# Requirements Modeling

CSC 326

© NC State Software Engineering Faculty



1

## Upcoming Lectures/Labs/Deadlines

| Date | Thing | Notes |
|---|---|---|
| ~~*M Sep 21*~~ | ~~*Lecture*~~ | ~~*Today – Architecture + Requirements Engineering*~~ |
| ~~W Sep 23~~ | ~~Lecture~~ | ~~Project Workday – no class meeting – work on Milestone 4!~~ |
| ~~W Sep 23~~ | ~~OBP M4 T~~ | ~~M4 Team Deadline @ 11:45 pm~~ |
| ~~H Sep 24~~ | ~~Lab~~ | ~~Grade M4 Team & Start M5 (Frontend)~~ |
| M Sep 28 | OBP M4 I | M4 Individual Deadline @ 5:00 pm + Retro @ 11:45 pm |
| M Sep 28 | Lecture | Requirements Modeling (+ User Stories?) |
| W Sep 30 | Lecture | Project Workday |
| H/F Oct 2/3 | Lab | Grade M4 Individual + Continue M5 (Frontend) |
| M Oct 5 | Lecture | Project Workday – no class meeting – work on Milestone 5! |
| M Oct 5 | OBP M5 T | M5 Team Deadline @ 11:45 pm |
| W Oct 7 | Lecture | Presentations + Software Process |
| W Oct 7 | OBP M5 I | M5 Individual Deadline @ 5:00 pm + Retro @ 11:45 pm |
| H Oct 8 | Lab | M5 Team Grading + M5 Individual Grading |
| F Oct 9 | OBP M6 T | M6 Team Deadline @ 11:45 pm |
| S Oct 10 | OBP M6 I | M6 Individual Deadline @ 5:00 pm + Retro @ 11:45 pm |
| M Oct 12 | Lecture | Midterm Exam |
| T Oct 13 | OBP M7 | M7 Reflection Deadline @ 11:45 pm |



2

# Plan early for M6: Onboarding Project Deadline!

| | | | |
|---|---|---|---|
| Milestone 4: Backend Implementation + Testing | H Sep 17 | W Sep 23 @ 11:45 pm | M Sep 28 @ 5:00 pm |
| Milestone 5: Frontend Implementation + Testing | H Sep 24 | M Oct 5 @ 11:45 pm | W Oct 7 @ 5:00 pm |
| Milestone 6: Onboarding Project Deadline | H October 8 | F Oct 9 @ 11:45 pm | S Oct 10 @ 5:00 pm |
| Milestone 7: Onboarding Project Reflection | S Oct 10 | N/A | T Oct 13 @ 11:40 pm |

You should be 99% done with the project by the end of Milestone 5. Milestone 6 is for last-minute bug fixes.

There is a 24-hour late window for Milestone 6 Team and Individual deadlines.
There will be a penalty of no more than 10% of the Milestone 6 grade (90 points).

3

3

# Requirements Engineering




Illustration: The overall requirements engineering process

img

4

4

# Recap: Requirements Engineering

- **Analysis**: studying user needs to generate system definition that **users understand**

**Previous class**

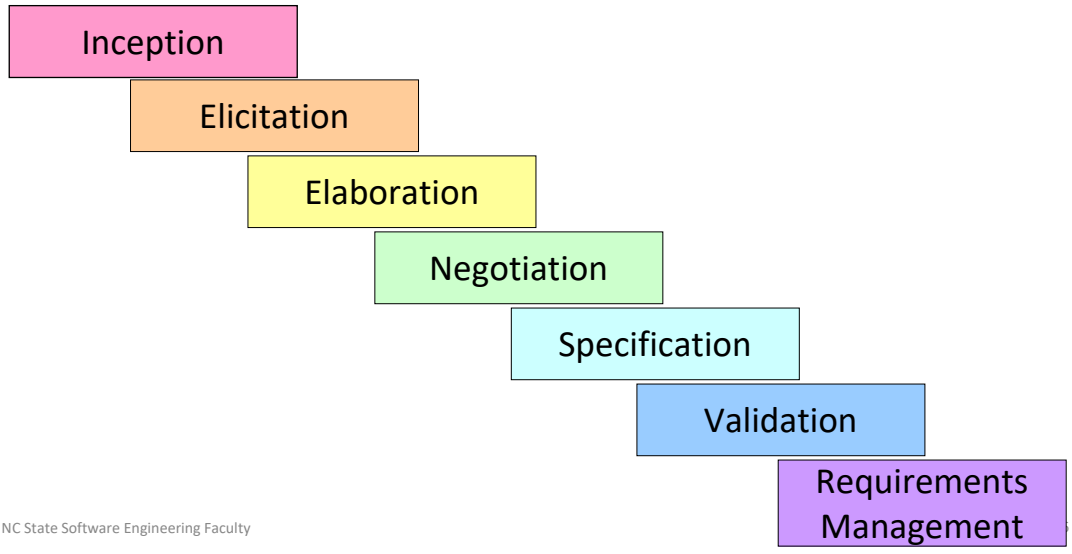
- **Modeling**: translating requirements the user understands to a form that software **engineers understand**



