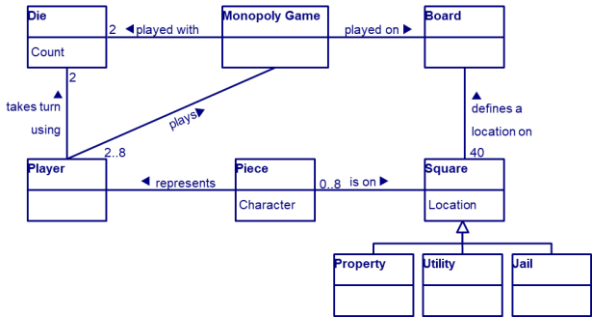
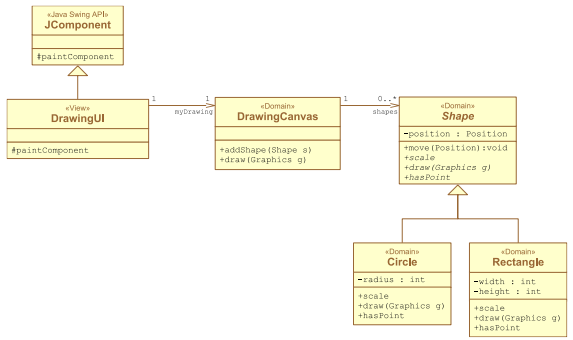
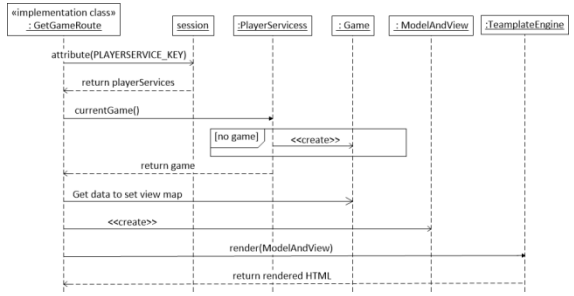
Design and Code Communication & Code Reviews








```java
/**
  * Get the {@linkplain GuessGame game} for the current user.
  * The user is identified by a {@linkplain Session browser session}.
  *
  * @param session
  *   The HTTP {@link Session}, must not be null
  *
  * @return
  *   An existing or new {@link GuessGame}
  *
  * @throws NullPointerException
  *   when the session parameter is null
  */
 public GuessGame get(final Session session)
```

SWEN-261
Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology

Software Engineering
Rochester Institute of Technology

Your communication about a project is not just in the form of presentations and meetings.

- The systems that you will develop are complex and have both static and dynamic design characteristics.
- To describe those characteristics, you will use several UML models.
  - ***Domain, class and sequence***
- Those who must use your implementation need a more productive description that studying lines of code.
- Those who must maintain your implementation must be able to quickly understand the code.

The domain model describes the product owner's understanding of the application's scope.

- Domain model
  - ***Describes the context in which the application will operate.***
  - ***Helps developers share the product owner's understanding of this context.***
  - ***Describes the product owner's world view of the domain entities and relationships between them.***
- The domain model will help developers create a structure for the implementation to the extent that is possible.

Design documentation can be a valuable communication tool.

A design document is a way for you to communicate to others what your design decisions are and why your decisions are good decisions.

From How to Write an Effective Design Document by Scott Hackett

- Design documentation should be short and easy to read.
- It should communicate key architecture and design decisions.
- It should generally move from high-level to low-level.
- It should provide justification for design decisions.

