Agenda

- Educate the reader to the goals of Responsibility Driven Design presented by Wirfs-Brock and McKean
- Discuss how the principles shape the Process of design:
  - During project planning
  - Creation of software objects and collaborations
- Discuss the Refinement principles that improve your product
  - What Patterns are useful
  - How to effectively increase the “ilities” of your design

Background

- Invented by Rebecca Wirfs-Brock while working at Tektronix in 1990[2].
- She founded Wirfs-Brock Associates in 1997 with Alan McKean[2], and together co-authored the book “Object Design: Roles, Responsibilities, and Collaborations” in 2003.
- She has been a leader in the field of design and as recently as 2008 was the keynote speaker at OOPSLA 2008[2]

What is it?

- Responsibility-Driven Design is a shift in focus that force the designer to think about objects in their roles and responsibilities
- Places focus on each use case as a converstaion that takes into account what each user wants to experience as the system output.
  - This allows a designer to focus on requirements early, and not the implementation of each object.
  - The creation of objects and their collaboration is one that takes into account their role rather than features.
  - The process is iterative and meant to be refined over time

It’s a clarification process

- Find initial requirements, create system descriptions and system models
- Then generate more detailed descriptions and models of objects
- Generate detailed descriptions of object responsibilities and patterns of collaboration

- Remain fluid through the process as initial requirements and decisions will change based on the time spent tackling the problem

- The process has built in stops and self-checks to “reexamine, adjust, and align our work to a changing set of conditions.”[1]

Project Planning

- Start out conventionally with a concise statement of the project
  - A statement of purpose

    “We are going to create a mobile app to aid game hunters find duck migration patterns”
  - Generate an overview

    “Our app will contain storm patterns, wildlife reports and crowd sourcing data that allows hunters to strategically place themselves in areas for successful duck hunting”
  - Definition of Scope
    - “Our project will only focus on duck migration in the continental United States using the latest Android OS”
  - Benefits
    - “At the conclusion of our development this app provides users a strategic advantage over traditional methods of duck hunting, provides more detailed analysis than the current app market, and gives a source of advertisement income for our organization”

Project Planning (cont.)

Describe the following:

- How the software will be developed
    - *"...using the latest Android OS"*
- The values that are important to the project and the people involved
    - *"advertisement income"*
    - *"strategic advantage over traditional methods of duck hunting"*
- The people and their roles, the processes, and the expected outcomes
- The expected deliverables

- The authors stress being able to move from discovery, reflection and description of the project
    - Add details, find the ambiguous, and resolve requirement conflicts
- This stage of planning should not be object-oriented
    - *"We will create a family of Weather interfaces for each data source"*

Two Phases of Design

1. Initial Design
    - Define action-oriented objects from the domain
    - Assign these objects a set of responsibilities
    - Organize this initial object set into logical collaborations
2. Comprehensive solutions
    - Document design decisions
    - Create class definitions and diagrams
    - Utilize UML that documents abstractions and roles
    - Define control patterns of the initial model

Stakeholder Considerations

- The authors stress that the designers take their stakeholders in consideration during design
- Each participant has criterion against the final product
  - *"The hunter in our app wants weather reports updated at a high frequency"*
  - *"The systems team will want to see low latency in weather updates"*
  - *"Management wants to keep licensing and subscription costs for weather updates cheap"*

Analysis Descriptions

- Spend time reviewing requirements developing specifications early in the design process
- **WARNING** – Ill defined product specification could lead to costly rework activity downstream
- Understand the environment of your software application
  - What are the external requirements on communication.
  - What devices will the software interact with.
  - What are the data persistence specifications

Uncommon Language

- How do you communicate specifications?
  - **User** - “I want to see weather updates fast”
  - **System Admin** - “We need to keep the load on our weather aggregation server under 60%”
  - **Architect** - “I want weather updates to occur within 200 ms”
  - **Manager** - “We can't afford over 200k queries against that external weather update service per month”
  - **Weather APIs** - JSON, POST/GET, OS service calls

Know your Actors

- Grouped into three roles: Users, Administrators, External programs and devices
  - Always external to a program
  - Initiate, stimulate, and interact with the application
- Develop **Use Cases** to help understand your actors. Descriptions should be of the form:
  - Text narratives
  - Scenarios consisting of numbered steps
  - Conversations between the user and system.