© NC State Software Engineering Faculty


5

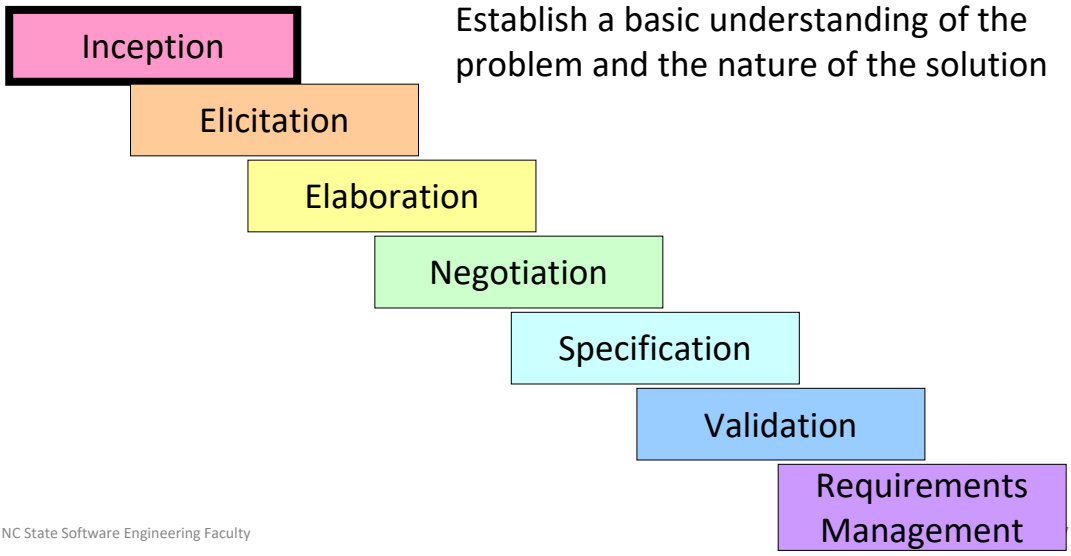
# Recap: Requirements Engineering Tasks




© NC State Software Engineering Faculty

6

# Recap: Requirements Engineering Tasks

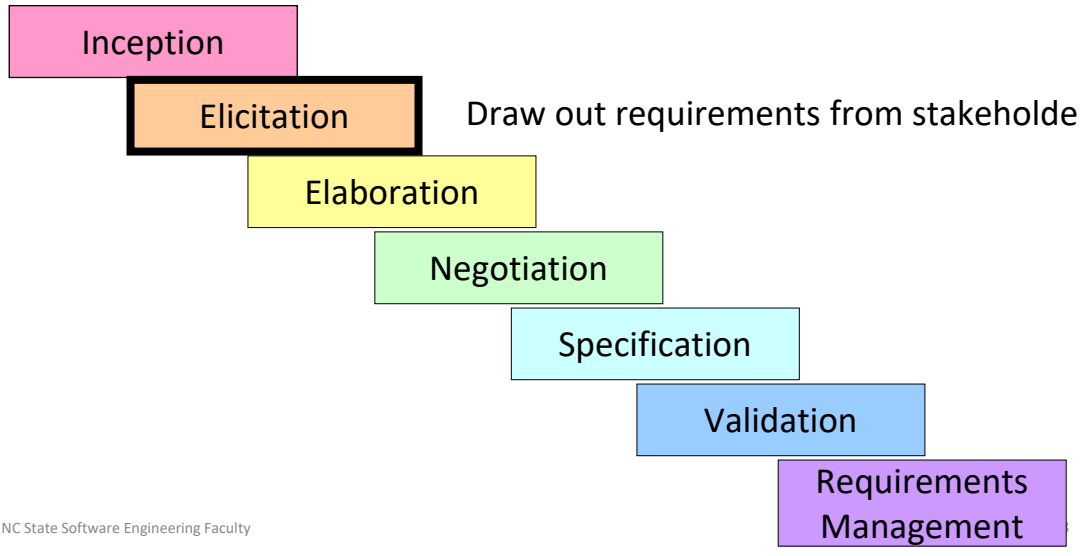



Establish a basic understanding of the problem and the nature of the solution

© NC State Software Engineering Faculty

7

# Recap: Requirements Engineering Tasks

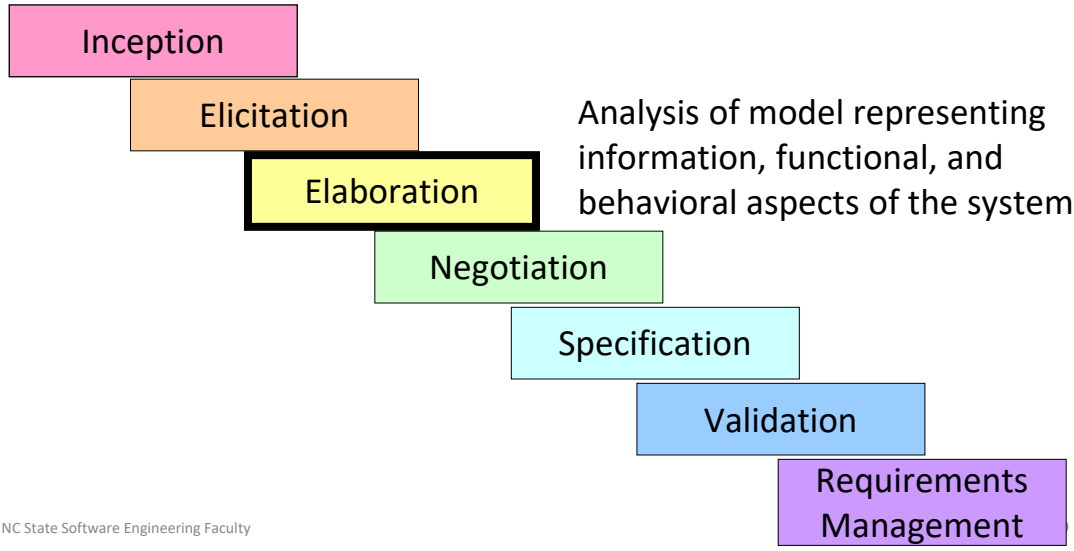



Draw out requirements from stakeholder

© NC State Software Engineering Faculty

8

# Recap: Requirements Engineering Tasks

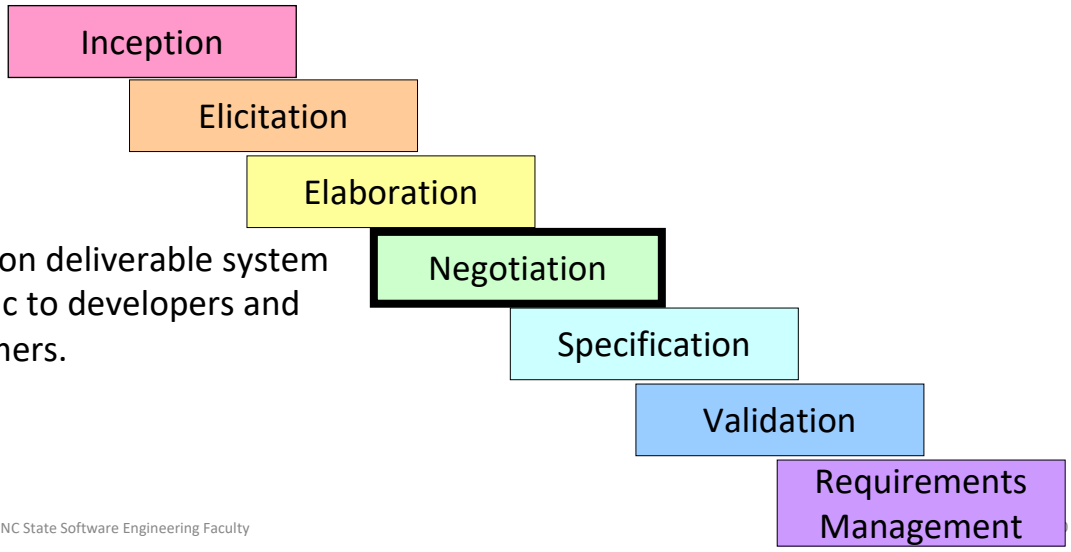



Analysis of model representing information, functional, and behavioral aspects of the system.

