ECE 321 Software Requirements Engineering

Lecture 8: An introduction to formal SRS methods

The 3 steps of the requirements development process

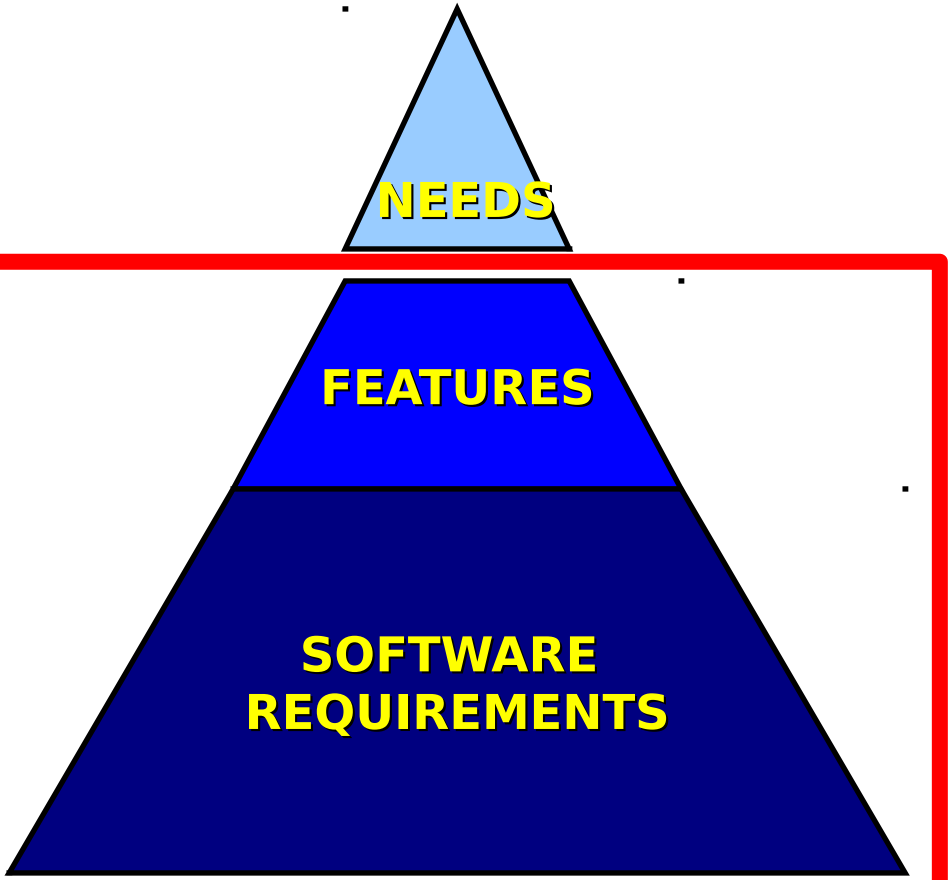
- **Requirement elicitation**
  - Understanding and analyzing the problem
  - Learning and understanding user needs
- **Requirement specification**
  - Developing a vision document
  - Developing requirement specification document
- **Requirement validation and verification**

The 3 steps of the requirements development process

- **Requirement elicitation**
  - Understanding and analyzing the problem
  - Learning and understanding user needs
- **Requirement specification**
  - Developing a vision document
  - Developing requirement specification document
- **Requirement validation and verification**

The software requirements pyramid

- problem domain
- solution domain



We have seen informal specifications until now

- Written in natural language
- What is the problem of informal specifications?
  - They may use terms that are not well-defined, or understood differently by stakeholders
  - We need formal methods as well

An example of an informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Have you ever used a formal method?

- Can you solve this riddle:
  - My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48.

- How old are they?

The solution to the riddle

Model

$$\boxed{x+(x+3)+(x+6) = 48}$$

$$\begin{aligned} 3x+9 &= 48 \\ 3x &= 39 \\ x &= 13 \end{aligned}$$

The ages of the children are 13, 16 and 19

The solution to the riddle

Formal method

$$\boxed{\begin{aligned} x+(x+3)+(x+6) &= 48 \\ 3x+9 &= 48 \\ 3x &= 39 \\ x &= 13 \end{aligned}}$$

The ages of the children are 13, 16 and 19

All of you have used formal methods before!