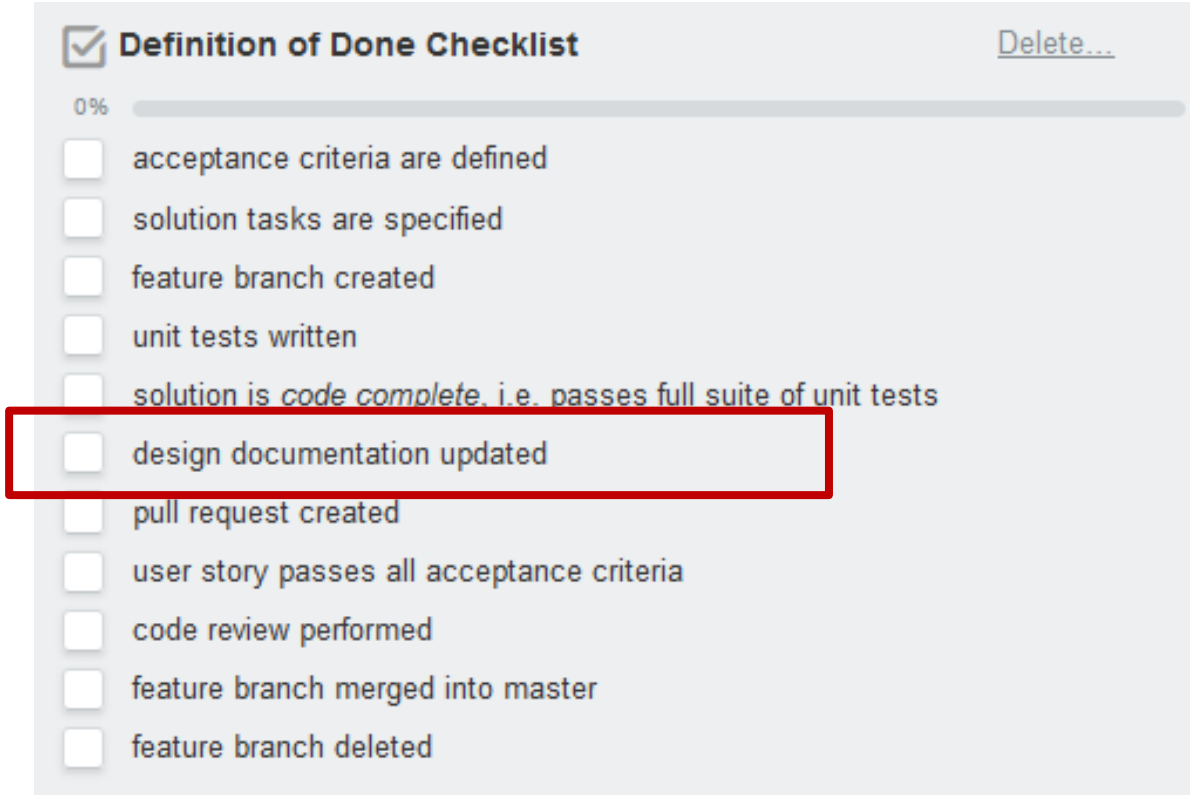
The class model defines the static structure of your implementation.

- It captures many constructs embodied in your implementation
  - ***Class attributes and methods with visibilities***
  - ***Relationships between classes with multiplicities***
  - ***Navigation between classes***
  - ***Structure via inheritance/interface***
  - ***Architectural tiers***
- The domain model inspires the first-cut for the implementation class structure.
  - ***Try to have the software structure match the product owner's domain structure, i.e. domain entities become implementation classes***

You also must describe the application's dynamic characteristics to fully describe its operation.

- An application's execution of a feature/operation involves multiple classes across architectural tiers.
  - ***The sequence diagram indicates which classes and methods are involved in an execution scenario.***
  - ***Formulate a user story solution with one or more sequence diagrams created before starting the implementation (not required for Sprint 2).***

Keeping your design documentation up-to-date will now be part of your standard workflow.



How your code "reads" is critically important for the humans who will read it.

- Code is read by humans as much as by machines.
- Code must be readable and understandable by all team members.
- Clear code communication includes:
  - ***A shared code style***
  - ***Use of good, meaningful names***
  - ***Component APIs are clearly documented***
  - ***Algorithms are clarified using in-line comments***
  - ***Indication of incomplete or broken code***

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Refactoring: Improving the Design of Existing Code
Martin Fowler, et. al (1999)

A shared code style is good etiquette.