© NC State Software Engineering Faculty

9

# Recap: Requirements Engineering Tasks

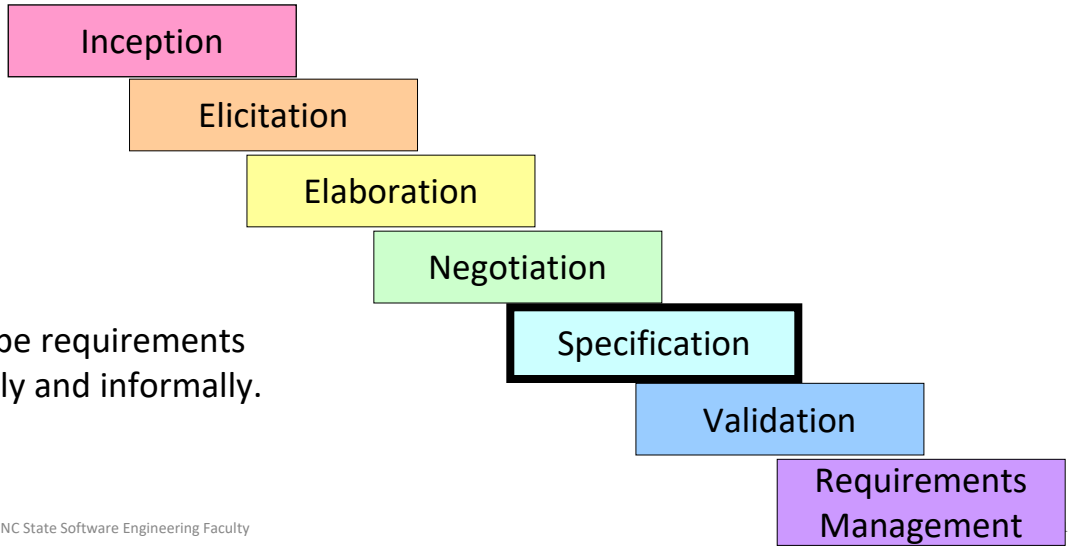



Agree on deliverable system realistic to developers and customers.

© NC State Software Engineering Faculty

10

# Recap: Requirements Engineering Tasks

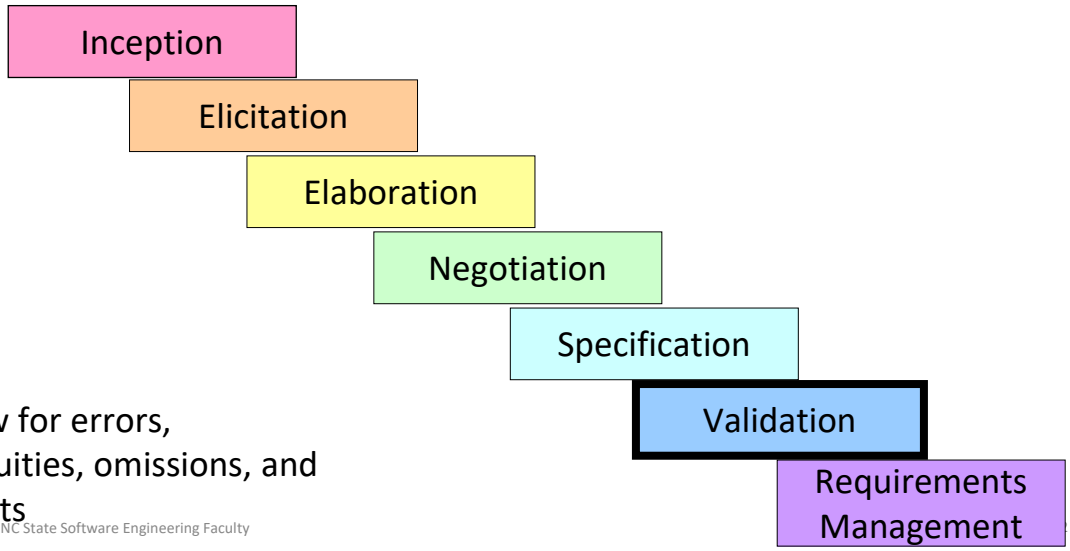



Describe requirements formally and informally.

© NC State Software Engineering Faculty

11

# Recap: Requirements Engineering Tasks

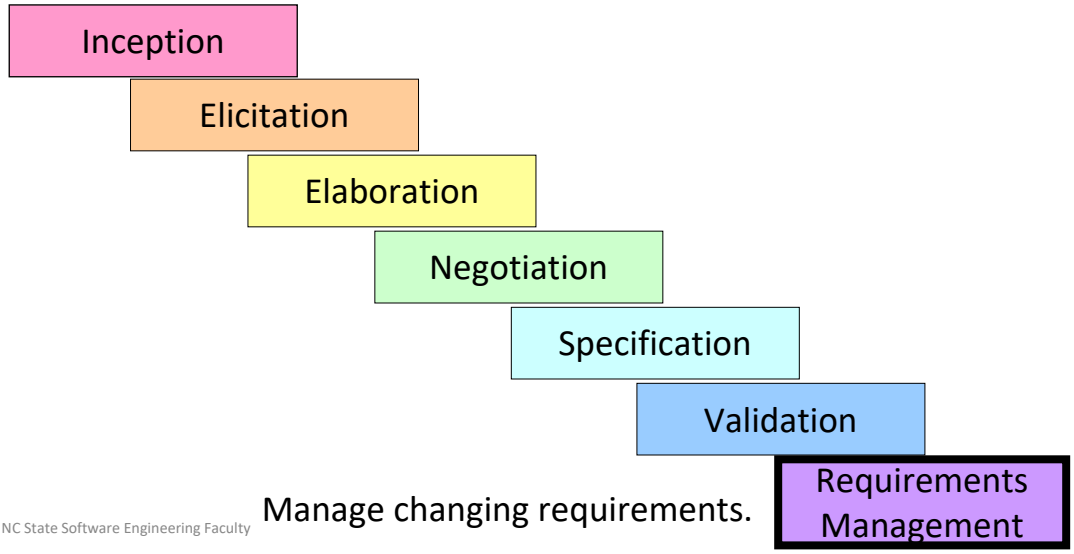



Review for errors, ambiguities, omissions, and conflicts

© NC State Software Engineering Faculty

12

# Recap: Requirements Engineering Tasks




Manage changing requirements.

