

CS 4350: Fundamentals of Software Engineering

CS 5500: Foundations of Software Engineering

Lesson 2.1 Documenting Your Design

Jon Bell, John Boyland, Mitch Wand
Khoury College of Computer Sciences

Outline of this lesson

1. Why documenting your design is important, and why it is different from just writing comments in your program.
2. Introduction to one way of documenting your design: CRC cards

Learning Objectives for this Lesson

- By the end of this lesson you should be able to:
 - Explain what it means to document a design
 - Describe the importance of having a shared vocabulary
 - for teams,
 - for communicating with management
 - for dealing with clients
 - Illustrate the basics of CRC cards

Remember the Challenge: Controlling Complexity

- Software systems must be comprehensible by humans
- Which humans?
 - The other members of your team
 - The folks who will maintain and modify your system
 - Management
 - Your clients
 - and ...
 - You, a week from now or 6 weeks from now

A Design is more than code

- Design is about **how your code relates to the real world**
- Design is about the **organization** of the code
- Design is about the **relationships** between different pieces of the code
- So: you need a different language to talk about your design

Remember Principle
#2: Design Your
Data!

Communication Requires a Shared Vocabulary

- You and your teammates need to have a common understanding of the **things** in your program.
 - What are their names? ←
 - What do they represent?
 - How do they interact? ←

You get to make up the names. But you should make them Good Names, of course

There are standard names for many of these interactions. These are called **Design Patterns**.

Design Languages

- We'll study two design languages
 - CRC Cards
 - UML (Unified Modeling Language) [next lesson]
- These are very different languages for describing designs
 - different level of formality
 - different scope

CRC Cards

- A CRC card looks like this:

Class Name	
Responsibilities	Collaborators

CRC Cards

- Class
 - the name of a "thing" in your program
 - could be a class, interface, type, etc.
- Responsibilities
 - the main job of this "thing" in the program
 - should be simple: Remember the Single Responsibility Principle
- Collaborators
 - the other "things" with which this thing interacts
 - for us this means the things to which this thing is **coupled**
 - includes at least: all the things that this thing uses, and all the things that use this thing, at least directly

Class Name	
Responsibilities	Collaborators

Some books say to list just the things that this thing depends upon.

The Agile Alliance says:

- CRC cards (for Class, Responsibilities, Collaborators) are an activity bridging the worlds of role-playing games and object-oriented design.
- With the intent of rapidly sketching several different ideas for the design of some feature of an object-oriented systems, two or more team members write down on index cards the names of the most salient classes involved in the feature. The cards are then fleshed out with lists of the responsibilities of each class and the names of collaborators, i.e. other classes that they depend on to carry out their own responsibilities.
- The next step is to validate – or invalidate as the case may be – each design idea by playing out a plausible scenario of the computation, each developer taking on the role of one or more classes.

<https://www.agilealliance.org/glossary/crc-cards>

CRC Cards in Practice

- Typically used during early analysis, especially during team discussions.
 - Low-tech
 - 4x6 index cards
 - They aren't pretty.
 - They aren't something you ever want to show your customers or even your own upper management.
- Each card is a concrete symbol for a thing in the program during discussion
- Kind of like thinking on a whiteboard, but...
- Cards can be stacked, moved, etc. to illustrate proposed relationships
 - If you come out of a group meeting and your CRC cards aren't smudged, dog-eared, with lots of scratched-out bits, you probably weren't really trying.

<https://www.cs.odu.edu/~zeil/cs330/live/website/Slides/crc/page/crc.html>

The metaphor: Sketching the conspiracy



CRC Cards for us

- HW1 will ask you to use CRC Cards to document an *existing* design.
- You may not be able to identify all the classes that use your class. Don't worry too hard about that.
- We will also ask you to put one more thing on your CRC cards:
- State: the piece of state that an object of this class keeps.

CRC Card Template

Class Name:	
State:	
Responsibilities	Collaborators

This template is available in Canvas under Files/Week 02, as a spreadsheet for typing on and as a png that you can print out and write on by hand.

CRC Card for TemperatureSensor

```
// temperatures are measured in Celsius  
type Temperature = number
```

```
interface TemperatureSensor {  
    // return the current temperature  
    // at the sensor location  
    getTemperature () : Temperature  
}
```

CRC cards are supposed to be informal, so don't get hung up on emulating the exact words or the exact layout I've used here.