- No code style is inherently better than any other one.
- A code style includes:
  - ***Spaces vs tabs***
  - ***Where to put curly-braces***
  - ***Naming conventions***
    - `CamelCase` for class names
    - `UPPER_CASE` for constants
    - `lowerCamelCase` for attribute and method names
  - ***And so on***
- Every team should choose a style and stick to it.
  - ***IDEs provide support for defining a code style***
  - ***If your team cannot choose one then we recommend using Google Java style (see resources)***

Make names reflect what they mean and do.

- Dos:
  - ***Use names that reflect the purpose***
  - ***Use class names from analysis and domain model***
  - ***Use method names that are verbs in your analysis***
  - ***Use method names that describe what it does not how it does it***
- Don'ts:
  - ***Don't abbreviate; spell it out***
    - `pricePerUnit` is better than pPU or worse just p
  - ***Don't use the same local variable for two purposes; create a new variable with an appropriate name***
  - ***Don't use "not" in a name***
    - `isValid` is better than `isNotValid`.

Document your component's API.

- In Java the `/**  …  */` syntax is used to denote a documentation for the thing it precedes.
- For example:

```
/**
 * Represents a Hero entity
 *
 * @author SWEN Faculty
 */
public class Hero
```

- At a minimum you should document all public members.
  - ***Also, good to document all methods including private methods***
  - ***Document attributes with complex data structures***

A method's javadoc must explain how to use the operation.

- Every method must have an opening statement that expresses what it does.
  - ***Keep this statement concise***
  - ***Additional statements can be added for clarification***
- Document the method signature
  - ***Use `@return` to describe what is returned***
  - ***Use `@param` to describe each parameter***
  - ***Use `@throws` to describe every exception explicitly thrown by the method***
- Link to other classes
  - ***Use `@link` to link to classes (`monospaced formatting`)***
  - ***Use `@linkplain` in opening statement (standard formatting)***

Example method javadocs.

Use @linkplain in the opening statement.

```
/**
 * Responds to the GET request for a {@linkplain Hero hero} for the given id
 *
 * @param id The id used to locate the {@link Hero hero}
 *
 * @return The {@link Hero hero} and HTTP status of OK if found
 *
 * @throws ResponseStatusException (NOT_FOUND) when a {@link Hero hero} with the id
 *                                  is not found
 * @throws ResponseStatusException (INTERNAL_SERVER_ERROR) if an issue with
 *          underlying storage
 */
```

Use @link in all other clauses.

getHero

```
@GetMapping("/heroes/{id}")
public org.springframework.http.ResponseEntity<Hero> getHero(@PathVariable
                                                              int id)
```

Responds to the GET request for a hero for the given id

Parameters:

`id` - The id used to locate the `hero`

Returns:

The `hero` and HTTP status of OK if found

Throws:

`org.springframework.web.server.ResponseStatusException` - (NOT_FOUND) when a `hero` with the id is not found

`org.springframework.web.server.ResponseStatusException` - (INTERNAL_SERVER_ERROR) if an issue with underlying storage

Use in-line comments to communicate algorithms and intention.

- Use in-line comments to describe an algorithm
  - ***Dos:***
    - Use pseudo-code steps
    - Explain complex data structures
  - ***Don'ts:***
    - Don't repeat the code in English

```
count++; // increment the count
```

- Use comments to express issues and intentions
  - ***A TODO comment hints at a future feature***
  - ***A FIX (or FIXME) comment points to a known bug that is low priority***

A code review can improve the quality of the product and the quality of the team.

- Increase product quality
  - ***Identify and fix design or coding violations***
  - ***Identify and fix code communication issues***
  - ***Analyze test coverage, identify new test scenarios***
- Increase overall team skill
  - ***Discuss code communication***
  - ***Share coding and testing techniques***
  - ***Discuss design principles & patterns, as appropriate***

There are several situations that warrant a code review.