- Formalism
  - Notation for writing specifications using the language of mathematics
- Model
  - Provides abstract representation of a system

Formal methods then

- In the 1970s, formal methods were used to produce automatic programming methods
  - They did not really work
  - Only successful in specific domains, like databases
    - Generate a database management system from a specification

Formal methods now

- Formal software engineering is broad
- Formal methods are considered part of the software engineering process
  - Specifications
  - Architecture
  - Analysis/testing
  - Reliability and performance engineering
  - Configuration management
  - Process management

What are formal methods?

- **Writing** a formal specification
- **Proving** properties about the specification
- **Constructing** a program by mathematically manipulating the specification
- **Verifying** the program by mathematical argument

How to write a formal specification?

- Three specification styles
  - Declarative specifications
    - Also called descriptive
  - Operational specifications
    - Also called imperative
  - Structural specifications
    - Also called relational

- We need all 3 styles to describe a system

Declarative / descriptive specifications

- Describe desired properties
- Given using:
  - Axioms – statements of properties
  - Algebras – sets of math operations and values
- Do not specify how to achieve properties
- Example: data structure design

Example: data structure design

delivery instruction = patron name
+ patron phone number
+ meal date
+ delivery location
+ delivery time window

Delivery location = “building and room to which an ordered meal is to be delivered”

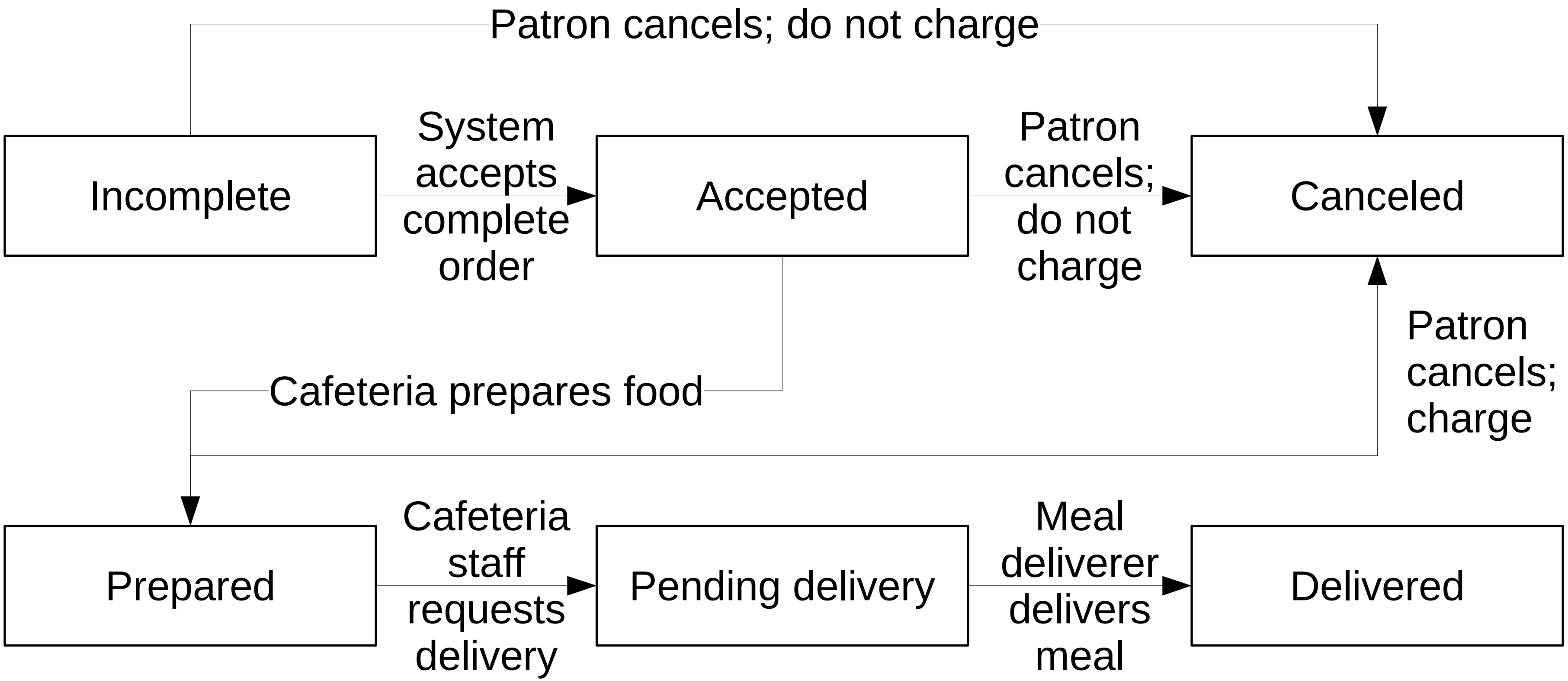
Delivery time window = “15 minute range during which an ordered meal is to be delivered; must begin and end on quarter-hour intervals”

