CS 520
Theory and Practice of Software Engineering
Fall 2021

Requirements specification

September 23, 2021
In-class exercise 1:
Advanced uses of Git

• **Due:** Tuesday September 28, 2021, 9:00 PM

• **Submission:**
  – Could either work collaboratively as a group (on your own video call)
  – Alternatively, could work individually and write up as a shared document (e.g., over email, Google Drive)
Homework 1: Tic Tac Toe game app

- **Due:** Thursday September 30, 2021, 9:00 PM

- **Code review:** Non-functional requirements, best practices

- **Architecture:** MVC (Model-View-Controller) pattern

- **Proposed extension**
Today

• Requirements Engineering
• Specification
  – Natural language
  – Finite State Automata (FSAs)
  – Specification patterns
• Final project
Requirements Engineering: What is a software requirements specification?

• Documents the assumptions about, features requested, and behavior of a given software application expected by the users

• Defines a set of requirements that must be satisfied by the software application
Requirements Engineering: Two key types of requirements

• **Non-functional requirement**: A quality constraint on the software application
  – e.g., extensibility, scalability, usability

• **Functional requirement**: An intended (or unintended) behavior of the software application
  – e.g., The Tic Tac Toe game must use the MVC architecture pattern.

*NOTE* There are other types of requirements to describe assumptions, features, and usage scenarios (e.g., UML use cases).
Requirements Engineering: Two key types of requirements

- **Non-functional requirement**: A quality constraint on the software application
  - e.g., extensibility, scalability, usability

- **Functional requirement**: An intended (or unintended) behavior of the software application
  - e.g., The Tic Tac Toe game must use the MVC architecture pattern.

**NOTE** There are other types of requirements to describe assumptions, features, and usage scenarios (e.g., UML use cases).
Requirements Engineering: Phases

1. Elicitation
2. Specification
3. Analysis
4. Management

Requirement → Design → Implementation
Requirements Engineering: Phases

1. Elicitation
2. Specification
3. Analysis
4. Management

Requirement ➔ Design ➔ Implementation

Requirement ← Design ← Implementation
Requirements Engineering: Phases

1. Elicitation
2. Specification
3. Analysis
4. Management

Requirement → Design → Implementation

Requirement ← Design ← Implementation
Example: Tic Tac Toe game app (from Homework 1)

• Apply the MVC (Model-View-Controller) architecture pattern
• Follow Tic Tac Toe game rules
Specication:
Natural language

• Elicitation often produces requirements written in natural language

• Sample:
  1. The ‘Tic Tac Toe’ game app must use the MVC architecture pattern.
     a. There must be a single game model created.
     b. Once the new game model is created, that model can have its state changed.
     c. If the game model has a state change, then its game view must be updated.
     d. ...
  2. ...

12
Specification: Disadvantages of natural language

• Natural language is often ambiguous
• Such ambiguity can lead to human misunderstandings
• The ambiguity also means verification & validation cannot be carried out by applying automated tools, e.g.,
  – Testing, model checking
Specification: Disadvantages of natural language

• Natural language is often ambiguous
• Such ambiguity can lead to human misunderstandings
• The ambiguity also means verification & validation cannot be carried out by applying automated tools, e.g.,
  – Testing, model checking

Therefore mathematical notations (e.g., finite state automata) are commonly used because they are rigorously defined
Specification:
Finite state automaton (FSA)

Event sequence over an alphabet (meaning a set of events)

Finite state automaton (FSA)

or

accept

reject
There must be a single *game model* created.

FSA: Sample 1a
FSA: Sample 1a

There must be a single game model created.
FSA: Sample 1a

There must be a single game model created.

Event sequence: [ gameModelNew, gameModelNew ]
FSA: Sample 1a

There must be a single game model created.

Event sequence: [
    gameModelNew,
    gameModelNew
]

reject
FSA: Sample 1a

There must be a single game model created.

Event sequence: [
  gameModelNew
]

1

→

gameModelNew

2
FSA: Sample 1a

There must be a single game model created.

Event sequence: [
  gameModelNew
]

accept
There must be a single game model created.

Event sequence: []
FSA: Sample 1a

There must be a single game model created.

Event sequence: []

reject
FSA: Sample 1b

gameModelNew -> 1

gameModelStateChange -> 2

gameModelStateChange -> 3

1 -> 2

2 -> 3
Some event sequences that are accepted (i.e. positive examples)?
Some event sequences that are accepted (i.e. positive examples)?

- [gameModelNew]
- [gameModelNew, gameModelStateChange]
FSA: Sample 1b

Other event sequences that are rejected (i.e. negative examples)?

- gameModelNew
- gameModelStateChange
- gameModelStateChange
Other event sequences that are rejected (i.e. negative examples)?

