Review 9: Week of Mar 24

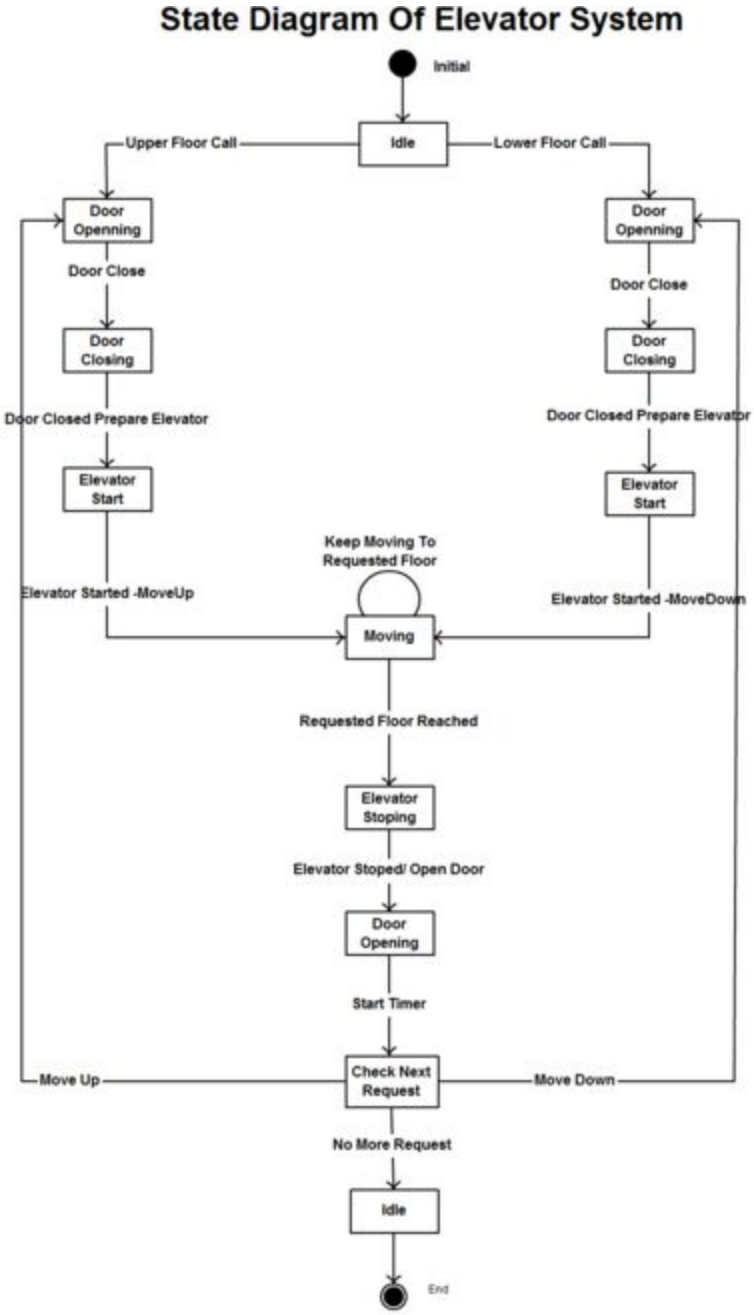
Visual Notation for Programming

1. Let us say that a 'useful diagram' is close to some code base. List two visual notations that are 'useful'. Justify your answer.
    1) **State charts**: Directly translatable to executable code.
    Statechart diagrams are useful for modelling the lifetime of an object.
    A statechart diagram shows flow of control from state to state.
    BUT:
        - hard to maintain unless auto-generated from some higher-level spec
        - hard to scale (all that detail! oh my!)
        - also, very low-level: hard to get that much information about all systems.
        - Cannot be used when interfacing to another black-box system you know nothing about
        - Irrelevant for the mash-up world (no knowledge of internals)
    2) **ER Diagrams**: Since code changes all the time, it does not document the code, but the data it runs on.
        - Diagram that can be directly implemented in SQL tools (design straight to running code! hooray!)
        - We've discussed before how this kind of ER modeling (and SQL) can actually slow up agile development (everybody start chanting NOSQL! NOSQL!)
        - Also, Once the data is modeled this way, you still need to write the associated procedural code for GUIs, intricate business logic etc etc
    3) **Compartmental Models**: Stocks, flows, stuff sloshing around some pipes.

