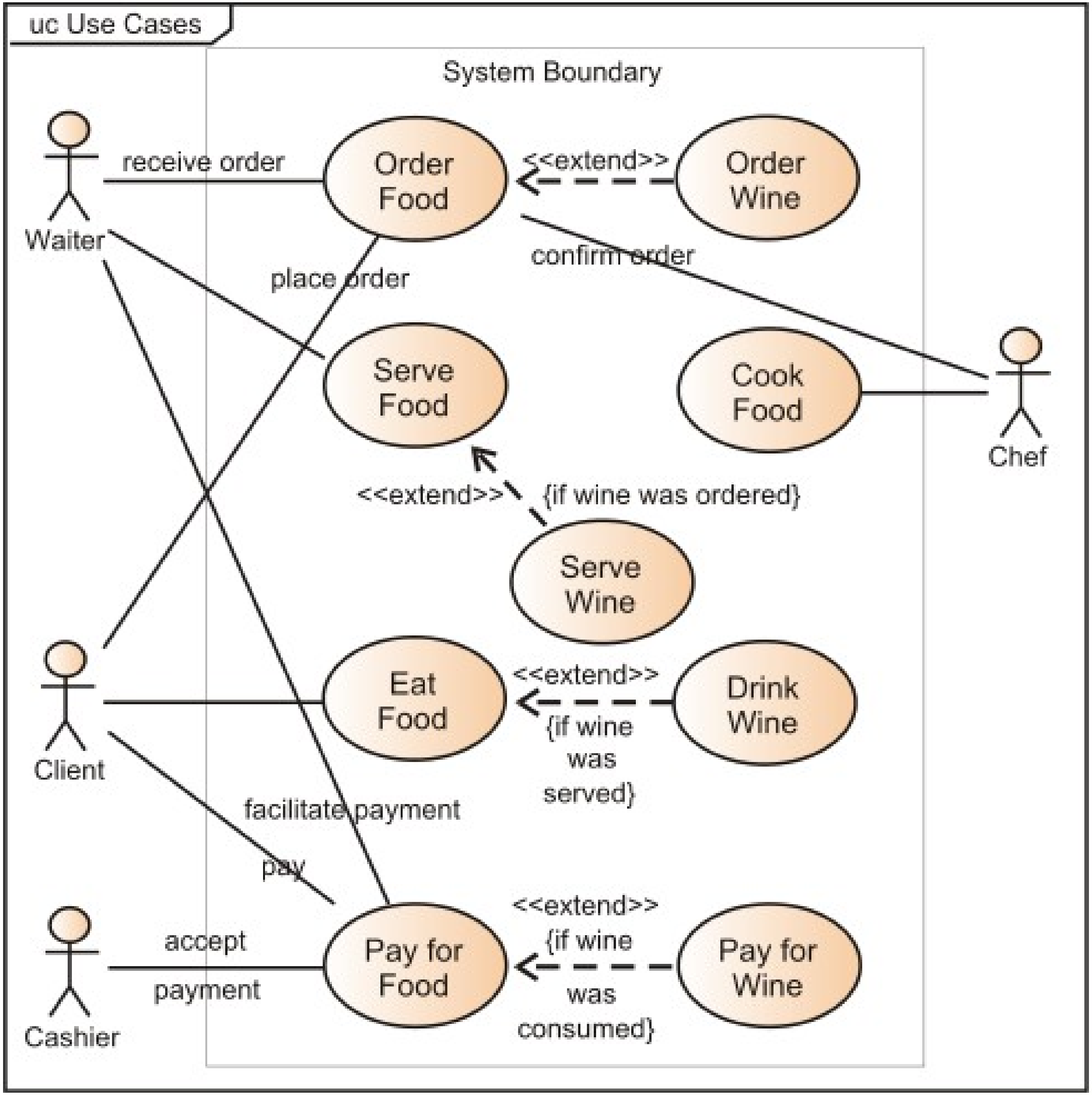


Use Cases

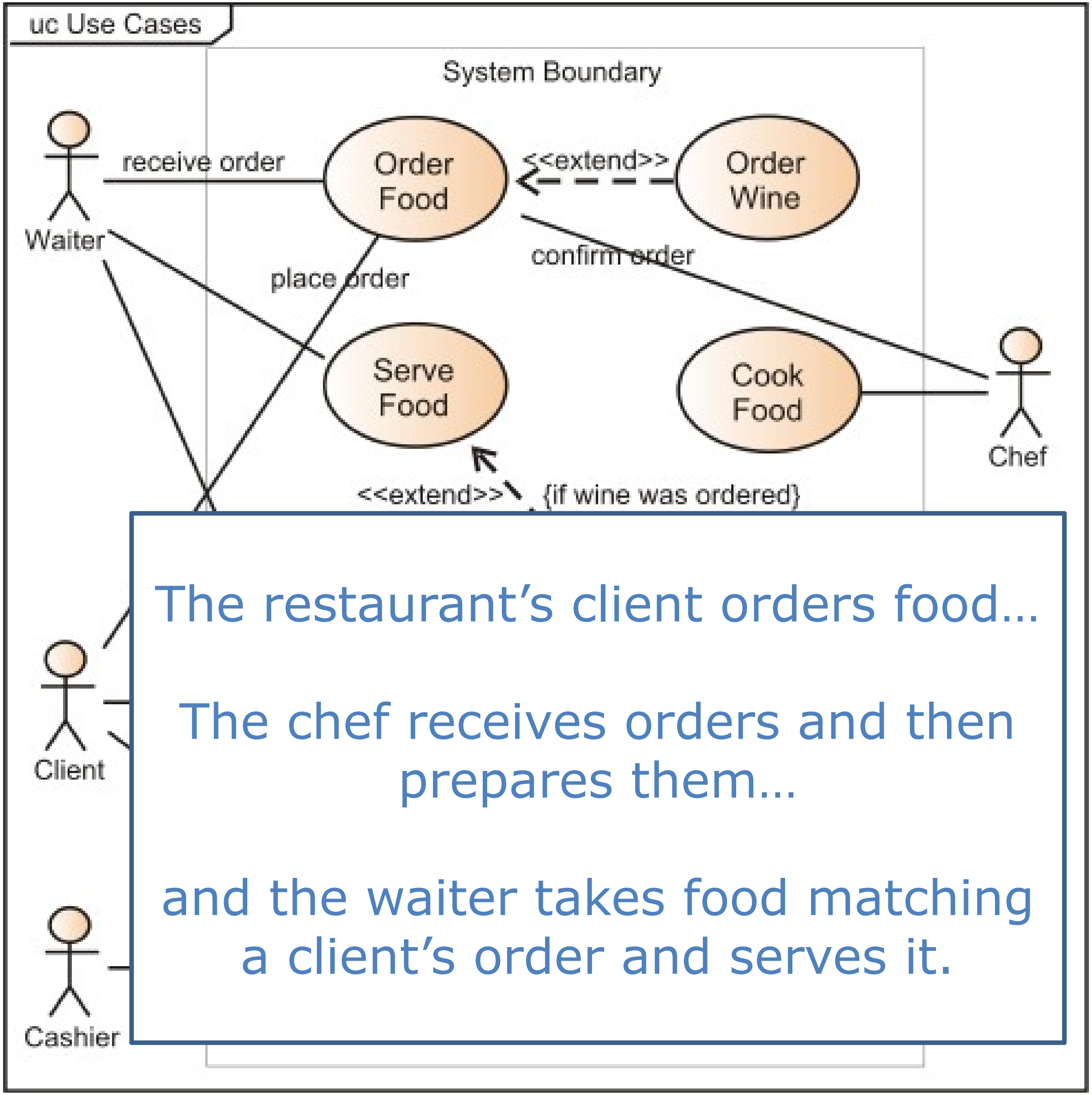
- The idea of *use cases* is to determine how potential users with differing roles would need to use a program to accomplish their tasks.
  - What sort of access would different types of users need within the system to perform their tasks?
  - These may often involve multiple scenarios.

Use Cases

- Focuses:
  - “Who” can do “what”?
  - Each potential “who” is referred to as an *actor*.
  - “What” needs to be done, rather than “how” it should be done.



Source: Wikipedia



The restaurant's client orders food…

The chef receives orders and then prepares them…

and the waiter takes food matching a client's order and serves it.