[]

[gameModelStateChange]

[gameModelNew, gameModelNew]
FSA: Sample 1b

Once the new game model is created, that model can have its state changed.

gameModelNew  gameModelStateChange

1  2  3

gameModelStateChange
FSA: Formal definition

Represented as a 5 tuple:

- **E**: The alphabet as a set of events
- **S**: The set of states
- **T**: The set of transitions where each transition is a directed edge from a source state $s$ on event $e$ to target state $t$
- **$s_0$**: The start state
- **A**: The set of accepting states

NOTE) Can be automatically converted to a regular expression
FSA: Disadvantages

• Natural language is generally more accessible than FSAs
• Hand writing FSAs can be error prone and time consuming
FSA: Disadvantages

- Natural language is generally more accessible than FSAs
- Handwriting FSAs can be error prone and time consuming

Specification patterns have been identified of commonly occurring intended (or unintended) behaviors of software applications
Property specification pattern: Formal definition

• Consists of a:
  – **Scope** captures parts of the trace where the behavior must be satisfied (e.g., Global)
  – **Behavior** captures occurrence or order of events/propositions (e.g., Bounded existence of *gameModelNew*)

• Provides mapping to various property specification languages (e.g., Regular Expressions, in this case *gameModelNew*)

[https://matthewbdwyer.github.io/psp/]
Property specification pattern: Scopes

Global

Before R

After Q

Between Q and R

After Q until R
Property specification pattern: Behaviors

Behavior

- Occurrence of event/proposition A
- Order of events/propositions A and B

Absence
- Universality
- (Bounded) Existence
- (Chained) Precedence
- (Chained) Response
Property specification pattern: Behaviors

- Occurrence of event/proposition A
- Order of events/propositions A and B
- Absence
- Universality
- (Bounded) Existence
- (Chained) Precedence
- (Chained) Response
Property specification pattern: Precedence description

Intent

To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second. We say that an occurrence of the second is enabled by an occurrence of the first.

Example Mappings

In these mappings $P$ is the consequent and $S$ is the enabling state/event.

- CTL
- LTL
- QRE
- INCA
- GIL

Examples and Known Uses

Precedence properties occur quite commonly in specifications of concurrent systems. One common example is in describing a requirement that a resource is only granted in response to a request.

Relationships

Note that a Precedence property is like a converse of a Response property. Precedence says that some cause precedes each effect, and Response says that some effect follows each cause. They are not equivalent, because a Response allows effects to occur without causes (Precedence similarly allows causes to occur without subsequent effects).

Note that this pattern does not require that each occurrence of a consequent will have its own occurrence of an enabling condition.

This is an Order pattern.
**Property specification pattern:**

**Precedence regular expressions**

### Precedence

$s$ precedes $p$:  

| Globally          | $\lbrack-P\rbrack^* | (\lbrack-\lbrack-S, P\rbrack^*; S; .*)\rbrack^* $ |
|-------------------|----------------------|
| Before $r$        | $\lbrack-R\rbrack^* | (\lbrack-\lbrack-P, R\rbrack^*; R; .*) | (\lbrack-\lbrack-S, P, R\rbrack^*; S; .*)\rbrack^* |
| After $q$         | $\lbrack-Q\rbrack^*; Q; (\lbrack-\lbrack-P\rbrack^* | (\lbrack-\lbrack-S, P\rbrack^*; S; .*)\rbrack^* $ |
| Between $q$ and $r$ | $\lbrack-Q\rbrack^*; Q; (\lbrack-\lbrack-P, R\rbrack^* | (\lbrack-\lbrack-S, P, R\rbrack^*; S; [\lbrack-R\rbrack^*])\rbrack^*; R; [\lbrack-Q\rbrack^*])^*; (Q; [\lbrack-R\rbrack^*])? |
| After $q$ until $r$ | $\lbrack-Q\rbrack^*; Q; (\lbrack-\lbrack-P, R\rbrack^* | (\lbrack-\lbrack-S, P, R\rbrack^*; S; [\lbrack-R\rbrack^*])\rbrack^*; (Q; ([\lbrack-P, R\rbrack^* | (\lbrack-\lbrack-S, P, R\rbrack^*; S; [\lbrack-R\rbrack^*])\rbrack^* ))? $ |
Property specification pattern: Precedence regular expressions

Precedence

$s$ precedes $p$:

| Globally                      | $[-P]^* | ([-S,P]^*; S; .*)$ |
|-------------------------------|----------------------|
| Before $r$                   | $[-R]^* | ([-P,R]^*; R; .*) | ([-S,P,R]^*; S; .*) |
| After $q$                    | $[-Q]^*; (Q; ([-P]^* | ([-S,P]^*; S; .*)) )? |
| Between $q$ and $r$          | $[-Q]^*; (Q; [-P,R]^* | ([-S,P,R]^*; S; [-R]^*) R; [-Q]^*)^*; (Q; [-R]^*)^* |
Once the new game model is created, that model can have its state changed.