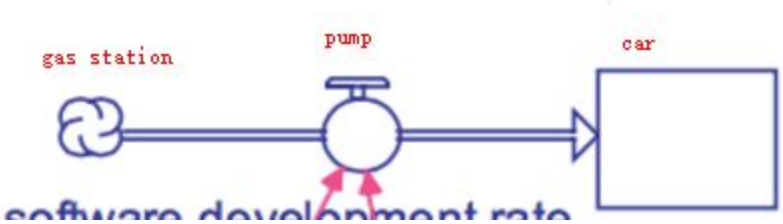
    a.Easy to code:
        - Flows change stuff (and stuff is called Stocks).
        - Stocks are real-valued variables , some entity that is accumulated over time by inflows and/or depleted by outflows.

    b. Note that the model is just some Python code so we can introduce any shortcut function (e.g. saturday).
    c. Can be used to model (e.g.) time-dependent business processes
    But: many (most) programming tasks are time-less. So CM is not applicable to general models.

2. Define the main parts of a state chart. Give a small example of state chart diagram of a person waiting for an elevator.
    - Black dot = state
    - Dashed lines: parallel work
    - Words on arcs: transition guards
    - Solid line: nested sub-state machines (and all transitions on super-states apply to sub-states)



3. Define the main parts of compartmental models. Give a small example using the scenario of filling your car with gasoline.
   Stocks, flows, stuff sloshing around some pipes.



4. Define the main parts of ER diagram. Give a small example using the scenario of placing a book order on Amazon.com.
   User table, Order table, Book table

5. What are the limitations of compartmental models?
   Many (most) programming tasks are time-less. So CM is not applicable to general models.

6. How is UML similar to ER diagrams?
   The entity model provided by ER and UML class diagrams is almost the same, they both provide information about relationships.

7. How is UML not similar to ER diagrams?
   While both models provide for relationships, UML can be more descriptive in its relationships which often reflect some sort of action, behavior, or complex relationships such as interface implementation or inheritance. ERD relationships, on the other hand, are all about keys and multiplicity.

8. How does compartmental model differ from UML?
   UML deals with architecture, rather than with the whole system.

9. Why it is bad to get the entity wrong early in the development cycle in ER diagrams?

    - In the research result, ER diagrams are used primarily as a communication and collaboration mechanism where there is a need to solve problems and/or get a joint understanding of the overall design in a group. So early wrong will cause an overall misunderstand to the project.

- Also in the research result, models are seldom updated after initially created. Once it is wrong at early time, people rarely fix this problem and may let “the after joining people” mass up.

10. What kind of visual notation is the following diagram? State and explain all of it's parts. CRC card.

Class CardReader	
Responsibilities	**Collaborators**
Tell ATM when card is inserted	ATM
Read information from card	Card
Eject card	
Retain card	

11. Propose and draw a class diagram equivalent to the above diagram.

Variable Index

- **atm**
  The ATM to which this card reader belongs

Constructor Index

- **CardReader**(ATM)
  Constructor

Method Index

- **ejectCard**()
  Eject the card that is currently inside the reader.
- **readCard**()
  Read a card that has been partially inserted into the reader
- **retainCard**()
  Retain the card that is currently inside the reader for action by the bank.

12. Why we should not build dialogs directly from the data?
    Concurrency, safety.
    - More time is needed to prepare data before build dialogs. Time to look at data, analyze data.
    - Time to look at the data rather than time spent creating the graphs and charts
    - People need to plan and implement strategies and then collect data to see if the strategies work
    - Data may be hard to collect or sufficiently disaggregated.
13. When and for what purposes should you use visual notations?

