## Phases of software lifecycle

Requirements Analysis:

* Objectives: What should this system do
* Inputs: From stakeholders
* Outputs: Requirements Definition Document
* Structured analysis, use case analysis

Design & Specification

* Objectives. How should this system do
* Inputs: Requirements Definition Document
* Outputs: Design documents
* Structured design, object-oriented design, component-based design

Implementation

* Objectives: Make the system
* Inputs: Design documents, RDD
* Outputs: Code
* Verification:
  + Unit testing
  + Integration testing
  + Inspection
  + Formal Mathematical methods

Quality Assurance

* Verification & Validation
  + Validation:
    - Producing valid product
  + Verification:
    - Producing product right
* Each stage
* System validation:
  + Does implementation meet requirements analysis

Maintenance

* Lifetime activity
* Bug-fixing
* Feature improvements
* Preventive maintenance

Process Models:

* Linear:
  + Waterfall
  + Allows predictive planning
  + Allows for finding errors early when less costly
  + Not flexible
  + Difficult to fully specify requirements
* Iterative:
  + Allows adaptive planning
  + Feedback
  + Difficult to stick to plans

## Requirements Engineering

Elicitation

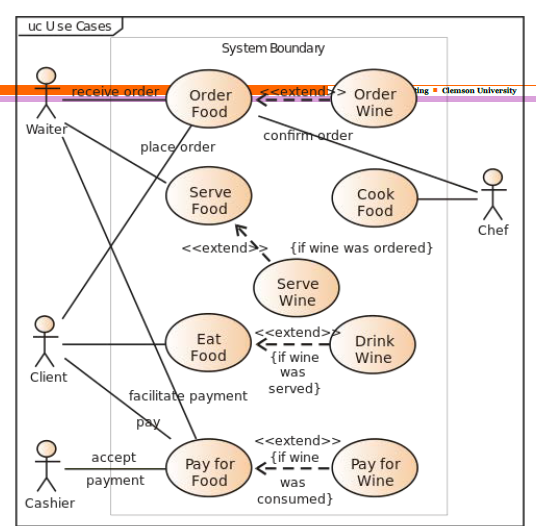
* Identifying stakeholders
* Making business case
* Understanding priorities
* Knowing risks
  + Expectation of loss
  + Causes:
    - Lack of information, control, or time
  + Examples:
    - Increase in production cost, development of poor quality software, unable to complete project on time
* Interviews
  + Questions concerning system
  + Context and scope of system
  + Meta-questions

Modelling

* Understand what aspects of the system are relevant to the model
* Sufficiently complex, but not more than necessary to capture essential elements

Functional Requirements

* Completeness
  + All requirements documented
* Consistency
  + No conflicts between requirements
* Precision
  + No ambiguity



# Use Case Analysis

UML

* Modelling language for object-oriented programming

Use case

* Sequence of actions, variant sequences and error sequences
* Set of scenarios tied together by common goal

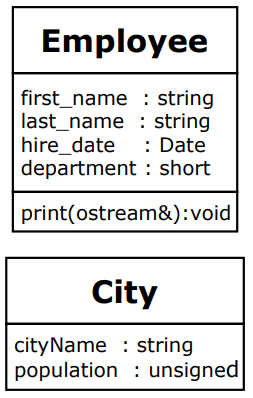
# Design and Class Diagrams

Coupling

* How inter-related modules are
* Goal: Minimize couples, to require less communication
* When unavoidable:
  + Avoid:
    - Global coupling
    - Content coupling (implementation inheritance)
    - Control coupling (external flags)
  + Parametric Coupling
    - Use parameters
    - Interfaces

Cohesion

* How single-minded each modules are
* Goal: Maximize
* Undesirable forms:
  + Coincidental (accidental)
  + Logical
  + Temporal (Time)
* Desirable forms:
  + Functional
    - Perform related functionality

Class Diagram Notation

* Boxes denote classes
  + Class name
  + Attributes
  + Operations
* Form: visibility name: type multiplicity = default {property-string}
  + Visibility: + or – or # or ~
    - Indicates public, private, protected, or package
  + Name: Name of field or assosciation
  + Type: Kind of object
  + Multiplicity:
    - 0..1 optional
    - 1..\* at least one
    - 0..\* any number
  + Default: Value of default value
  + Property-string: Additional properties using keywords (eg ReadOnly)

Associations:

* Notes a property
* Relationship to other pclasses or objects
* Solid line from source class to target class