© NC State Software Engineering Faculty

13

# Requirements Engineering

- **Analysis**: studying user needs to generate system definition that **users understand**

**Previous class**

**Input**

- **Modeling**: translating [requirements the user understands] to [a form that software **engineers understand**]

**Output**



© NC State Software Engineering Faculty



14

# Requirement Types





## Types of Requirements

- **Business Requirements:** provide vision and scope of goals
  - Improve customer satisfaction for the business's customers
  - Increase revenues
  - Reduce costs
  - Improve operations (business process, technology, or infrastructure)
- **User Requirements:** description of an elicited user goal, need, or want
  - Features, user stories, scenarios
- **Product Requirements:** influences or constrains the product's functionality or attributes
- **Process Requirements:** constrain or influence how a product is developed

# Types of Product Requirements

**Product Requirements**: influences or constrains the product's functionality or attributes

- **Functional Requirements**: requirements that specify a function that a system or system component must be able to perform
  - Describe in terms of input/output or user actions and system response
  - **Example**: The system shall accept a new recipe composed of zero to many units of any ingredients.
- **Non-functional Requirements (Quality Attribute)**: how well a product must do its job.
  - User visible aspects of the functional parts of the system
  - Include properties such as scale, performance, reliability, usability, privacy, security, and availability
  - **Example**: The response time must be less than 1 second.

17

17

# Process Requirements (Constraints)

- **Process Requirements (Constraints)**: constrain or influence how a product is developed, and they influence the development process.
  - Not user-visible
  - Imposed by the client or stakeholders
  - Restricts the implementation of the system
  - Restricts the development process
  - **Example:** The backend must be implemented in Java
  - **Example:** Unit tests must be written in Junit
  - **Example**: Tests must be selected to satisfy Modified Condition/Decision Coverage (MC/DC)

18

18

# Ways to Model Requirements

- Traditional/Plan-Driven (system-centric view)
  - "The system shall…"
- Use case-based (user-centric view)
  - Include a use case diagram to visualize the relationship between use cases
- User story (user-centric view)
  - Agile requirements
  - Married with an acceptance test to supply the details

©2019 NC State Software Engineering Faculty

19

# Functional, Non-Functional, Constraints?

R1: "The application must be developed using Python programming language."

R2: "The system must be available 99.99% of the time, excluding scheduled maintenance."

R3: "The website must allow users to create an account and log in using their email and password."

20

20

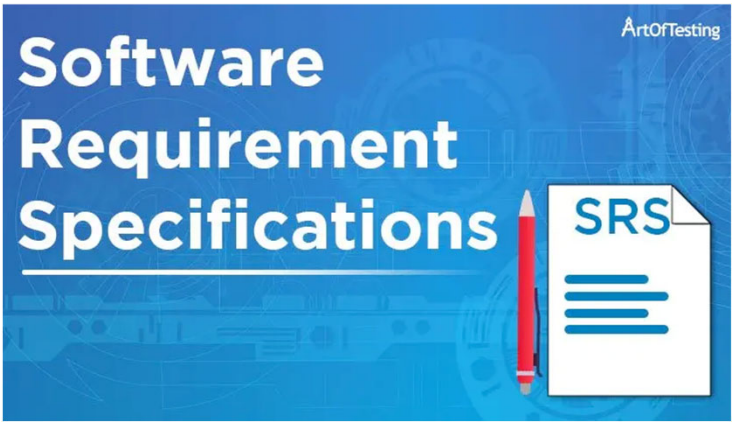
# Exercise 14.01

Functional, Non-Functional, Constraints?





# Software Requirement Specification (SRS)




A software requirements specification (SRS) is **a description of a software system to be developed.**

© NC State Software Engineering Faculty

## Guidelines for Writing SRS

- Find a standard format and use it for all requirements.
- Use language in a consistent way.
- Use text highlighting to identify key parts of the requirements.
- Avoid the use of jargon.

23

## IEEE Standard for Writing SRS

- Defines a **generic structure for a requirements document** that must be instantiated for each specific system.
- **1. Introduction**
  - Glossary
- **2. Overall description**
  - User roles
- **3. Specific requirements** → Main body of the document
  - Functional Requirements
  - Non-functional Requirements
  - Constraints
- **4. Appendices**
- **Index**

} Supporting information

24

# 1. Introduction

- Purpose (business case)
- Scope
  - Identify the problem (application) domain
- Definitions, acronyms, and abbreviations
- Reference documents
- Overview
  - Describe the structure of the SRS (Software Requirements Specification)

25

# 2. Overall Description

- Presents a high-level overview of the product and the environment in which it will be used, the anticipated product users, and known constraints, assumptions, and dependencies.
- Product perspective
  - Describe all external interfaces (system, hardware, software, user, communication interfaces, etc.)
- Product functions
  - Summary of major functions that the software will perform
- User characteristics & roles
  - General characteristics of the intended users
- Constraints
  - General description of items that will limit the developer's options (these may also have their own section in the Specific Requirements portion of the document)
- Assumptions and dependencies

26

# 3. Specific Requirements

- The main body of the document
- Contains all the software requirements in detail
  - External interface
  - **Functional requirements**
  - **Non-functional requirements (Quality Attributes)**
    - Performance, usability, accessibility, security, privacy, reliability
  - **Constraints**
    - Product and process constraints
  - Software system attributes
  - Other requirements

27

# 4. Appendices

- Appendices may include additional information about the system
  - Interacting systems & their interfaces
  - Data formats
  - Laws, regulations, etc.