Use Cases

- Why use them?
  - They’re very useful for communicating with clients.
  - Relatively simple notation, visually clear.
  - Helps to flesh out requirements – the visual representation makes it easier to sense if something’s missing.
  - They also help to design test cases for later use.

Requirements

- Why is it important to get good requirements initially?
  - 2003 study: 70% of all projects failed, or were incomplete due to going overbudget or overschedule.
  - A related study quoted by this one: 83.8%!
  - The common, primary cause: changing or unclear (poor) requirements.

Requirements

- Mercer Consulting:
  - When the true costs are added up, as many as 80% of technology projects actually cost more than they return. It is not done intentionally but the costs are always underestimated and the benefits are always overestimated. Dosani, 2001

Requirements

- Requirements could be
  - Overlooked: certain expectations won’t be met, and thus critical features will be missing.
  - Incorrect: features won’t be implemented in an effective way for the end user.
  - Poorly communicated.

Analysis

- Previously in the semester, we examined how analysis is performed on an individual object level.
  - Similar analysis may be performed upon available requirements in order to fully flesh them out into a structured, concrete specification.
  - We'll also add in a few additional items for consideration at this time.

Analysis

1. Inputs: what data will be input into the program by one or more "actors"?
   - What commands will be available?
   - What will be the format for each type of data?

Analysis

2. Outputs: what data will our program generate that will be needed by one or more "actors"?
   - How will data be output?
   - Are there different ways it may potentially need to be output?
   - Might the user wish to output only a subset of the data?

Analysis

3. Constraints: we've previously noted that sometimes objects should have limitations imposed on them "artifically," in order to model what they represent in the real world more accurately.
   - Might there be analogous system-level constraints which should be enforced?

Analysis

4. <u>Assumptions</u>: Are we operating under any sort of assumptions?
   - Either on our own part or on those of the client?)

Analysis

5. <u>Modifications</u>: Whenever an object within the program is modified, will any corresponding changes be expected elsewhere within the system?
   - Adding a new element into a data structure (typically) may not automatically add it into corresponding structures of the user interface.

