Act Responsibly!

Responsibility-Driven Design Concepts and Tools

Jennifer Wood

CSCI 5448 – Fall 2012

Act Responsibly!

Responsibility-Driven Design Concepts and Tools

- Executive Summary:
  - Structuring objects by their responsibilities was originally presented as an alternative to Data-Driven Design by Rebecca Wirfs-Brock and Brian Wilkerson in 1989
  - Responsibility-Driven Design promotes the analysis and design of software systems by focusing on what things the application and its objects must know, do or decide to fulfill its purpose
    - A detailed guide to the Responsibility-Driven Design process is presented in Rebecca Wirfs-Brock and Alan Kean's text Object Design: Roles, Responsibilities, and Collaborations (2003)
  - Maintaining a responsibility-driven design approach can result in a system with excellent encapsulation, loose coupling and strong cohesion by:
    - Keeping related attributes and methods within the same object or "neighborhood of objects"
    - Preventing knowledge of a data structure in an object from being revealed in the object's public methods

What are responsibilities?

- Let's start with the big concepts and definitions:
  - **Objects are things with responsibilities**
  - Responsibilities fall into one of three categories:
    - To know things
    - To do things
    - To decide things
  - A group of related responsibilities is a role
    - An object can implement more than one role if appropriate
    - Roles commonly encountered across many designs are called Role Stereotypes – more on these later
  - When two or more roles or objects work together it is called a collaboration

Early Responsibility-Driven Design

- Conceived by Rebecca Wirfs-Brock and Brian Wilkerson and first documented in the OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) Conference Proceedings of 1989 in "Object-Oriented Design: A Responsibility-Driven Approach"

- They contrasted Responsibility-Driven Design with Data-Driven Design
  - Data-Driven Design meant designing objects around the data they contained and was a common approach in early Object-Oriented programming
    - Wirfs-Brock and Wilkerson argued that this violated encapsulation because the internal structure of the (hidden) data was visible in the implementation of the object containing it
  - Responsibility-Driven Design tries to avoid revealing internal structure by ignoring implementation details until after the responsibilities of an object have been characterized
    - Forces encapsulation during the design phase rather than in implementation when it can be difficult to achieve

Benefits of Responsibility-Driven Design

- In a data-driven design, the structure of the data is often reflected in the objects designed around it – compromising encapsulation by providing a view into that structure
- Many design heuristics can be strengthened by applying Responsibility-Driven Design
  - By identifying as many responsibilities as possible within a design prior to implementation it can be easier to:
    - Find opportunities for polymorphism (you can see that there are several different objects that need to be processed in the same way)
    - Determine what and where abstractions add value to your design
    - Group related responsibilities into the same object or cluster of objects making your design more cohesive
    - Group information and methods performed on that information within the same object or cluster of objects, reducing coupling

Growth of Responsibility-Driven Design

- Wirfs-Brock and her colleagues took that original view "An object is something with responsibilities" and developed tools and techniques to use this insight throughout the design process

- Wirfs-Brock's Object Design: Roles, Responsibilities, and Collaborations (2003) provides a guide to taking a responsibility-driven viewpoint through the analysis and design process
  - The goal in Responsibility-Driven Design (RDD) is to identify as many responsibilities and roles present in your system as possible BEFORE you begin to implement them into objects and classes
  - It's easier to change what objects you have or what responsibilities they have BEFORE you have thousands of lines of code written

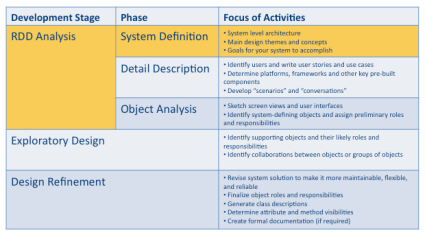
Responsibility-Driven Analysis & Design Stages

- Wirfs-Brock identifies three main analysis and design phases in the development of a project from the responsibility-driven perspective

Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop “scenarios” and “conversations”
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Analysis and Design from the RDD Perspective