- Indices may be helpful for very long requirements documents

© NC State Software Engineering Faculty


28

## Skim SRS, then activity 14.02

The SRS (source from Dr. Lawrence Chung from UT Dallas)

Many companies use variations on documents like this - they call a **Product Requirements Document (PRD)** at Google, Microsoft, Amazon

We're using the IEEE standard in this lecture. The goals are all the same.

29

29

## Three Forms Of Requirements

Traditional/Plan-Driven.
Use Case.
User Story.

© NC State Software Engineering Faculty

30

30

# Ways to Model Requirements

- Traditional/Plan-Driven (system-centric view)
  - "The system shall…"
- Use case-based (user-centric view)
  - Include a use case diagram to visualize the relationship between use cases
- User story (user-centric view)
  - Agile requirements
  - Married with an acceptance test to supply the details

Which form do you think was used in the SRS we just examined?

©2021 NCSU Software Engineering Faculty

31

## Spectrum of Appropriate Requirements Processes and Tools

*Requirements Process*

*more formal* ← → *less formal*



### Traditional Requirements Use Cases

- Stable
- Larger
- Safety Critical

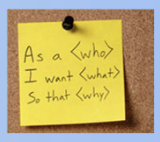


Examples:
- Surgery Robots
- Health Care
- Nuclear Power Plant

Img: istock

- Plan-driven, well-documented

### User Stories




- Changing
- Smaller
- Fast-to-market
  - Examples:
    - Games
    - Most Apps
    - Social media

TikTok

- Agile, Scrum, XP

32

# Traditional/Plan-Driven Requirements

© NC State Software Engineering Faculty 



## Traditional Requirements: Terminology

- **Shall** (== *is required to*): used to indicate **mandatory requirements** strictly to be followed in order to conform to the standard and from which no deviation is permitted (must or will is deprecated)

> *3.4.9* ***Detailed invoice for customer.***
>
> The system shall display detailed invoice for current order once it is confirmed.
>
> The system shall optionally allow user to print the invoice.
>
> *3.4.10* ***Provide shopping cart facility.***
>
> The system shall provide shopping cart during online purchase.
>
> The system shall allow user to add/remove products in the shopping cart.

https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf **See Section 10.2.2**

© NC State Software Engineering Faculty 

# Traditional Requirements: Terminology

- **Should** (== *is recommended that)*: used to indicate
  - among several possibilities one is recommended as particularly suitable, without mentioning or excluding others
  - or that a certain course of action is preferred but not necessarily required;

> **3.15 Legal, Copyright, and Other Notices**
>
> E-store should display the disclaimers, copyright, word mark, trademark and product warranties of the Marvel electronics and home entertainment.

https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf **See Section 10.2.2**

© NC State Software Engineering Faculty


35

# Traditional Requirements: Terminology

- **May** (== *is permitted to*): used to indicate a course of action permissible within the limits of the standard

> *3.8.2* ***Data Storage***
>
> The customer's web browser shall never display a customer's password. It shall always be echoed with special characters representing typed characters.
>
> The customer's web browser shall never display a customer's credit card number after retrieving from the database. It shall always be shown with just the last 4 digits of the credit card number.
>
> The system's back-end servers shall never display a customer's password. The customer's password may be reset but never shown.
>
> The system's back-end servers shall only be accessible to authenticated administrators.

https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf **See Section 10.2.2**

© NC State Software Engineering Faculty


36

# Traditional Requirements: Terminology

- **Can** (== *is able to*): used for statements of possibility and capability, whether material, physical, or causal

> *3.13.1* ***User Interfaces***
>
> The user interface for the software shall be compatible to any browser such as Internet Explorer, Mozilla or Netscape Navigator by which user can access to the system.
>
> The user interface shall be implemented using any tool or software package like Java Applet, MS Front Page, EJB etc.

https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf **See Section 10.2.2**

© NC State Software Engineering Faculty



37

# Traditional/Plan-Driven Templates

- The <system> **shall** <response>
- If <preconditions> **then** the <system> shall <response>
- When <trigger> the <system> **shall** <response>
- If <preconditions> **when** <trigger> the <system> **shall** <response>
- While <state> the <system> **shall** <response>
- Where <feature> is enabled the <system> **shall** <response>



38

## Use Case 4: Edit Recipe

**Exercise 14.04: Traditional Requirements**

### Preconditions

User selected the Edit Recipe option from the menu [UC1]

9duzoo

### Main Flow

1. The user will be shown a list of all recipes in the system.
2. The user may select one recipe that they want to edit [E1].
3. After selecting a recipe, the form will be pre-filled with all existing values (price and **units of all ingredients**) for the selected recipe.
4. The user edits the price and **the ingredients** [Add Ingredient] [Remove Ingredient] [Edit Units] that make up the recipe [Cannot Edit] [Invalid Price] [No Ingredients].
5. The user returns to the main menu [UC1].

39

39




# Use Case Requirements

© NC State Software Engineering Faculty

40

40

# Use Case Requirements

Use case diagrams (UML)

- Show relationships between actors (people and systems) and their connection to the use cases

Use case flow-of-events

- These are the UC requirements you've been seeing
- Textual description of a sequence of transactions
- Describe what the system should do, not how to do it

41

# First Principles of Use Cases

- Keep it simple by telling stories
- Understand the big picture
- Focus on value
- Build the system in slices
- Deliver the system in increments
- Adapt to meet the team's needs

Ivar Jacobson, Ian Spence, Kurt Bittner, "Use-Case 2.0 – The Guide to Succeeding with Use Cases," 2011.

© NC State Software Engineering Faculty

42

42

# Principle 1: Telling Stories

- Helps with describing what a system should do
- Capture the goals of the system and how to handle problems
- Story is captured and becomes part of the use case narrative

- Capture stories that are ***actionable*** and ***testable***
- Test cases demonstrate the system meets the requirements
  - ***Done criteria!***

Ivar Jacobson, Ian Spence, Kurt Bittner, "Use-Case 2.0 – The Guide to Succeeding with Use Cases," 2011.

© NC State Software Engineering Faculty




# Principle 2: Big Picture

- Understanding the big picture is important as your system changes and grows to make correct decisions about
  - What to include
  - What to leave out
  - What it will cost
  - What benefit it will provide
  - What is the scope
  - What is the progress

- Use case diagrams model the big picture

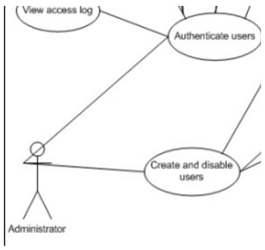
Ivar Jacobson, Ian Spence, Kurt Bittner, "Use-Case 2.0 – The Guide to Succeeding with Use Cases," 2011.