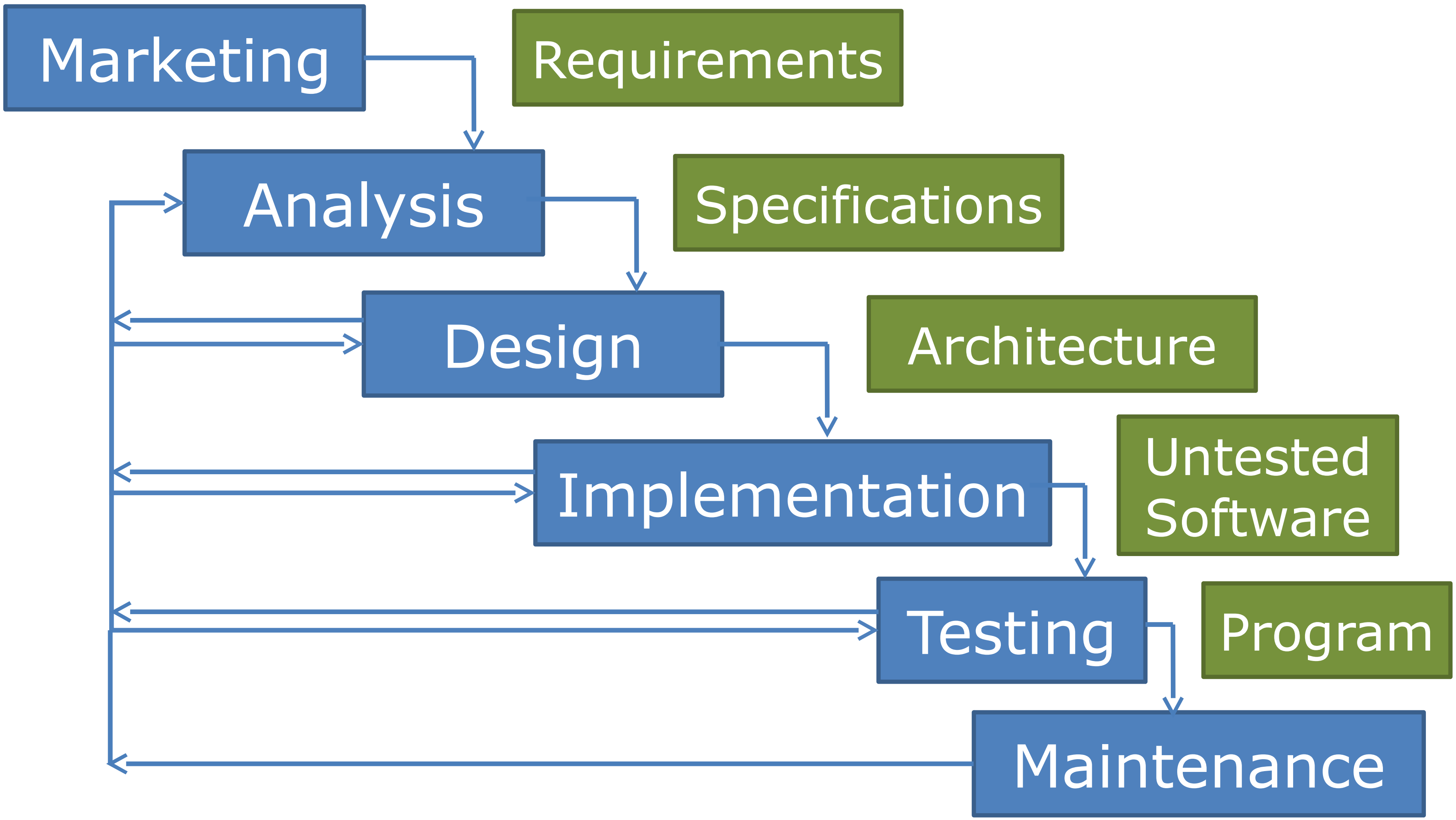
Analysis

6. Relationships/Effects: How are system-level modifications related to system constraints?

Software Engineering

Design & UML

The Waterfall Model



Design

- Once the specification is completed, the next step is to determine the design for the desired system.
  - That is, once we know “what” sort of system is both possible and acceptable to both parties, we may then turn to the question of “how” to make that system.