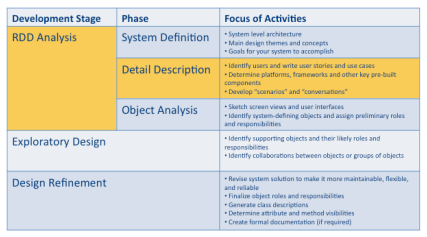
- "Responsibility-Driven Design is a clarification process."
  - Wirfs-Brock and Kean, Object Design: Roles, Responsibilities, and Collaborations
- Each phase is more detail-oriented than the preceding one
- Although presented as a linear progression, in reality you may find you have to double back to gain clarification in some aspects of your design at the higher levels before you can implement your objects
- The following slides will discuss each phase of activity including
  - Team discussions and decisions occurring in that stage
  - Results for the stage
  - Any additional tools or concepts needed to complete the stage



Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

RDD Analysis Stage: System Definition Phase

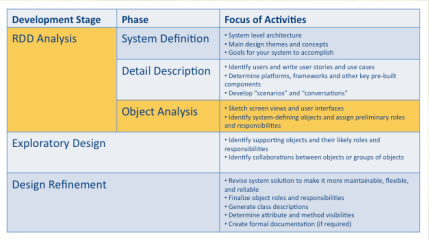
- Discussions your team will have at this stage:
  - Define the main goals of your system
    - What is it trying to accomplish?
    - Identify what system-level responsibilities must be present in your final system
  - Compare what you think the system should do with what your team members envision
    - Do we mean the same things? Does our team have a common vocabulary for domain level concepts and ideas?
    - Are you using the same identifiers for potential objects and responsibilities?
    - Are you sure you mean the same thing when you are using a term or throwing around a concept?
  - Define the boundaries of your system
    - The system does *this*, but it doesn't do *that*…
  - Diagram what your team envisions the architecture of the system to look like at its highest levels
- Stage Results:
  - System level discussions
  - Sketches of desired system architecture
  - Understanding of constraints, technical limits, schedule, and budget
  - List of potential user types and their perspectives



Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

RDD Analysis Stage: Detail Description Phase

- Discussions your team will have at this stage:
  - Where will the system be developed and implemented (i.e. platforms, programming languages, existing frameworks, etc.)?
  - What do the potential users of this system want it to accomplish?
  - What is the expected work flow through the system?
- Stage Results:
  - Define development environment
  - User stories
    - Simple narratives of how a user expects to use the system and the task(s) they seek to accomplish
  - Scenarios
    - Use case for a specific path through a task or user story
  - Conversations
    - Descriptions of how the user and the system will interact with each other over time
  - Activity Diagrams



Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

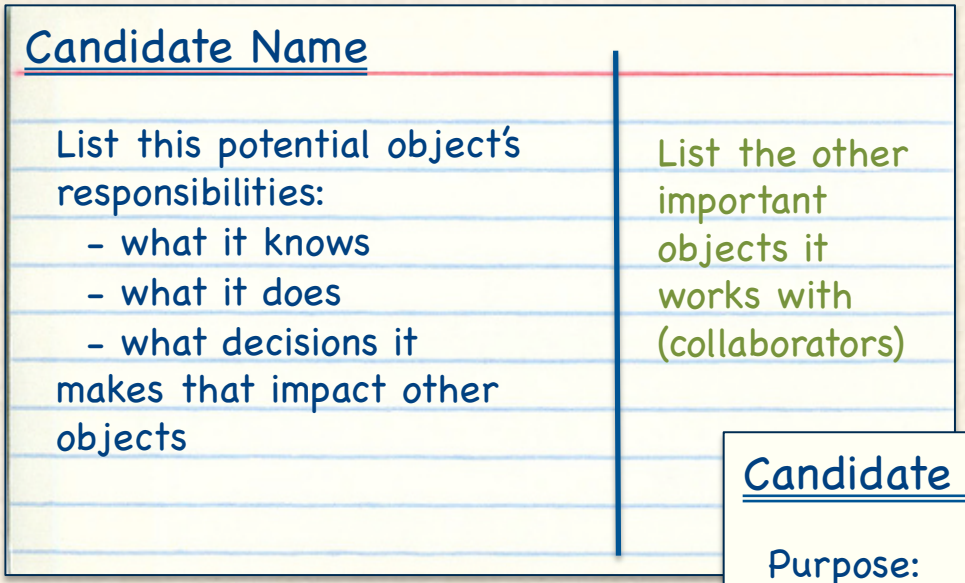
RDD Analysis Stage:
Object Analysis Phase