- Visual systems are more motivating for beginners than textual systems.
- In the case of *spatial reasoning problems* (e.g. finding an "as the crow flies" path between two points on a paper), a picture may indeed be worth 10,000 words. Given some 2-D representation of a problem (e.g. an array representation), spatial reasoning can make certain inferences very cheaply.
- Also, ill-structured diagramming tools are a very useful tool for brainstorming ideas

Other notes about diagram:

- Many software engineering and knowledge engineering problems are not inherently spatial.

- Many visual programming systems do not support mucking around with ill-structured approach to brainstorming.
- Claims as to the efficacy of VP systems have been poorly documented.
- Diagrams often over-elaborated with spurious detail (see case study at end on model-itis)
- Diagramming can work well for small tasks but scale up is a problem
- Writing diagrams can be slow, Getting all that detail
- Diagrams are be completed (hooray!) and still miss important aspects of the system

Case study: *model-itis*

Definition: the obsessive and needless over-elaboration of descriptions of a system

- Models become out of data, no one uses them
- Time is wasted maintaining dull models.

Case study of *model-itis*: options for software configuration:

- Understanding options are important, especially when combining different systems.
- Models of config options illustrate the strengths and weaknesses of models.

Summarizing the answers of 3785 developers answering the simple question on the extent to which design models are used before coding

- The use of models decreased with an increase in experience and increased with higher level of qualification.
- Overall we found that models are used primarily as a communication and collaboration mechanism where there is a need to solve problems and/or get a joint understanding of the overall design in a group.
- We also conclude that models are seldom updated after initially created and are usually drawn on a whiteboard or on paper.

Prolog

1. Why prolog can be used in a mass distributed system easily?
2. What is clause reordering in Prolog? Why is it important? Give an example. What other languages uses clause reordering?
   When declaring clauses in prolog (specifically recursive ones), the ordering of the clause is important. Consider the following prolog program:
   numeral(succ(X)) :- numeral(X).
   numeral(0).

If we want to generate numerals by giving it the query: numeral(X). This program would not halt because the program is going to generate using the first clause which will recurse indefinitely. If we reverse the order, the second rule will satisfy and give 0.

This concept is also extended to another semi-logical language - SQL, a very good example is the query: find all american cousins of Sally. This can be achieved through two approaches:

1. Find all americans then find among them the cousins of Sally.
2. Find all cousins of Sally and then find the american ones among them.

Clearly the second is more efficient than the first, simply because of the order of the queries.

3. From the given facts below, write a function in Prolog to find a food's flavor. a) Use your function to find the flavor of 'velveeta'. b) Use your function to find the the foods of 'savory' flavor.

```
food_type(velveeta, cheese). food_type(ritz, cracker).
food_type(spam, meat). food_type(sausage, meat).
food_type(jolt, soda). food_type(twinkie, dessert).
flavor(sweet, dessert). flavor(savory, meat).
flavor(savory, cheese). flavor(sweet, soda).
```

flavor_food(flavor, food) :- flavor(flavor, type), food_type(food, type).

4. Consider the following facts and function in Prolog. Will the following two coloring result in 'Yes' or 'No' in Prolog?

```
different(red, green). different(red, blue).
different(green, red). different(green, blue).
different(blue, red). different(blue, green).
coloring(Alabama, Mississippi, Georgia, Tennessee, Florida) :-
 different(Mississippi, Tennessee),
 different(Mississippi, Alabama),
different(Alabama, Tennessee),
different(Alabama, Mississippi),
different(Alabama, Georgia),
different(Alabama, Florida),
  different(Georgia, Florida),
 different(Georgia, Tennessee).

% Coloring 1
```

```
Alabama = blue.
Florida = green.
Georgia = red.
Mississippi = red.
Tennessee = green.

% Coloring 2
Alabama = red.
Florida = green.
Georgia = green.
Mississippi = red.
Tennessee = blue.
```

1: yes; 2: no

5. Explain what the following prolog code does.

```
append([], List, List).
append([Head|Tail], List, [Head|Rest]) :- append(Tail, List, Rest).
```