- For new members of the team
  - ***Along with reading the Design documentation***
  - ***Code review (walk-through) with a senior developer***
- For Spikes
  - ***To impart lessons from the Spike to the rest of the team***
- For User Stories
  - ***To improve the quality of the feature code***
  - ***To share best practices with the rest of the team***
  - ***Even trivial stories should have reviews***

There are several code review techniques.

- Individual
  - *A senior developer sits with a junior developer*
  - *The review can be focused on a specific problem or for general understanding a subsystem*
- Synchronous
  - *A team meets to review some code*
  - *Usually the most formal process*
  - *Disadvantage of needing to sync schedules*
- Asynchronous
  - *A developer uses an online tool to create a review*
  - *Shows the diffs between two branches*
  - *Reviewers make comments in the tool*
- Hybrid approaches

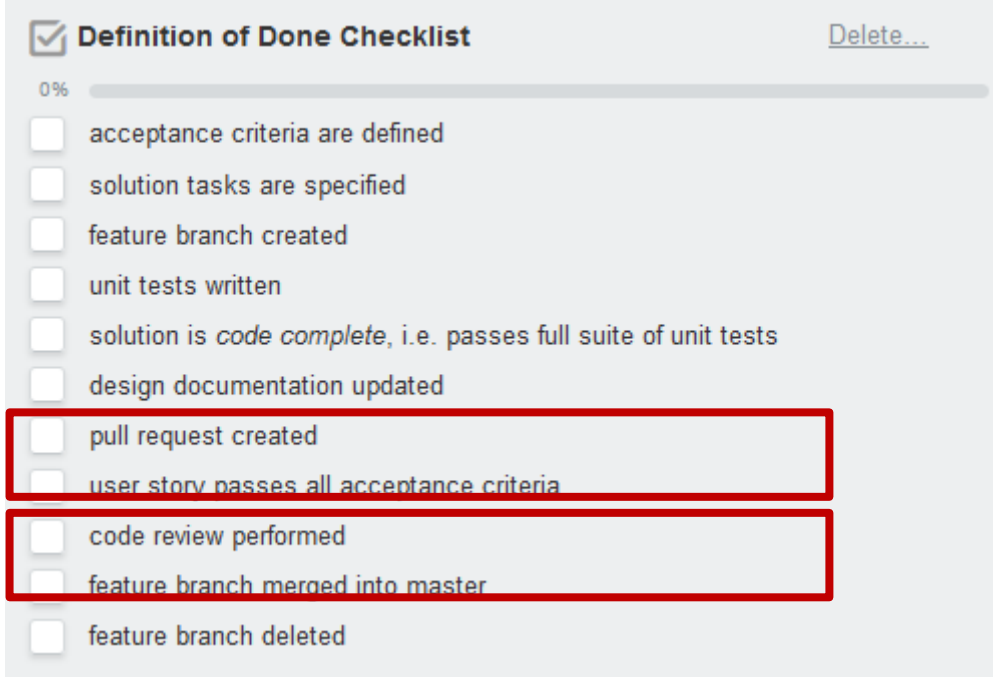
A team will often have a checklist of things to look for during the code review.

- Coding practices
  - ***Code communication***
  - ***Defensive programming practices***
- Design practices
  - ***Adherence to architectural tiers***
  - ***Adherence to core OO principles***
  - ***Adherence to OO design principles***
- Testing practices
  - ***Are test suites comprehensive (enough)***
  - ***Test code follows good code and design practices***
- Design documentation
  - ***Is the documentation being kept up-to-date***

The activity will guide the team through doing an asynchronous review.

- You will create a git *pull request* for a selected feature branch.
- Team members will review the code using GitHub's PR review user interface.
  - ***We'll provide a checklist and document to record your suggested changes***
  - ***Team submits the document to a Dropbox***
- After the changes are approved, the feature branch is merged into master

Issuing pull requests and performing code reviews will now be a part of your development workflow.



- The *Pull Request* is made when the story moves to *Ready for Test*, i.e. after the user story is code complete, and the design documentation is updated.
- Review should be done by a minimum of two team members other than the developer of the story.
- Acceptance testing can be performed in parallel.