© NC State Software Engineering Faculty

## External System Behavior: Use Case Diagram

- Complete course of events in the system, from the user's perspective
- Use Cases Model: Illustrates
  - (use cases) the system's intended functions from the user's perspective – ovals
  - (actors) surroundings – external to the system – can be other systems – stick figures
  - (use case diagrams) relationships between use cases and actors

**The collection of all use cases is everything that can be done to/with the system**

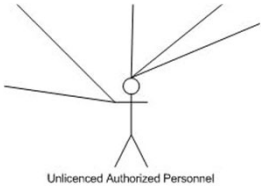



© NC State Software Engineering Faculty


45

## Actors

- Are NOT part of the system – they represent anyone or anything that must interact with the system
  - Only input information to the system
  - Only receive information from the system
  - Both input to and receive information from the system
- Represented in UML as a stickman, even when they are not "people", such as a billing system




© NC State Software Engineering Faculty


46

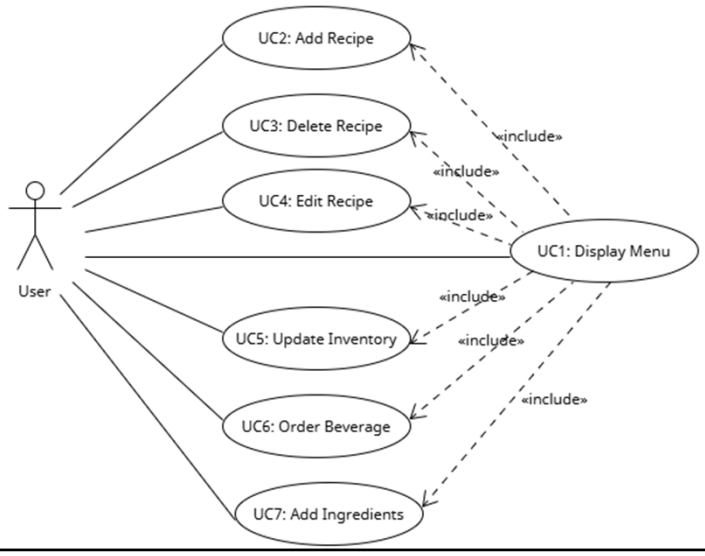
# Use Case Diagrams – Inclusion & Extension

- Inclusion of Use Cases
  - Explicitly called as an action from a basic flow
  - Incorporates the behavior of another use case
  - The callee is unaware of its caller
  - Allows for shared behavior
  - Example: authentication
- Extensions of Use Cases (inheritance)
  - A use case *V* that is invoked at a name extension point *p* in another use case *U*. *U* is unaware of the extension *V*.
  - Optional functionality under specific conditions
  - Example: recommendations

47

47

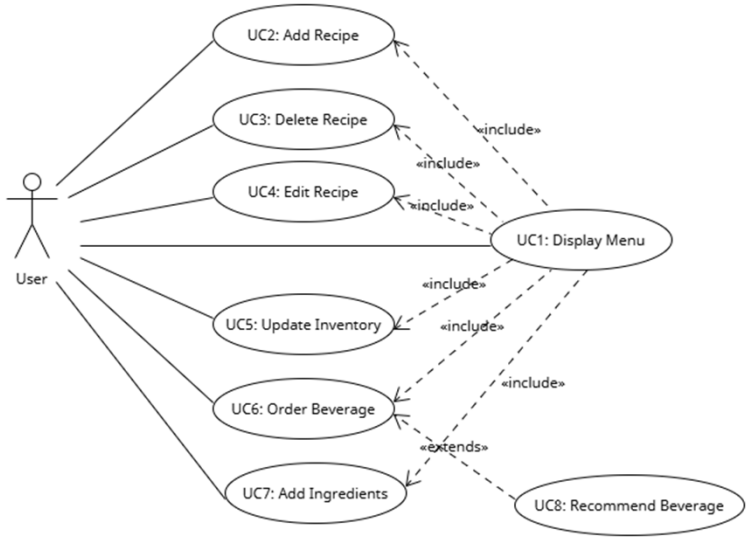
# CoffeeMaker Use Case Diagram




48

48

# CoffeeMaker Use Case Diagram 2








# Principle 3: Value

- "Value is only generated if the system is actually used…"
  - How a system will be used
  - Why the system is needed
- Use cases focus on how a system will achieve a goal for a user
  - Successful or happy paths
  - Problem or alternative paths
- Use case Narrative
  - Basic flow – steps that capture the interaction between user and system
  - Alternative flows – other ways of using the system or problem paths

Ivar Jacobson, Ian Spence, Kurt Bittner, "Use-Case 2.0 – The Guide to Succeeding with Use Cases," 2011.

© NC State Software Engineering Faculty

# Principles 4 – 6: Implementing

- Principle 4: Slices
  - Slice up a use case by stories and implement a story at a time
- Principle 5: Incremental Delivery
  - Provide value in increments by adding additional slices
- Principle 6: Adapt to Team
  - Scale up or down to team needs

Ivar Jacobson, Ian Spence, Kurt Bittner, "Use-Case 2.0 – The Guide to Succeeding with Use Cases," 2011.

© NC State Software Engineering Faculty

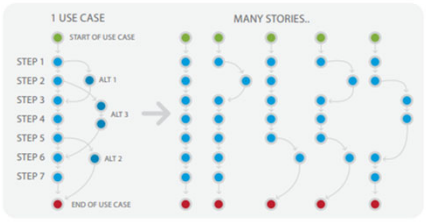



FIGURE 8:
THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES

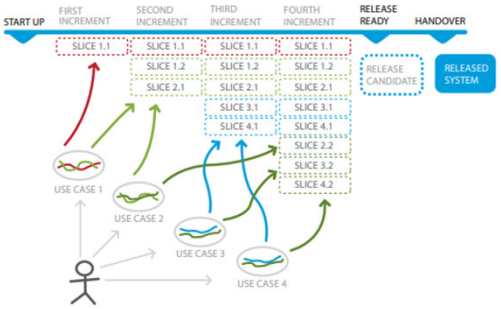
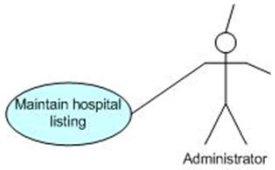



FIGURE 4: USE CASES, USE-CASE SLICES, INCREMENTS, AND RELEASES

51

# Use Case Narrative

- A sequence of transactions performed by a system that yields a measurable result of values for a particular actor/user
- A use case typically represents a major piece of functionality that is complete from beginning to end. A use case must deliver something of value to an actor.
- Use cases that an actor "wants" begin with verbs.