Behavior: `gameModelNew` must precede `gameModelStateChange`

Scope: Globally
Property specification patterns: Disadvantages

• Use 80/20 rule
  – May need to hand write FSAs

• Don’t provide support for real-time or probabilistic constraints
  – Has been extended [See http://ps-patterns.wikidot.com]

• Designed for experts
  – Can be difficult for novices to select among patterns and customize them
Property specification patterns: Disadvantages

• Use 80/20 rule
  – May need to hand write FSAs

• Don’t provide support for real-time or probabilistic constraints
  – Has been extended [See http://ps-patterns.wikidot.com]

• Designed for experts
  – Can be difficult for novices to select among patterns and customize the selected pattern
PROPEL tool: What?

- Builds on the property specification patterns specified as finite state automata
- Provides guidance to select among the patterns and customize that pattern

[http://laser.cs.umass.edu/tools/prope1.shtml]
PROPEL tool: User interface

Scope

Behavior

[Available from http://laser.cs.umass.edu/release/]
Specify the following:

“If the game model has a state change then its game view must be updated”
PROPEL tool: Demonstration

Specify the following:

“If the game model has a state change then its game view must be updated”
Behavior: gameModelStateChange must have response gameViewUpdate
Scope: Globally
PROPEL tool: Sample DNL 1c

BEHAVIOR:

1. The events of primary interest in this behavior are gameModelStateChange and gameViewUpdate.

2. There are no events of secondary interest in this behavior.

3. If gameModelStateChange occurs, gameViewUpdate is required to occur subsequently.

4. Before the first gameModelStateChange occurs, gameViewUpdate is allowed to occur zero or more times.

5. gameModelStateChange is not required to occur.

6. After gameModelStateChange occurs, but before the first subsequent gameViewUpdate occurs, gameModelStateChange is not allowed to occur again.

7. After gameModelStateChange and the first subsequent gameViewUpdate occur:

   - gameViewUpdate is not allowed to occur again until after another gameModelStateChange occurs;

   - gameModelStateChange is allowed to occur again and, if it does, then the situation is
Final project selection

• Form team of 4 or 5 students
• Select one of the following 4 topics:
  1. MSR mining challenge
  2. Replication study
  3. ML development toolkit
  4. EleNa: Elevation-based navigation
• Due: Tuesday October 5, 2021 9:00 PM

https://people.cs.umass.edu/~hconboy/class/2021Fall/CS520/finalProject.pdf
Final project: Selected topic

1. Read some background materials (e.g., papers, user manuals, code)
2. Start to develop
3. Create and give a mid-point presentation
4. Continue to develop
5. Create and give a final presentation
6. Put together final deliverables
Final project: MSR mining challenge objectives

• Read 8-10 papers
• Select one or more research questions
• Propose an approach to investigate the research question(s)
• Develop experiments to evaluate the proposed approach by applying to the provided dataset
• Write up the approach and experimental results


https://2021.msrconf.org/track/msr-2021-mining-challenge
Final project: Replication study objectives

- Read 4-5 papers
- Select one of the debugging/testing tools (e.g., SimFix)
  - Learn about the benchmark (i.e. initial data) for the selected tool
- Replicate the experiments to evaluate the selected tool by applying to the initial data
  - Extend the experiments to further evaluate the selected tool on additional data
- Write up the replication study
Final project: ML development toolkit

• Read any necessary documentation
• Select one or more ML development toolkits (e.g., Weights & Biases)
  – Also select one or more data sets
• Develop experiments to evaluate the selected tool(s) by applying to the selected data set(s)
• Write up the experimental results
  – Including any Software Engineering topics used by these toolkits

https://wandb.ai/site
Final project: Elevation-based Navigation (EleNa)

- **Goal:** Develop a software system that determines, given a start and an end location, a route that maximizes or minimizes elevation gain, while limiting the total distance between the two locations to x% of the shortest path.

- **Components:**
  - Data model that represents the geodata
  - A component that populates the data model, querying, e.g., [https://www.openstreetmap.org](https://www.openstreetmap.org)
  - The actual routing algorithm that performs the multi-objective optimization
  - Another component that outputs or renders the computed route
Final project: EleNa objectives

• Read any necessary technical documents
• Design 4 main components: At least 2 complex
• Implement the designed components
• Build a test plan for the implemented components and carry out that test plan
• Could additionally perform other evaluation of the system
• Demo at the final presentation
• Submit your documentation and version control repository