- Discussions your team will have at this stage:
  - What will the user interfaces look like? What will the user(s) see?
  - What domain-associated Candidates (roughed-in object assignments) can we define?
    - Example: If we're building a jukebox we probably need a Player
  - What major responsibilities will these Candidates have?
    - What will they know?
    - What will they do?
    - What decisions will they make that impact other objects?
- Stage Results:
  - User interface mock-ups
  - CRC Cards for the potential objects (Candidates) we've identified for our system
    - Wait! What's a CRC Card? Let's talk about them for a bit…

Terminology Detour: CRC Cards

- In Responsibility-Driven Design, CRC stands for Candidates, Responsibilities, and Collaborators
- CRC Cards were originated by Kent Beck and Ward Cunningham in 1989 at the same time that Wirfs-Brock was developing her ideas on Responsibility-Driven Design
  - Originally, the first 'C' stood for Class and the cards were used as a tool when teaching programmers to work from an object-oriented perspective instead of a procedural one
- CRC Cards in RDD are index cards used to document potential objects (called Candidates) that you believe your final design will include and what responsibilities they will have along with what other objects they will likely work with (collaborators)
  - Allows you to design on paper and rearrange your model before committing by writing code
    - Keeps your options and your mind open longer, allowing for a more flexible approach
    - Cards and the responsibilities they carry can be easily arranged and rearranged – not true for code

Terminology Detour:
CRC Card Layout



Front

Back

Candidate Name (again)

Purpose:
Write a short description about what the purpose of this object is.

Role: Name a set of responsibilities that has a shared meaning in your system
Pattern: List this object's role in any Design Patterns
Stereotype: List any role stereotypes that apply

What's a role stereotype? Go to the next slide...

Terminology Detour:
Role Stereotypes – Slide 1

- Role stereotypes act like Design Patterns in thinking and talking about candidates in your design
  - You can say, "I think we need this to be a Service Provider," and the rest of your team will be on the same page
- Casting candidates in your design into applicable role stereotypes can help you detail their responsibilities
  - Each role stereotype has a set of responsibilities commonly required to fulfill its function
- Not all objects in your system will fit into one of these stereotypes
- Some objects in your design may hold the responsibilities of more than one stereotype

Terminology Detour:
Role Stereotypes - Slide 2

- Six common role stereotypes (Wirfs-Brock, February 1992 SmallTalk Report):
  - Information Holders
  - Structurers
  - Service-Providers
  - Controllers
  - Coordinators
  - Interfacers
- If you find that these role stereotypes don't help in your domain, try to identify your own and use those instead
- Common usage constructs like these can be helpful when working on a team, so we'll go into some more detail…

Terminology Detour:
Role Stereotypes - Slide 3

- Information Holders
  - Primary Responsibility: To hold and maintain its information
  - Related responsibilities (things the Information Holder or one of its collaborators may need to do)
    - Gathering or creating the information
      - Does it keeps its own copy or ask for its information again from its source?
    - Deriving its held information from other data (calculations, conversions, or other processing)
    - Handling any need for information persistence
    - Updating the information and coordinating any updates

Terminology Detour:
Role Stereotypes - Slide 4

- Structurers
  - Primary Responsibility: To structure and organize objects that the Structurer may or may not own
    - Think about if this work is visible or hidden – can the organized objects be seen/used by other objects?
    - Does the structurer need to know about the objects it organizes? And vice versa?
  - Related responsibilities (things the Structurer or one of its collaborators may need to do)
    - Accessing or creating the objects it organizes
    - Processing the objects
    - Handling any need for persistence of the Structurer or the things it organizes
    - Answering requests for information about the objects it structures

Terminology Detour:
Role Stereotypes - Slide 5

- Service Providers
  - Primary Responsibility: To perform specialized calculations or tasks on behalf of its collaborators
    - If a task or calculation may change over a system's life cycle, it may be better to encapsulate the task or calculation into a Service Provider
  - Related responsibilities (things the Service Provider or one of its collaborators may need to do)
    - Accessing the information to be used
      - Does it ask or is it told?
    - Configuring the Service Provider for the current task, if required
    - Handling related, but different service requests
      - Is there a family of Service Providers or is it all performed in one?