6. A predicate is not a function - discuss.

Requirements Engineering

1. What is requirements document? What are the properties of requirements document?
    1) The official statement of what is required of the system developers.
    2) Should include both a definition and a specification of requirements;
       It is NOT a design document;
       As far as possible, it should set of WHAT the system should do rather than HOW it should do it;
       Also, it should have tests that can be applied incrementally.
2. State the components of the requirements document structure and explain them.
    1) Introduction
       • Describe need for the system and how it fits with business objectives
    2) Glossary
       • Define technical terms used

3) System models
   • Define models showing system components and relationships
4) Functional requirements definition
   • Describe the services to be provided
   • User stories go here
   • Add in notes for the commit partition here
5) Non-functional requirements definition
   • Define constraints on the system and the development process
   • Add in notes for the commit partition here

2) Constraints
   • Add in notes for the commit partition here
3) System evolution
   • Define fundamental assumptions on which the system is based and anticipated changes
4) Requirements specification
   • Detailed specification of functional requirements
5) Appendices
   • System hardware platform description
   • Database requirements (as an ER model perhaps)
6) Index

3. Define functional requirements. Give an example.
   the functionality of the product
   - Scope of the Product - defines the product boundaries and its connections to adjacent systems;
   - Functional & Data Requirements - things the product must do and the data manipulated by the functions
4. Define non-functional requirements. Give an example.
   the products qualities
   - Look & Feel Reqt's - the intended appearance
   - Usability Reqt's - based on the intended users
   - Performance Reqt's - how fast, big, accurate, safe reliable, etc.
   - Operational Req'ts - the product's intended operating envt.
   - Maintainability & Portability Reqt's - how changeable the product must be
   - Security Reqt's - the security, confidentiality & integrity of the product
   - Cultural & Political Reqt's - human factors
   - Legal Reqt's - conformance to applicable laws
5. Define product constraints. Give examples.
   restrictions & limitations that apply to the product & problem

- Purpose of Product - the reason for building it and the business advantage if we do so
- Stakeholders - the people with an interest in the product
- Users - the intended end-users, & how they affect the product's usability
- Requirements Constraints - limitations on the project & restrictions on product's design
- Naming Conventions & Definitions - vocabulary of the product
- Relevant Facts - outside influences that make some difference to this product
- Assumptions - that the developers are making

6. State and explain three types of product constraints.
    - Purpose of Product - the reason for building it and the business advantage if we do so
    - Stakeholders - the people with an interest in the product
    - Users - the intended end-users, & how they affect the product's usability
    - Requirements Constraints - limitations on the project & restrictions on product's design
    - Naming Conventions & Definitions - vocabulary of the product
    - Relevant Facts - outside influences that make some difference to this product
    - Assumptions - that the developers are making
7. What are the issues encountered in product requirements.
    - Open Issues - as yet unresolved issues w/ a possible bearing on the product's success
    - Off-the-Shelf Solutions - components that may be used instead of building something
    - New Problems - caused by the introduction of new product
    - Tasks - things to be done to bring the product into production
    - Cutover - tasks to convert from existing systems
    - Risks - the risks the project is most likely to face
    - Costs - early estimates of cost or effort needed to build it
    - User Documentation - plan for building user documentation
    - Waiting Room - req'ts to be included in future releases
8. What are the identifiable requirements classes. Explain and give examples for each.

1) Enduring requirements

   • Stable requirements derived from the core activity of the customer organization. E.g. a hospital will always have doctors, nurses, etc.

   • May be derived from domain models

2) Volatile requirements

   • Requirements which change during development or when the system is in use. E.g. In a hospital, requirements derived from health-care policy

9. What is a stakeholder in software?

   System stakeholders are people or organisations who will be affected by the system and who have a direct or indirect influence on the system requirements

10. Why are stakeholders important when reasoning about software?

    Active stakeholder participation is vital to the success of IT projects. Stakeholders should be kept engaged throughout the life of the project. Without the fully engaged cooperation of the stakeholders, it is difficult to have a successful software development project. This is especially true in agile development, where everything is moving very quickly and so much depends on the Product Owner obtaining a complete understanding of the stakeholder needs and communicating these needs to the development team.