... etc.

Operational / imperative specifications

- Describe desired actions
  - Sequences of states
  - Behaviour of system
  - How we achieve properties
- Example: state transition diagram

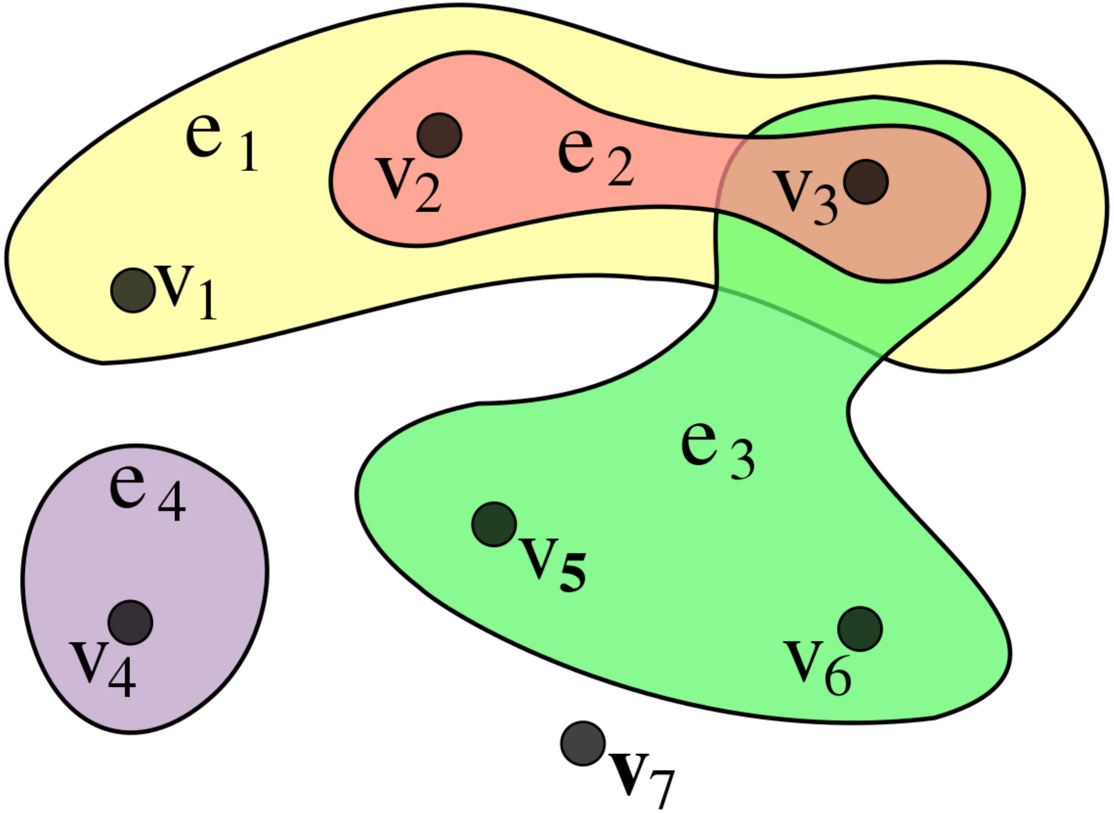
Example: state transition diagram


Patron cancels; do not charge
Incomplete
System accepts complete order
Accepted
Patron cancels; do not charge
Canceled
Patron cancels; charge
Cafeteria prepares food
Prepared
Cafeteria staff requests delivery
Pending delivery
Meal deliverer delivers meal
Delivered

Structural / relational specifications

- A special type of descriptive specification
- Describe desired relationships
- Can be described using a multi or hypergraph
  - Multigraph: graph in which two vertices can have more than one edge
  - Hypergraph: graph in which an edge can join more than two vertices

