CS 520
Theory and Practice of Software Engineering
Spring 2021

Software requirements

March 2, 2021
In-class exercise 1: Advanced uses of Git

• **Due:** Wednesday March 3, 2021, 9 AM EDT

• **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:
Three in a Row game app

• **Due:** This Saturday March 6, 9 PM EDT

• **Code review:** OO design principles, best programming practices

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

• **Design:** Observer pattern

• **Office hours:** This Thursday after lecture, This Friday 9:30 – 10 AM
Today

• Requirements Engineering
• Specification
  – Natural language
  – Finite State Automata (FSAs)
  – Specification patterns
• Final project
Requirements Engineering: Stakeholders

• “individuals and organizations who are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or successful project completion”

[Project Management Institute (PMI®), 1996]
Example: Three in a Row game app
(from Homework 1)
Example: Three in a Row game app
(from Homework 1)

• Game app players

• Game app company: Developers, upper management, user support, beta (user) testers

• UI experts: Human factors, HCI (Human-Computer Interaction)

• App store company: Developer support, Accounting department

• Federal agencies: ADA (Americans with Disabilities Act)
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 Three in a Row 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 Three in a Row 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
Requirements Engineering: Phases

1. Elicitation
2. Specification
3. Analysis
4. Management
Example: Three in a Row game app (from Homework 1)

• Apply the MVC (Model-View-Controller) architecture pattern
• Follow Three in a Row game rules
Specification: Natural language

• Elicitation often produces requirements written in natural language

• Sample:
  1. The ‘Three in a Row’ game app must use the MVC architecture pattern.
     a. There must be a single game model.
     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. ...

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 game model.
FSA: Sample 1a

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

There must be a single game model.

Event sequence: [gameModelNew, gameModelNew]
There must be a single game model.

Event sequence: 
\[\text{gameModelNew, gameModelNew}\]

reject
FSA: Sample 1a

There must be a single game model.

Event sequence: [
    gameModelNew
]
FSA: Sample 1a

There must be a single game model.

Event sequence: [ gameModelNew ]

accept ✔
FSA: Sample 1a

There must be a single game model.

Event sequence: []
FSA: Sample 1a

There must be a single game model.

Event sequence: []

reject
FSA: Sample 1b

1. gameModelNew
2. gameModelStateChange
3. gameModelStateChange
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)?

1. gameModelNew
2. gameModelStateChange
3. 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
```

**Diagram:**
- State 1
- Transition to State 2 labeled 'gameModelNew'
- Transition from State 2 to State 3 labeled 'gameModelStateChange'
- Transition from State 3 back to State 1 labeled '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
- Handwriting 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 \texttt{gameModelNew})

• Provides mapping to various property specification languages (e.g., Regular Expressions, in this case \texttt{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 enabing 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>Globally</td>
<td>$[-P]^* \mid ([S,P]^<em>; S; .^</em>)$</td>
</tr>
<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>$[-Q]^<em>; (Q; [P,R]^</em> \mid ([S,P,R]^<em>; S; [R]^</em>) R; [-Q]^<em>)^</em>; (Q; [-R]^*)^?$</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>
Property specification pattern: Precedence regular expressions

**Precedence**

s precedes p:

<table>
<thead>
<tr>
<th>Globally</th>
<th>$[-P]^* \mid ([-S,P]^* ; S ; .*)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before r</td>
<td>$[-R]^* \mid ([-P,R]^* ; R ; .<em>) \mid ([-S,P,R]^</em> ; S ; .*)$</td>
</tr>
<tr>
<td>After q</td>
<td>$[-Q]^* ; (Q ; ([-P]^* \mid ([-S,P]^* ; S ; .*)))$</td>
</tr>
<tr>
<td>Between q and r</td>
<td>$[-Q]^* ; (Q ; [-P,R]^* \mid ([-S,P,R]^* ; S ; [-R]^<em>)) R ; [-Q]^</em>))) \mid (Q ; [-R]^*))$</td>
</tr>
<tr>
<td>After q until r</td>
<td>$[-Q]^* ; (Q ; [-P,R]^* \mid ([-S,P,R]^* ; S ; [-R]^<em>)) R ; [-Q]^</em>))) \mid (Q ; ([-P,R]^* \mid ([-S,P,R]^* ; S ; [-R]^*)) )$</td>
</tr>
</tbody>
</table>
Property pattern specification: Sample 1b

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/propel.shtml]
PROPEL tool: User interface

Scope

Behavior

[Available from http://laser.cs.umass.edu/release/]
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: Group and topic selection

• Form team of 4 or 5 students
• Select one of the following 4 topics:
  1. MSR mining challenge: 2020 or 2021
  2. Replication study
  3. ML development toolkit
  4. EleNa: Elevation-based navigation
  5. Proposed your own group project
• Due: Thursday March 4, 2021 9:00 PM EDT

https://people.cs.umass.edu/~hconboy/class/2021Spring/CS520/finalProject.pdf