Terminology Detour:
Role Stereotypes – Slide 6

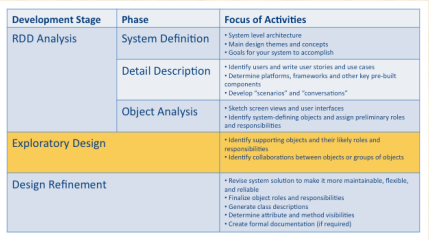
- Controller
  - Primary Responsibility: To make decisions and command other objects into action
  - Related responsibilities (things the Controller or one of its collaborators may need to do)
    - Gathering the information it uses to make its decisions
    - Communicating its commands
    - Monitoring in-process events

Terminology Detour:
Role Stereotypes – Slide 7

- Coordinators
  - Primary Responsibility: To pass information and to request action from other objects
    - Manages the connections between multiple objects
  - Related responsibilities (things the Coordinators or one of its collaborators may need to do or help with)
    - Passing messages to request action or communicate a change in state
    - Delegating tasks

End of Terminology Detour:
Role Stereotypes – Slide 8

- Interfacers
  - Primary Responsibility: To allow communication between different components of a system
    - User Interfacers
      - Pass user requests or display information
      - Typically collaborate with event handlers and the objects that update the information they display
    - Internal Interfacers
      - Provide a Façade into their neighborhood of objects for other parts of the system to communicate through
      - Delegates the requests made to it to the appropriate object it collaborates with
      - It may provide an Adapter by translating requests made by external objects into formats usable by its internal collaboraters
    - External Interfacers
      - Similar to an Internal Interfacer, but they work with collaborators outside of the system
      - May have to manage its connection to the external systems it assists

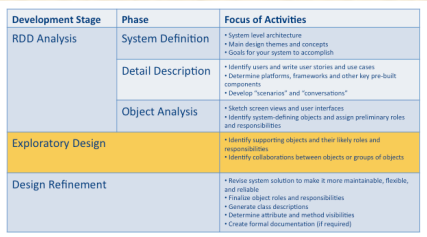


Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Continuing our Tour of RDD:

Exploratory Design Phase

- Now that you know about CRC Cards and Role Stereotypes, let's move onto our next design phase: Exploratory Design
- As you move out of the analysis phase, you'll have a handful of object Candidates noted on CRC cards
  - These objects will help you explore the rest of your design
  - Ask what supporting objects will your domain-level Candidates need to do their jobs?
- Discussions your team will have at this stage:
  - Finding Objects:
    - What execution-related objects will the domain-level objects identified previously need?
      - What responsibilities will these objects have?
      - How will the work flow through the system?
    - What other "Core" objects can we find? From Wirfs-Brock and Kean, Object Design: Roles, Responsibilities and Collaborations, they might be:
      - Key domain objects, concepts, and processes
      - Objects implementing complex algorithms
      - Technical infrastructure
      - Objects managing application tasks
      - Custom user interface objects



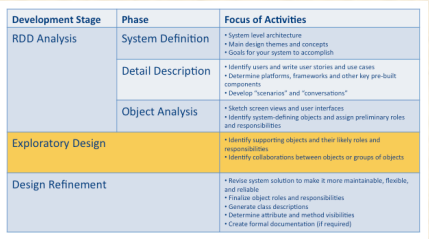
Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Continuing our Tour of RDD:

Exploratory Design Phase

Finding Responsibilities

- Stage discussions, continued:
  - Finding responsibilities:
    - Wirfs-Brock and Kean in Object Design: Roles, Responsibilities and Collaborations suggest that responsibilities can be uncovered by:
      - Examining system functionality called upon either explicitly or implicitly stated in use cases and user stories
      - Looking for objects tied to important events in the system
      - Looking for important events in the life of an object
        - "When an object is created and when it is no longer used are common places to find responsibilities for gracefully entering and leaving the scene." (Wirfs-Brock, '03)
      - Providing identified stereotype players with their stereotypical responsibilities
      - Identifying responsibilities needed to cover edge cases in your design
      - Creating the private responsibilities that will support the public ones
      - Filling gaps between the subsystems, objects or responsibilities identified in earlier sweeps