Example: a hypergraph



Comparing the specification styles

- Members of my family are tall and most of them are smart
  - Declarative specification
- My family consists of my father, mother, and brother. Also it includes two uncles
  - Structural specification
- During Christmas, we drive to my grandmother first by car, then walk to my aunt. After that we relax at home.
  - Operational specification

Static vs. dynamic analysis of specifications

- Static
  - Examines specification to reveal properties
    - Looks at all possible states of a system
- Dynamic
  - Executes specification to reveal properties
    - Looks at a particular state of a system

- Example: deadlock analysis
  - Static analysis to show that deadlock does not exist
  - Dynamic analysis to show that deadlock exists

Some building stones of specifications 1/2

- Mathematical
  - Set theory
  - Graph theory
  - Automata theory
  - Abstract algebra
  - Probability and statistics

Some building stones of specifications 2/2

- Logical
  - Predicate/propositional logic
    - NOT, AND, OR, for all, exists
  - Temporal logic systems
    - Captures time (while, until)

Next few lectures

- Brief example of a formal method/language that you are likely to encounter as a software engineer
- Semi-formal specifications
  - UML
- Descriptive specifications
  - Algebraic specifications
  - Logic-based specifications
- Operational specifications
  - Finite State Machines
  - Petri-Nets

Let’s go back to our example of an informal specification...

An example of an informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

How can we formalize the specification?

- Step 1: Analyze informal specification
- Step 2: Develop descriptive specification
- Step 3: Develop operational specification
- Step 4: Develop structural specification

Step 1: Analyze informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Step 1: Analyze informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Step 1: Analyze informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Step 1: Analyze informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Step 1: Analyze informal specification

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived

Step 2: Develop descriptive specification 1/3

- A system consists of a set of object files
- Each object file is derived from one or more source files
- Set of object files

  $O = \{o_1, o_2, o_3, ...\}$

- Set of source files

  S = {s1, s2, s3, ...}

- Set of all files

  F = O (set sum) S

Step 2: Develop descriptive specification 2/3

- A system consists of a set of object files
- Each object file is derived from one or more source files
- **Derive function (relation)**

  $D: O \rightarrow 2^S$

  - Maps object files into one or more source files
  - 2^S is the power set of S (set of all subsets)

Step 2: Develop descriptive specification 3/3

- Object and source files have a timestamp indicating when they were last modified
- If an object file is older than any source file, the object file must be rederived
- **Timestamp function (relation)**

  T: F -> R

- Assertion

  Object timestamp must be greater than source timestamp

  F in O such that

  for all o_i in O, for all s_j in D(o_i)

  $T(o_i) > T(s_j)$

Can our current formal specification answer these questions?

- Do we allow an infinite number of files in our system?
  - Specification does not tell us
- What does it mean to be older in our system?
  - Can a source file have the same age as the object file from which it was derived?
  - System time granularity can affect the “older” relation

A descriptive specification alone is not enough!

- Usually we need a combination of descriptive, operational and structural styles

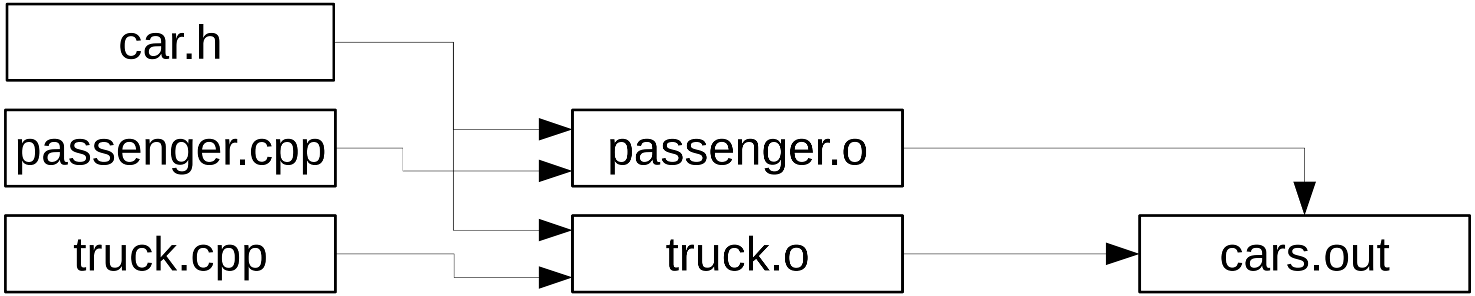
- For our example, we can use the **make** specification language
  - It specifies dependencies between artifacts
  - It specifies rules for creating new artifacts
  - It specifies actions to carry out the rules