Narrative Descriptions

- Write them in a meaningful way for the user.
- Use natural language and limit them to two paragraphs.

“ A hunter has options on how the mobile app receives weather updates in their area. By going to the options menu the hunter can select the service providers and a frequency in minutes that the weather is updated. Upon saving the options the app incorporates those changes immediately”

Scenario Descriptions

- Specific steps that the user must take to fulfill the use case
    1. The user clicks the icon to go to “Options”
    2. The app presents a screen consisting of weather service check boxes and a combo box of update frequency
    3. The user selects save and the Options form sends the form data to the weather controller for future updates.

Conversations

- Describe interactions between the software and the user as dialog
- Each conversation is made up of a set of actions/inputs and related software responses
- Depending on the situation these conversations can be highly detailed or written in a batch-oriented manner.
- Can provide a higher level of detail than narratives or scenarios accounting for decision points in the software.

Conversations (cont.)

User Actions	System Responsibilites
Show weather service providers	Display the name of each provider and a description of each service. If connectivity to a weather service is unavailable it shall be grayed out.
Optionally, choose an update frequency different from default	Update display and set form to new value
Optionally, make a change in weather service selection from default	Update display and set form to new value(s).
Optionally, provide the option to select “no weather service.”	Update display to gray out service providers and frequency. Set form to new value(s).
Indicate “Save” on form	If app shall provide weather service but no weather services are selected then alert user. Save form values and send to the weather update control in the app.

Designer Considerations

- Describe the information that the implementors will need to deliver a satisfactory software system.
  - Include specification of user supplied information and any default values
  - Provide constraints against critical system actions
  - Identify decision points in the software
  - Formulate key algorithms
  - Identify latency requirements
  - Provide references to related specification documentation

Conceptual Candidate Objects

- Wirfs-Brock and McKean stress concentrating on the “core” of a well designed application:
  - Key domain objects, concepts and processes
  - Objects implementing complex algorithms
  - Technical infrastructure
  - Objects managing application tasks
  - Custom user interface objects
- Expand on each object created and figure out if there are more candidate objects that can be gleaned from them.
- Find gaps is responsibilities and create objects from them

  “This app will need to display the most up-to-date map of the user's hunting area. We need objects that display a map, update that map regularly, provide layers of display, and allow for our specialized duck migration patterns to be displayed”

Candidates, Responsibilities, and Collaborators (CRC) Cards

- Whether candidate objects or roles, it is encouraged that designers utilize index cards to represent CRC cards
  - **Unlined side** - Write an informal description of each candidates purpose and role stereotypes
  - **Lined side** - Record the responsibilities of the candidate and information it must know to perform a set of specific actions. Define collaborators (other objects) that this candidate calls on.

CRC Card Example

DisplayLayer

Purpose: Transparent container for a set of datapoints and image objects to display

Patterns: Composite-component

Stereotypes: Structurer

DisplayLayer
Can control its transperency
Can be told to resize
Can be moved about the map using x and y pixel coordinates
Can be moved to different display layers

Object Stereotypes

How does your Object collaborate with the system?

- Information Holders - These objects typically do not collaborate but rather gather information at creation or over the course of the program. Consider whether or not the object holds facts of the system, How it is handled and persisted, and where it comes from.
- Structurers- These objects gather information or objects and structure them for use in the application. Consider where this information comes from and how objects the structerer knows about are accessed. The authors warn that trying to debate whether a structure is a composition or aggregation is futile as this stereotype could not fit either role completely
- Service-Providers - Responsibilities that require a specialized skill or role. Consider who has the information a service provider uses? Are services configurable? Does the application need different versions of the same service?

Object Stereotypes (cont.)

- Controllers – Control and direct the actions of others. They fall under two rules: To gather the information in order to make decisions and to call on others to act. Consider who knows the information that the controller needs? Who does it delegate responsibilities to? Who is responsible for the results of any of the controllers actions?
- Coordinators – These objects facilitate action among objects by passing along information. Consider the visibility of the coordinator in an object neighborhood?
- Interfacers – Act as bridges, usually between the I/O of a system or disjoint parts between two software entitites. They can be either internal or external interfacers based on whether they are a “storefront” to the activities of an internal system or exist to send requests to service-providers for non-object oriented APIs.