Class Name: TemperatureSensor (interface)	
State: none	
Responsibilities	Collaborators
establish interface for thermometers in the system	RefrigeratorThermometer
	OvenThermometer
	etc.
	TemperatureMonitor

TemperatureMonitor (1)

```
class TemperatureMonitor {  
    constructor(  
  
        // the sensors  
        private sensors: TemperatureSensor[],  
  
        // map from sensor to its location  
        private sensorLocationMap: SensorLocationMap,  
  
        private maxTemp: Temperature,  
        private minTemp: Temperature,  
        private alarm: IAlarm,  
    ) { }  
  
    // sensor in range?  
    private isSensorInRange (sensor: TemperatureSensor) : boolean {  
        const temp: Temperature = sensor.getTemperature()  
        return ((temp < this.minTemp) || (temp > this.maxTemp))  
    }  
}
```

Here's a slightly more elaborate
TemperatureMonitor

It monitors multiple sensors

And it knows where each sensor is

Better division into one method/one
job than our earlier version.

TemperatureMonitor (2)

```
// if the any of the sensors is out of range, sound the alarm
public checkSensors(sensor:TemperatureSensor): void {
    this.sensors.forEach(sensor => {
        if (!(this.isSensorInRange(sensor))) {
            this.soundAlarm(sensor)
        }
    })
}
```

```
private soundAlarm (sensor) {
    const location = this.sensorLocationMap.getLocation(sensor)
    this.alarm.soundAlarm(location)
}
```

```
}
```

CRC Card for TemperatureMonitor

Class Name: TemperatureMonitor	
State: sensors, maxTemp, minTemp, alarm	
Responsibilities	Collaborators
if any of the sensors is out of range, tell the alarm to sound at its location	TemperatureSensor
	SensorLocationMap
	IAlarm

IAlarm

```
// sound alarm for issue at the given location  
interface IAlarm { soundAlarm(location:Location): void }
```

Class Name: Ialarm (interface)	
State: none	
Responsibilities	Collaborators
Interface for classes that will sound an alarm	TemperatureMonitor
	all implementations of IAlarm

SensorLocationMap

```
class SensorLocationMap {  
    private locationMap : Map<TemperatureSensor,Location> = new Map ()  
  
    // get the location, if any. If none, throw error  
    public getLocation (sensor:TemperatureSensor) : Location {  
        if (this.locationMap.has(sensor)) {  
            return this.locationMap.get(sensor)  
        } else {  
            throw new Error (`sensor ${sensor} location unknown`)  
        }  
    }  
  
    // methods to add and remove sensors from the map...  
}
```

CRC Card for SensorLocationMap

Class Name: SensorLocationMap	
State: Map from Sensors to their Location	
Responsibilities	Collaborators
Maintain the map from Sensors to their Location	TemperatureMonitor

FireAlarm

- A hypothetical implementation of IAlarm

Class Name: FireAlarm	
State: socket for communicating with Fire Dept	
Responsibilities	Collaborators
when sounded, call the FireDept	IFireDept
when FireDept responds, turn off alarm	

Mapping the Conspiracy

Class Name: TemperatureSensor (interface)	
State: none	
Responsibilities	Collaborators
establish interface for thermometers in the system	RefrigeratorThermometer
	OvenThermometer
	etc.
	TemperatureMonitor

Class Name: TemperatureMonitor	
State: sensors, maxTemp, minTemp, alarm	
Responsibilities	Collaborators
if any of the sensors is out of range, tell the alarm to sound at its location	TemperatureSensor
	SensorLocationMap
	IAIarm

Class Name: SensorLocationMap	
State: Map from Sensors to their Location	
Responsibilities	Collaborators
Maintain the map from Sensors to their Location	TemperatureMonitor

Class Name: IAlarm (interface)	
State: none	
Responsibilities	Collaborators
Interface for classes that will sound an alarm	TemperatureMonitor
	all implementations of IAlarm

Class Name: FireAlarm	
State: socket for communicating with Fire Dept	
Responsibilities	Collaborators
when sounded, call the FireDept when FireDept responds, turn off alarm	IFireDept



Review: Learning Objectives for this Lesson

- You should now be able to:
 - Explain what it means to document a design
 - Describe the importance of having a shared vocabulary
 - for teams,
 - for communicating with management
 - for dealing with clients
 - Illustrate the basics of CRC cards

Next steps...

- In our next lesson, we'll talk about UML, a far more elaborate system for documenting designs