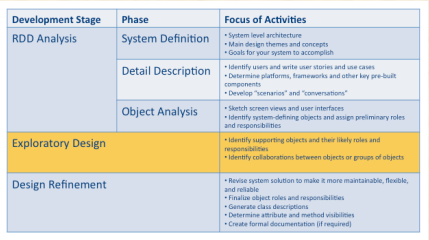
Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Continuing our Tour of RDD:

Exploratory Design Phase

Assigning Responsibilities

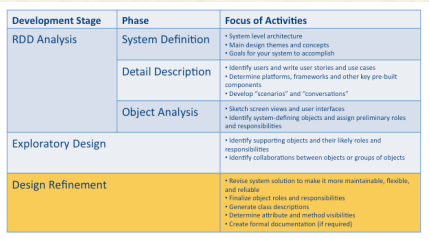
- Stage discussions, continued:
  - Assigning responsibilities:
    - Wirfs-Brock and Kean in Object Design: Roles, Responsibilities and Collaborations provide some suggestions for dividing up responsibilities in your system:
      - If the object is responsible for holding the information, make it responsible for maintaining and performing operations with that information
      - Try to give objects as much intelligence and responsibility as they can handle, but avoid centralizing all your decision-making in one place
      - Don't duplicate information: make one object responsible for keeping a piece of information and allow others to make copies
      - The set of responsibilities an object hold should fit into a role – they should be related and coherent
      - Keep objects and their responsibilities at the same level (don't give a high-level task to a low-level object)
      - Each object should take on only as much responsibility as it must to complete its job



Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Continuing our Tour of RDD:
Exploratory Design Phase

- Stage discussions, continued:
  - What obvious collaborations (objects working together) can we identify?
  - What subsystems can we identify?
  - Are there any concepts/implementations we want to use, but aren't sure will work?
    - Build working prototypes of these objects or collaborations
  - Do we have any responsibilities that we can't assign to an object yet?
    - Keep them written on a stack of Post-Its so you can try different assignment combinations as you narrow down your design
    - Keep trying to assign them to likely Candidates as you add to your system
- Stage Results:
  - More CRC Cards for the additional Candidates we've identified for our system
  - Unassigned responsibilities on Post-Its
  - Collaboration model for your system
  - Sequence diagrams
  - Working prototypes for important concepts or potential problem areas in your system solution

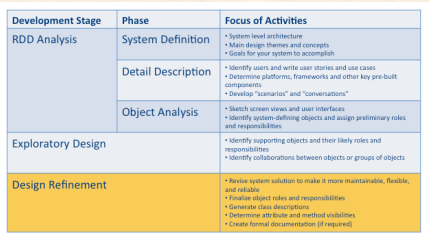


Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Finishing our Responsibility-Driven Design:

Design Refinement Phase

- At this point in your design process:
  - Majority of the Candidates defined for your system and documented on CRC Cards
  - Collaborations and subsystems built around core Candidates of your system established
- Now you iterate on your model until you have the arrangement you can implement with confidence
- May have to return to higher levels in the Analysis & Design sequence briefly if you uncover an area of your system not previously explored
- Discussions your team will have at this stage:
  - Are there design trade-offs we've had to make? Are we confident in our choices
  - How will control be distributed in this system?
    - Centralized
      - Decision-making and intelligence of system concentrated in a small number of objects, maybe even just one
    - Delegated
      - System intelligence divided among a moderate number of objects
      - Considered the most desirable control style for most systems designed with RDD
    - Distributed
      - system intelligence and decision making capabilities spread throughout the whole system