Where Do Responsibilities Come From?

- Identify system responsibilities from use cases
  - Use cases only tell what actions should occur for a system and not how those actions will be accomplished.
  - Take time to digest each user story and glean responsibilities from each.
- Plug in gaps with lower level responsibilities
  - Use cases do not explain error detection, timing, synchronization, aspects of control, or coordination.
  - Identify responsibilities as well as unresolved questions. Either work on what you know or work on those that have significant impact on your design.
- Identify those responsibilities that arise between candidates relationships
  - Use the natural language of the objects purpose when determining its responsibilities. In the case of a structurer it is almost always “managing” or “maintaining” other objects.
- Follow “what if....then...and how?” chains of reasoning
  - Don't focus on a specific task, but start with a high level goal.

    “The system shall maintain 100% uptime when tasked 10% over recommended capacity.”

Assessing an Objects Responsibilities

- When assessing an objects responsibilities in its object neighborhood the author's recommend the following tests
  - Does it stick to its purpose?
  - Are its responsibilities clearly stated?
  - Do its responsibilities match its role?
  - Is it of value to other objects in its neighborhood?

Those other “ilities” : Reliability

- Wirfs-Brock and McKean stress that designers spend time listing exceptions and errors, noting their handling and recovery
- Work on the differences between “trusted” and “untrusted” collaborations
- Identify which collaborations need to be reliable
  - By their specific task
  - Object neighborhood
  - Interfaces with an external system
  - Responses to exceptions of objects under its control

Those other "ilities" : Predictability

- Create software solutions in a consistent manner and don't vary solutions to each problem/algorithm.
- Factors that contribute to a consistent, comprehensible design:
  - Object are grouped in neighborhoods
  - There are few lines of communication between neighborhoods.
  - No one object knows, does, or controls too much
  - Objects perform according to their designated role
  - Apply the same solution to all variants when possible
  - Don't unnecessarily repeat the same patterns of collaboration throughout the design

Those other "ilities" : Flexibility

- Understand that not every object needs to be flexible and there are plenty of tradeoffs to take into consideration.
- One must take a look at the real problem and establish a vision that is cost effective and provides real benefits
- "Hot Spot Cards" handle what you don't know and identify where the architecture could flex or vary. Each "Hot Spot Card" should be filled out as early as the requirement gathering phase and is divided into three sections:
  - Top section is the name,
  - Middle summarizes the functionality that varies
  - Bottom lists two specific examples of the variation
- Design Patterns that increase Flexibility
  - Hide object interaction with a mediator
  - Vary an objects behavior with a strategy pattern
  - Use the Adapter pattern to make an object or system fit in the design

Developing The Solution

- Use your CRC cards to help find patterns
  - *"With so many ways to access each weather service API we need to pick a design pattern that simplifies our design"*
- Pursue a solution that explores all options, chooses simplicity, and works best for your application
  - *"We should explore the use of the Strategy pattern versus the Adapter pattern for each weather update API."*
  - *"Let's choose a solution that treats all weather services like our most popular service provider's API"*
  - *"The Strategy pattern in this case does more to complicate our design than make it easier to do work."*
  - *"Considering schedule and budget, let's not spend too much time making the Adapter pattern implementation elegant for service provider X. Follow best practice and get it done"*

Conclusion

- A Responsibility-Driven Design provides a very clear and Agile way to design large scale systems
- The use of this design pattern on small projects can be debated on it's applicability
- One could “over engineer” the design and end up with weak objects or weak collaborations[3]
- The focus on each object's roles and responsibilties can be very beneficial for junior developers

References

1. Wirfs-Brock, Rebecca, and Alan McKean. Object Design: Roles, Responsibilities, and Collaborations. Boston [Mass.: Addison-Wesley, 2003. Print.
2. "Wirfs-Brock Associates Responsibility-Driven Design." Wirfs-Brock Associates Responsibility-Driven Design. Wirfs-Brock Associates, 2010. Web. 17 Nov. 2012. < http://www.wirfs-brock.com/Design.html>.
3. "Responsibility-driven Design." Wikipedia. Wikimedia Foundation, 16 Nov. 2012. Web. 17 Nov. 2012. <http://en.wikipedia.org/wiki/Responsibility-driven_design>.