CS 520
Theory and Practice of Software Engineering
Fall 2022

Requirements specification

October 6, 2022
Today

• Requirements Engineering

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

- Documents the assumptions about, features requested, and behavior of a given software application excepted 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
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
Example: Basic Stats app
(from Homework 1)

Visualize some common summary statistics of a user-defined list of doubles
Specification: Natural language

• Elicitation often produces requirements written in natural language

• Sample:
  1. There is a single UI created.
  2. Once a single UI is created, that UI allow user interactions.
  3. After a user interaction in the UI, update the visualizations in it
     - ...

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
FSA: Sample 1a

There must be a single UI created.
There must be a single UI created.
There must be a single UI created.

Event sequence: [
  uiNew,
  uiNew
]

FSA: Sample 1a
FSA: Sample 1a

There must be a single UI created.

Event sequence: [
uiNew,
uiNew
]
reject
There must be a single UI created.

Event sequence: [ uiNew ]
FSA: Sample 1a

There must be a single UI created.

Event sequence: [
  uiNew
]

accept
FSA: Sample 1a

There must be a single UI created.

Event sequence: []
FSA: Sample 1a

There must be a single UI created.

Event sequence: []

1 -> uiNew -> 2

reject
FSA: Sample 1b

1 -> uiNew -> 2
2 -> userInteraction -> 3
3 -> userInteraction

Diagram of a Finite State Automaton (FSA) with states 1, 2, and 3, labeled with transitions uiNew and userInteraction.
FSA: Sample 1b

1. Some event sequences that are accepted (i.e. positive examples)?

2. uiNew

3. userInteraction

23
Other event sequences that are rejected (i.e. negative examples)?
FSA: Sample 1b

Once a single UI is created, that UI allows user interactions.
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
• Hand writing 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 $p$:

<table>
<thead>
<tr>
<th></th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Globally</strong></td>
<td>$[-P]^* \mid ([S,P]^<em>; S; .</em>)$</td>
</tr>
<tr>
<td><strong>Before $r$</strong></td>
<td>$[-R]^* \mid ([P,R]^<em>; R; .</em>) \mid ([S,P,R]^<em>; S; .</em>)$</td>
</tr>
<tr>
<td><strong>After $q$</strong></td>
<td>$[-Q]^<em>; (Q; ([P]^</em> \mid ([S,P]^<em>; S; .</em>)) )?$</td>
</tr>
<tr>
<td><strong>Between $q$ and $r$</strong></td>
<td>$[-Q]^<em>; (Q; [-P,R]^</em> \mid ([S,P,R]^<em>; S; [-R]^</em>); R; [-Q]^* )^<em>; (Q; [-R]^</em>) )?$</td>
</tr>
<tr>
<td><strong>After $q$ until $r$</strong></td>
<td>$[-Q]^<em>; (Q; [-P,R]^</em> \mid ([S,P,R]^<em>; S; [-R]^</em>); R; [-Q]^* )^<em>; (Q; ([P,R]^</em> \mid ([S,P,R]^<em>; S; [-R]^</em>)) )?$</td>
</tr>
</tbody>
</table>
Property specification pattern: Precedence regular expressions

Precedence

$s$ precedes $p$:

<table>
<thead>
<tr>
<th>Globally</th>
<th>$[-P]^* \mid ([S,P]^<em>; S; .</em>)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before $r$</td>
<td>$[-R]^* \mid ([P,R]^<em>; R; .</em>) \mid ([S,P,R]^<em>; S; .</em>)$</td>
</tr>
<tr>
<td>After $q$</td>
<td>$[-Q]^<em>; (Q; ([P]^</em> \mid ([S,P]^<em>; S; .</em>)) )?$</td>
</tr>
<tr>
<td>Between $q$ and $r$</td>
<td>$[-Q]^<em>; (Q; [P,R]^</em> \mid ([S,P,R]^<em>; S; [-R]^</em>) R; [-Q]^<em>)^</em>; (Q; [-R]^<em>)^</em>$</td>
</tr>
<tr>
<td>After $q$ until $r$</td>
<td>$[-Q]^<em>; (Q; [P,R]^</em> \mid ([S,P,R]^<em>; S; [-R]^</em>) R; [-Q]^<em>)^</em>; (Q; ([P,R]^* \mid ([S,P,R]^<em>; S; [-R]^</em>)) )?$</td>
</tr>
</tbody>
</table>
Once a single UI is created, that UI allows user interactions.

Behavior: uiNew must precede userInteraction

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/propel.shtml]
PROPEL tool: User interface

[Available from http://laser.cs.umass.edu/release/]
PROPEL tool: Demonstration

Specify the following:

“After a user interaction in the UI, update the visualizations in it”
PROPEL tool: Demonstration

Specify the following:

“After a user interaction in the UI, update the visualizations in it”
**Behavior:** userInteraction must have response viewUpdate

**Scope:** Globally
PROPEL tool: Sample DNL 1c

**SCOPE:**

1. From the start of any event sequence through to the end of that event sequence, the behavior must hold.

**BEHAVIOR:**

1. The events of primary interest in this behavior are `userInteraction` and `viewUpdate`.

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

3. If `userInteraction` occurs, `viewUpdate` is required to occur subsequently.

4. Before the first `userInteraction` occurs, `viewUpdate` is allowed to occur zero or more times.

5. `userInteraction` is not required to occur.

6. After `userInteraction` occurs, but before the first subsequent `viewUpdate` occurs, `userInteraction` is
“State machine diagram is a behavior diagram which shows discrete behavior of a part of designed system through finite state transitions.

State machine diagrams can also be used to express the usage protocol of part of a system.”

[https://www.uml-diagrams.org/state-machine-diagrams.html]
Example:
UML State Machine Diagram
Final project: Group and topic selection

• Form team of 2, 3, or 4 students
• Select one of the following 3 topics:
  1. MSR mining challenge
  2. Experimental evaluation (replication study or tool comparison)
  3. Development (e.g., EleNa)
     NOTE) Could propose your own
• Due: Thursday October 6, 2022, 11:59 PM