Specification styles in make

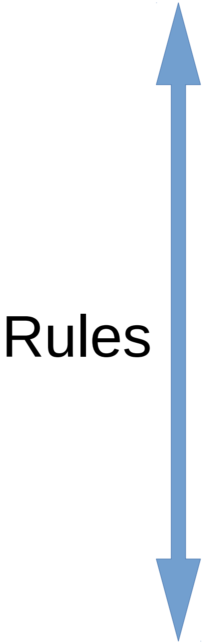
- **Declarative/descriptive**
  - Rules
  - Primitive objects are files
- **Operational/imperative**
  - Actions, which are shell commands
- **Structural/relational**
  - Dependencies denoted by rules
- Rules are placed in a makefile to denote specification

Step 3: Develop operational specification 1/3

- Specification in a makefile
  - We start with *.h and *.cpp files
  - Then *.o files are derived
  - Finally *.out file is derived



Step 3: Develop operational specification 2/3



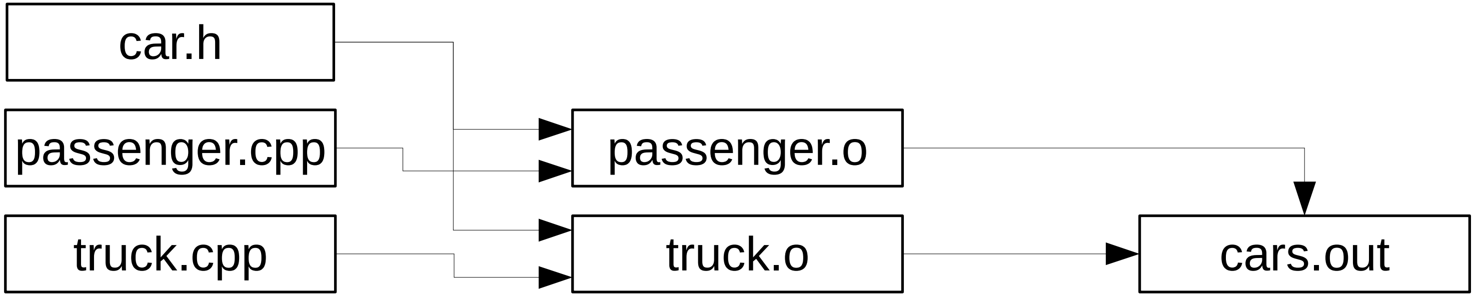
```
cars.out: passenger.o truck.o
    cc passenger.o truck.o -o cars
truck.o: truck.cpp car.h
    cc -c truck.cpp
passenger.o: passenger.cpp car.h
    cc -c passenger.cpp
```

Step 3: Develop operational specification 3/3

- If an object file is older than any source file, the object file must be rederived
- Passenger.o should be younger than passenger.cpp and car.h
  - If not, it is recompiled, meaning a new younger passenger.o is generated
- Rules are executed bottom to top, left to right

Step 4: Develop structural specification

- Dependency graph
  - Can show if there are no contradictions (cycles)
  - Shows that we can have shared dependencies



Make is a very commonly-used method for specifying build systems

- If you want more examples look at any C-based open source projects
- No need to know make in detail for the exam
  - Just what I discussed in the slides

Semi-formal method: UML

- Unified Modeling Language (UML)
- A consolidation of the best practices of modeling languages over the years
- General purpose language
- Based on 14 types of diagrams
  - We will only discuss a few :)

The purpose of models

- Give an abstraction of a system
- Models as a sketch
- Models as a blueprint
- Models as executable programs

The UML diagrams that you should know

- Use case diagram
- State machine diagram
- Class diagram
- Sequence diagram
- Activity diagram

The UML diagrams that you should know

- Use case diagram
- State machine diagram
- Class diagram
- Sequence diagram
- Activity diagram