© NC State Software Engineering Faculty



52

# Use Case
## Template for Flow of Events

**UC-X Use Case Name**

**X.1 Preconditions**

**X.2 Main Flow**

**X.3 Subflows**

**X.4 Alternative Flows**

© NC State Software Engineering Faculty

## Use Case 4: Edit Recipe

**Preconditions**

User selected the Edit Recipe option from the menu [UC1]

**Main Flow**

1. The user will be shown a list of all recipes in the system.
2. The user may select one recipe that they want to edit [E1].
3. After selecting a recipe, the form will be pre-filled with all existing values (price and **units of all ingredients**) for the selected recipe.
4. The user edits the price and **the ingredients** [Add Ingredient] [Remove Ingredient] [Edit Units] that make up the recipe [Cannot Edit] [Invalid Price] [No Ingredients].
5. The user returns to the main menu [UC1].

**Subflows**

- **[Add Ingredient] A user can select a new ingredient to add to the recipe.**
- **[Remove Ingredient] A user can remove an ingredient in the recipe.**
- **[Edit Units] A user can provide a unit for the ingredient [Invalid Unit].**

**Alternative Flows**

- [Cannot Edit] If the recipe is deleted by a concurrent user before it's selected for edit, an error message is printed and the user is returned to the edit recipe page with an updated list of recipes.
- [Invalid Price] The price must be a positive integer. If the price is not a positive integer then, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [Invalid Unit] The units of coffee, sugar, milk, and chocolate must be positive integers. If the unit value on any of the ingredients is not a positive integer, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- **[No Ingredients] If the recipe has no ingredients, then an error message is printed, and the user is returned to the form to edit. The recipe is not added.**

53

# Template for Flow of Events

**UCX Use Case Name**

**X.1 Preconditions**

**What needs to happen (in another use case) before this use case can start?**

**X.2 Main Flow**

**Describes the different scenarios of the use case as steps**

**X.3 Subflows / Extensions**

**Break "normal" flow into pieces**

**"called" by Main Flow or another subflow**

**X.4 Alternative / Error Flows**

**Things that happen outside of the "normal" flow**

**"called" by Main Flow or a subflow**

Use Cases: Cover multiple related scenarios or stories!!!
- Related functionality
- Subflows detail scenarios

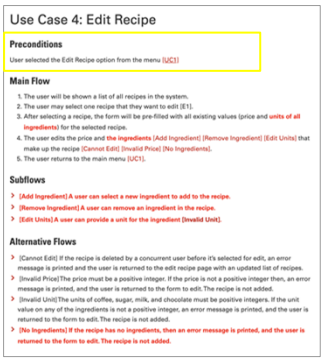
Additional Information (e.g., iTrust2)
- Logging
- Data Format
- Acceptance Scenarios (ex. UC8)

© NC State Software Engineering Faculty



54

# Use Case Elements


Use Case 4: Edit Recipe

**Preconditions**

User selected the Edit Recipe option from the menu [UC1]

**Main Flow**

1. The user will be shown a list of all recipes in the system.
2. The user may select one recipe that they want to edit [E1].
3. After selecting a recipe, the form will be pre-filled with all existing values (price and units of all ingredients) for the selected recipe.
4. The user edits the price and the ingredients [Add Ingredient] [Remove Ingredient] [Edit Units] that make up the recipe [Cannot Edit] [Invalid Price] [No Ingredients].
5. The user returns to the main menu [UC1].

**Subflows**

- [Add Ingredient] A user can select a new ingredient to add to the recipe.
- [Remove Ingredient] A user can remove an ingredient in the recipe.
- [Edit Units] A user can provide a unit for the ingredient [Invalid Unit].

**Alternative Flows**

- [Cannot Edit] If the recipe is deleted by a concurrent user before it's selected for edit, an error message is printed and the user is returned to the edit recipe page with an updated list of recipes.
- [Invalid Price] The price must be a positive integer. If the price is not a positive integer then, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [Invalid Unit] The units of coffee, sugar, milk, and chocolate must be positive integers. If the unit value on any of the ingredients is not a positive integer, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [No Ingredients] If the recipe has no ingredients, then an error message is printed, and the user is returned to the form to edit. The recipe is not added.

## Use Case 4: Edit Recipe

### Preconditions

User selected the Edit Recipe option from the menu [UC1]

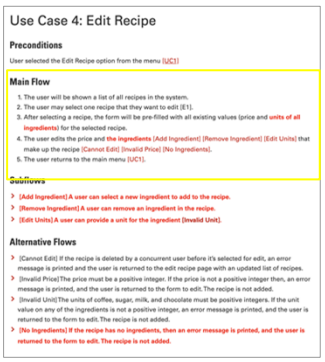
"Role"

© NC State Software Engineering Faculty





# Use Case


Use Case 4: Edit Recipe

**Preconditions**

User selected the Edit Recipe option from the menu [UC1]

**Main Flow**

1. The user will be shown a list of all recipes in the system.
2. The user may select one recipe that they want to edit [E1].
3. After selecting a recipe, the form will be pre-filled with all existing values (price and units of all ingredients) for the selected recipe.
4. The user edits the price and the ingredients [Add Ingredient] [Remove Ingredient] [Edit Units] that make up the recipe [Cannot Edit] [Invalid Price] [No Ingredients].
5. The user returns to the main menu [UC1].

**Subflows**

- [Add Ingredient] A user can select a new ingredient to add to the recipe.
- [Remove Ingredient] A user can remove an ingredient in the recipe.
- [Edit Units] A user can provide a unit for the ingredient [Invalid Unit].