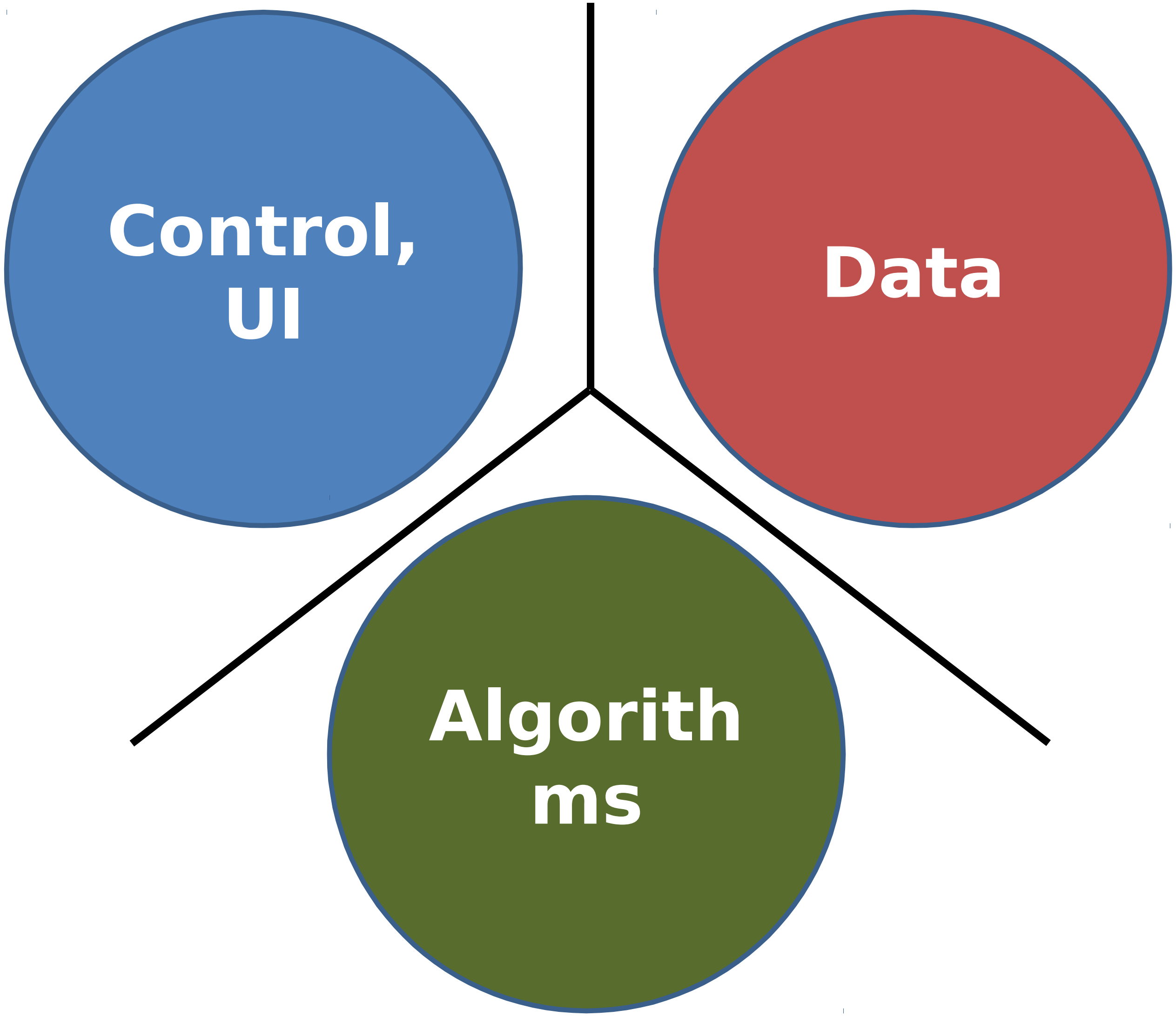
Design

- The goal of the design process is to develop the potential structure for a codified system which would fulfill the determined specification for the desired program.
  - Basically, we want to figure out how we would ideally code up the program before actually writing a line of code.

Design

- This process will often involve splitting the underlying problem into multiple pieces that are simpler to solve.
  - This is then done, repeatedly, until these smaller problems are reduced to the object level.
- One early potential split of the problem…

Ideal Program Division



Ideal Program Division

- The manner by which the different data elements will be represented internally does not have to be tied to its representation for input or output.
  - At the same time, we should design objects to make the task of input and output easier.

Ideal Program Division

- As noted when discussing “use cases”, sometimes not all users of system should have access to the same user interface. (UI)
  - Each type of “actor” should only be able to use program features it needs.
  - As such, the true, core functionality of a program should not be linked directly to any single UI within the system.

Class Hierarchy

- As with the process of determining requirements and the specification of a program, it is often helpful to have visual diagrams to aid in the design process as well.
  - For design, we now wish to capture the relationships among individual classes.

Class Hierarchy

- How can we represent these design ideas for a given programming project effectively and efficiently?
  - One super-common visualization tool for the design process is known as UML: the *Unified Modeling Language*.
  - Not to be confused with HTML, XML, …

Class Hierarchy

- In UML, each class and interface gets specified, along with arrows to show the relationships among them.
- Furthermore, the methods (and fields, for classes) of each are also specified.
  - This establishes a standardized, known interface that other coders on the team may then use for each object type.

UML

- Standard structure of a UML diagram element for a class:

```
MyClass
------------------------------------------
- int integer_field
- String string_field
------------------------------------------
+ int get_Integer_Field()
+ void set_Integer_Field(int)
+ void doSomething(String, int)
```

```
Class Name
--------------------
----
fields

--------------------
----
methods
```

UML

- A '-' indicates the *private* modifier, and '+' the *public* modifier.

```
MyClass
-----------------------------------------
- int integer_field
- String string_field
-----------------------------------------
+ int get_Integer_Field()
+ void set_Integer_Field(int)
+ void doSomething(String, int)
```

```
Class Name
--------------------
----
fields

--------------------
----
methods
```

UML

- UML thus allows us to visually model the conceptual (and eventually-to-be codified) relationships among the elements of a program.
  - It visually represents the polymorphic nature of the different types which will be implemented.
  - It also models the general dependencies across types.