Development Stage	Phase	Focus of Activities
RDD Analysis	System Definition	• System level architecture • Main design themes and concepts • Goals for your system to accomplish
	Detail Description	• Identify users and write user stories and use cases • Determine platforms, frameworks and other key pre-built components • Develop "scenarios" and "conversations"
	Object Analysis	• Sketch screen views and user interfaces • Identify system-defining objects and assign preliminary roles and responsibilities
Exploratory Design		• Identify supporting objects and their likely roles and responsibilities • Identify collaborations between objects or groups of objects
Design Refinement		• Revise system solution to make it more maintainable, flexible, and reliable • Finalize object roles and responsibilities • Generate class descriptions • Determine attribute and method visibilities • Create formal documentation (if required)

Finishing our Responsibility-Driven Design:

Design Refinement Phase

- Possible stage discussions, continued:
  - Which attributes and methods will be private? Which will be public?
  - Can we rearrange our Candidates, responsibilities or collaborations to make the system more consistent? Easier to maintain? Flexible?
    - What interfaces/protocols can we implement?
    - What additional abstractions can we create?
    - What design patterns can we implement?
    - Some factors that can contribute to a consistent, comprehensible design (Wirfs-Brock, '03):
      - Objects are grouped in neighborhoods
      - There are few lines of communication [messages passed] between neighborhoods (logical groupings of collaborators, possibly as subsystems)
      - No one object knows, does or controls too much
      - Objects perform according to their designated role
      - When one solution is designed, variants will be applied to other parts that are similar
      - There are few patterns of collaboration that repeat throughout the design
  - And at last, are we ready to finalize our design and go implement it?
- Stage Results:
  - Control style
    - Agreed to patterns for assigning decision making responsibilities throughout the system
  - Finalized CRC Cards
    - Blueprints for implementation of actual objects in system
      - Reflect final control style in the assignments of decisions-making responsibilities
      - Include decision results of public/private responsibilities
      - Document any design patterns or role stereotypes we've identified for the object
  - Any other formal documentation the project might require before beginning implementation

Yikes! That was a lot of A & D!

- Even so, this tour through Responsibility-Driven Design was a very brief summary of the design advice and tips Wirfs-Brock and Kean have compiled in their book Object Design: Roles, Responsibilities, and Collaborations
- The process, when written out in a linear set of steps, can appear cumbersome, but in practice many of these steps will occur simultaneously
  - Working through the design, you will cycle between higher and lower levels of design quickly and often

Using Responsibility-Driven Design

- Focus on describing your system by its responsibilities and the objects that will hold them
  - In your design process, expand out from a handful of high level/important objects and/or responsibilities to discover the rest of your model
  - You will not find every responsibility before you begin implementation
    - That's okay!
    - Finding and assigning responsibilities is an iterative process
  - Don't be afraid to explode roles or objects and reassign their responsibilities to others
    - Keep trying new arrangements and combinations until you see the system emerge that meets your requirements and design sensibilities
- The ideas and concepts of Responsibility-Driven Design can be integrated into common design life cycles like Agile
  - RDD provides a unique viewpoint into breaking down user stories into implementable objects
- Note that you may not need to use all of the tools in RDD in every design, but they can be helpful in your analysis and design process

References

- For more information on Responsibility-Driven Design
  - Wirfs-Brock, Rebecca and Alan McKean. Object Design: Roles, Responsibilities, and Collaborations. Addison-Wesley, 2003.
    - Includes orders of magnitude more detail on each aspect of RDD than could be included in this presentation
  - Wirfs-Brock, Rebecca and Brian Wilkerson. (1989) "Object-Oriented Design: A Responsibility Driven Approach," OOPSLA 1989 Proceedings, October, 1989, pp 71 -75.
    - This paper is where Responsibility-Driven Design was first put forth
  - Wirfs-Brock, Rebecca. "What Drives Design?" Presented at OOPSLA 2008 Conference, January, 2009. http://http://www.infoq.com/presentations/What-Drives-Design-Rebecca-Wirfs-Brock
    - Talk includes her recent views on the evolution of RDD and compares it with other design techniques like Test-Driven Design and Event-Driven Design
  - Wirfs-Brock Associates. http://http://www.wirfs-brock.com/index.html
    - Provides links to a number of Wirfs-Brock's articles on design as well as to her blog: The Responsible Designer