CS 520
Theory and Practice of Software Engineering
Spring 2022

Requirements specification

February 24, 2022
Recap: Design patterns

- Addresses a recurring, common design problem.
- Provides a generalizable solution.
- Provides a common terminology.

1. Structural
   - Composite
   - Decorator
   - ...

2. Behavioral
   - Observer
   - Strategy
   - Template (method)
   - ...

3. Creational
   - Factory (method)
   - ...
Design pattern: Decorator

```
<<interface>>

Component
+operation()

1

CompA
+operation()
read(): int
java.io.FileReader

CompB
+operation()
read(): int

Decorator
-decorated: Component
+Decorator(d: Component)
+operation()
read(): int
java.io.BufferedReader
```
Design pattern: Template method

- The template method (median) implements the algorithm but leaves the sorting of the array undefined.

- The concrete subclass only needs to implement the actual sorting.
“median” delegates the sorting of the array to a “sortStrategy”
Template method vs. strategy

Two solutions to the same problem

Template method
- Behavior selected at compile time.
- Template method is usually final.

Strategy
- Behavior selected at runtime.
- Composition/aggregation over inheritance.
Here is the Java Arrays utility class: https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/Arrays.html

We’ll focus on the following method:
public static <T> void sort(T[] a, Comparator<? super T> c)

This method takes as input a Comparator class. Which design pattern is being applied?
• Composite
• Decorator
• Strategy
Here is the Java Collections utility class:
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/Collections.html

We’ll focus on the following method:

```java
public static <T> List<T> unmodifiableList(List<? extends T> list)
```

For this method, which design pattern is being applied?
- Composite
- Decorator
- Strategy
Here is the Java AbstractList class: https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/AbstractList.html

Here is the ArrayList class: https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ArrayList.html

Which design pattern is being applied?
- Decorator
- Strategy
- Template (method)
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

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: Tic Tac Toe game app (from Homework 1)

• Apply the MVC (Model-View-Controller) architecture pattern

• Follow Tic Tac Toe game rules
Specification: 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. ...
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

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

There must be a single game model created.

Event sequence: [gameModelNew, gameModelNew]
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
]

1

2
FSA: Sample 1a

There must be a single game model created.

Event sequence: [ gameModelNew ]

accept
FSA: Sample 1a

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
FSA: Sample 1b

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

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

1. `gameModelNew`
2. `gameModelStateChange`
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`)
Property specification pattern: Scopes

- **Global**
- **Before R**
- **After Q**
- **Between Q and R**
- **After Q until R**
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: 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: 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>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globally</td>
<td>$[-P]^*</td>
</tr>
<tr>
<td>Before $r$</td>
<td>$[-R]^*</td>
</tr>
<tr>
<td>After $q$</td>
<td>$[-Q]^<em>; (Q; ([P]^</em></td>
</tr>
<tr>
<td>Between $q$ and $r$</td>
<td>$[-Q]^<em>; (Q; [-P,R]^</em></td>
</tr>
<tr>
<td>After $q$ until $r$</td>
<td>$[-Q]^<em>; (Q; [-P,R]^</em></td>
</tr>
</tbody>
</table>


### Property specification pattern: Precedence regular expressions

**Precedence**

$s$ precedes $p$:

| Globally                              | $[-P]^* | ([-S,P]^*; S; .*$ |
|---------------------------------------|--------------------------------------------------|
| 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]^*$$)|
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”
PROPEL tool: Demonstration

Specify the following:

“If the game model has a state change then its game view must be updated”
Behavior: $\text{gameModelStateChange}$ must have response $\text{gameViewUpdate}$

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
Homework 1: Tic Tac Toe game app

• **Due:** Thursday February 24, 2022, 11:59 PM (a little before midnight) // Submit on Gradescope

• **Code review:** Non-functional requirements, design principles, best practices

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

• **Proposed extension**
Final project selection

- Form team of 2, 3, or 4 students
- Select one of the following 5 topics:
  1. MSR mining challenge
  2. Replication study
  3. ML development toolkit
  4. EleNa: Elevation-based navigation
  5. Propose your own
- Due: Thursday February 24, 2022, 11:59 PM

https://people.cs.umass.edu/~hconboy/class/2022Spring/CS520/finalProject.pdf