**Alternative Flows**

- [Cannot Edit] If the recipe is deleted by a concurrent user before it's selected for edit, an error message is printed and the user is returned to the edit recipe page with an updated list of recipes.
- [Invalid Price] The price must be a positive integer. If the price is not a positive integer then, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [Invalid Unit] The units of coffee, sugar, milk, and chocolate must be positive integers. If the unit value on any of the ingredients is not a positive integer, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [No Ingredients] If the recipe has no ingredients, then an error message is printed, and the user is returned to the form to edit. The recipe is not added.

[E1]

## Use Case 4: Edit Recipe

### Preconditions

User selected the Edit Recipe option from the menu [UC1]

### Main Flow

1. The user will be shown a list of all recipes in the system.
2. The user may select one recipe that they want to edit [E1].
3. After selecting a recipe, the form will be pre-filled with all existing values (price and **units of all ingredients**) for the selected recipe.
4. The user edits the price and **the ingredients** [Add Ingredient] [Remove Ingredient] [Edit Units] that make up the recipe [Cannot Edit] [Invalid Price] [No Ingredients].
5. The user returns to the main menu [UC1].

[Add Ingredient]

### Subflows

- **[Add Ingredient] A user can select a new ingredient to add to the recipe.**
- **[Remove Ingredient] A user can remove an ingredient in the recipe.**
- **[Edit Units] A user can provide a unit for the ingredient [Invalid Unit].**

### Alternative Flows

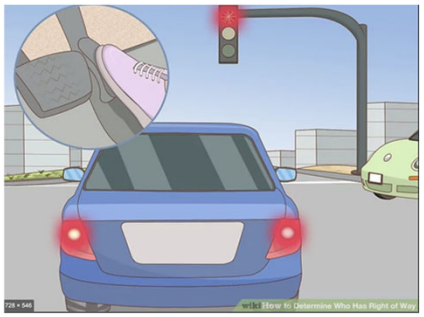
- [Cannot Edit] If the recipe is deleted by a concurrent user before it's selected for edit, an error message is printed and the user is returned to the edit recipe page with an updated list of recipes.
- [Invalid Price] The price must be a positive integer. If the price is not a positive integer then, an error message is printed, and the user is returned to the form to edit. The recipe is not added.
- [Invalid Unit] The units of coffee, sugar, milk, and chocolate must be positive integers. If the unit value on any of the ingredients is not a positive integer, an error message is printed, and the user is returned to the form to edit. The recipe is not added.

© NC State Software Engineering Faculty

# Use Case Example: Driving

- Story: Clear Intersection

A driver wants to drive through an intersection. The driver can only clear through the intersection if the traffic light is green and there are no cars in the intersection. Otherwise, the car needs to join a queue.



© NC State Software Engineering Faculty





# Flow of Events for the Clear Intersection Use Case

**Preconditions:** the traffic light has been initialized

Actor / User

**Main Flow**

1. The driver approaches the intersection.
2. The driver checks the status [Check Status].
3. The driver clears the intersection [Go].

Connections for flow of events

**Subflows**

- **[Check Status]** The driver checks the light [Check Light] and the queue [Check Queue]. If the light is green and the queue is empty the driver clears the intersection [Go]. Otherwise, the driver joins a queue [Join Queue].
- **[Check Light]** Check if the light is red, yellow, or green.
- **[Check Queue]** Check if the queue is empty or not.
- **[Go]** The driver clears the intersection and the use case ends.
- **[Join Queue]** The driver joins the end of the queue and checks status every 15 seconds [Check Status].

© NC State Software Engineering Faculty

## Scenario/Story: Car approaches intersection with green light and no queue



**Preconditions:** the traffic light has been initialized

**Main Flow**

1. The driver approaches the intersection.
2. The driver checks the status [Check Status].
3. The driver clears the intersection [Go].

**Subflows**

- **[Check Status]** The driver checks the light [Check Light] and the queue [Check Queue]. If the light is green and the queue is empty the driver clears the intersection [Go]. Otherwise, the driver joins a queue [Join Queue].
- **[Check Light]** Check if the light is red, yellow, or green.
- **[Check Queue]** Check if the queue is empty or not.
- **[Go]** The driver clears the intersection and the use case ends.
- **[Join Queue]** The driver joins the end of the queue and checks status every 15 seconds [Check Status].

© NC State Software Engineering Faculty



59

## Scenario/Story: Car approaches intersection with red light and no queue



**Preconditions:** the traffic light has been initialized

**Main Flow**

1. The driver approaches the intersection.
2. The driver checks the status [Check Status].
3. The driver clears the intersection [Go].

**Subflows**

- **[Check Status]** The driver checks the light [Check Light] and the queue [Check Queue]. If the light is green and the queue is empty the driver clears the intersection [Go]. Otherwise, the driver joins a queue [Join Queue].
- **[Check Light]** Check if the light is red, yellow, or green.
- **[Check Queue]** Check if the queue is empty or not.
- **[Go]** The driver clears the intersection and the use case ends.
- **[Join Queue]** The driver joins the end of the queue and checks status every 15 seconds [Check Status].

© NC State Software Engineering Faculty



60

## More Scenarios/Stories: Alternative Flows



**Preconditions:** the traffic light has been initialized

**Main Flow**

1. The driver approaches the intersection.
2. The driver checks the status [Check Status].
3. The driver clears the intersection [Go].

**Subflows**

- **[Check Status]** The driver checks the light [Check Light] and the queue [Check Queue][Light Out][Accident][Ice Storm]. If the light is green and the queue is empty the driver clears the intersection [Go]. Otherwise, the driver joins a queue [Join Queue].
- **[Check Light]** Check if the light is red, yellow, or green.
- **[Check Queue]** Check if the queue is empty or not.
- **[Go]** The driver clears the intersection and the use case ends.
- **[Join Queue]** The driver joins the end of the queue and checks status every 15 seconds [Check Status].

**Alternative Flows**

- **[Light Out]** The light is not turned on. Wait for a clear intersection and peel out.
- **[Accident]** An accident is blocking the intersection. Rubber neck and slowly drive around it.
- **[Ice Storm]** There is an ice storm preventing safe driving. Abandon your car and walk.

© NC State Software Engineering Faculty




## Flow of Events vs Scenario/Story

- Flow of events enumerates all subflows and exception flows.
- Scenario or Story is one path through your flow of events
  - Becomes a slice of the use case that is used to push forward development
- When you're testing, make sure you cover a reasonable set of scenarios.
  - That way you know you're done!

© NC State Software Engineering Faculty