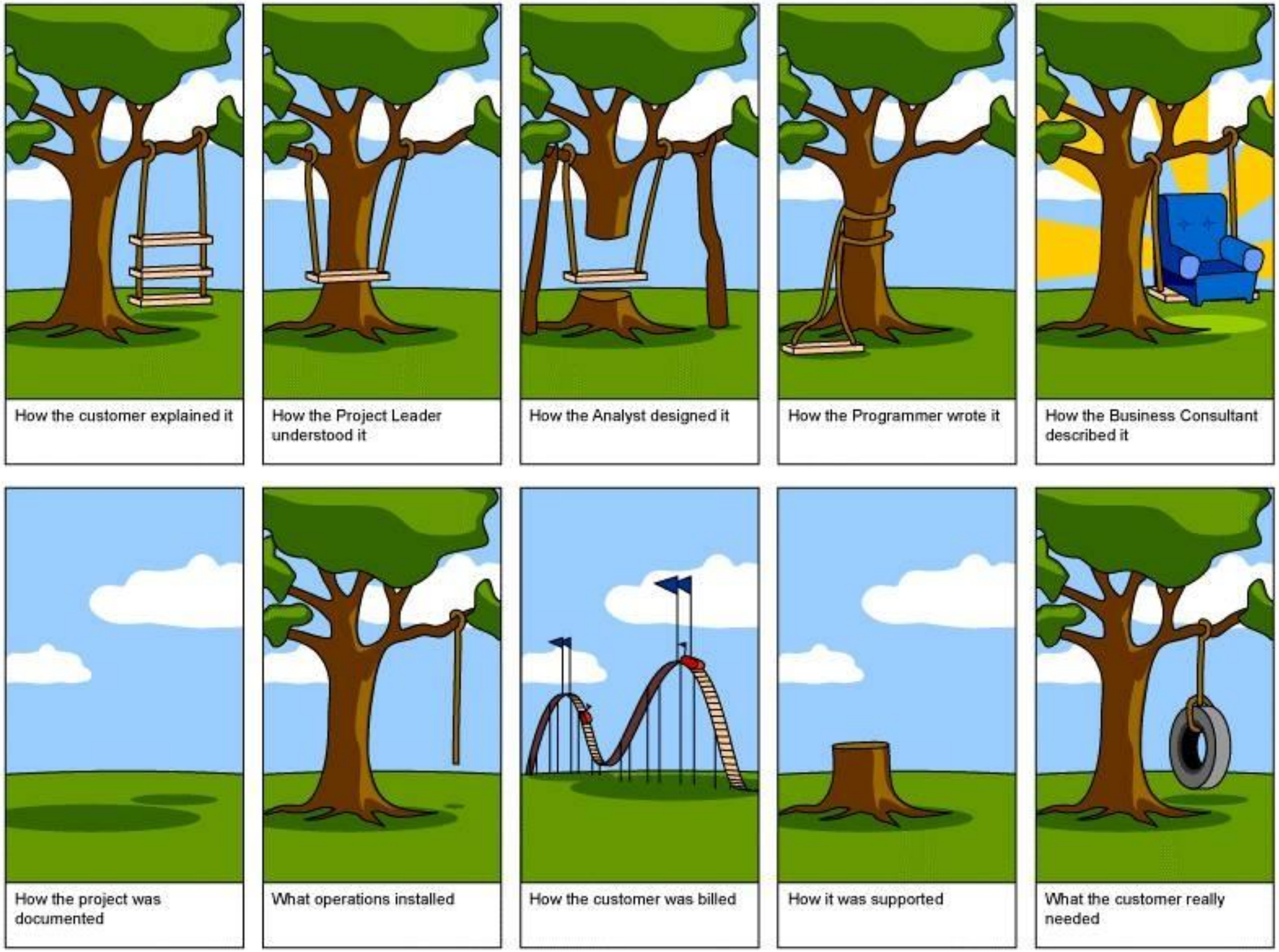
Requirements Modeling

CSC 326 – Spring 2023

Dr. Sandeep Kuttal

© NC State Software Engineering Faculty


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Requirements

- **Software Requirement**: condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component. Represented in a contract, standard, specification, or other formally imposed document
  - Informally: a description of what the customers want
  - May be buried beneath layers of assumptions, misconceptions, and politics

Requirements

- Requirements may range from a **high-level** abstract statement of a service or of a system constraint to a **detailed** mathematical functional specification

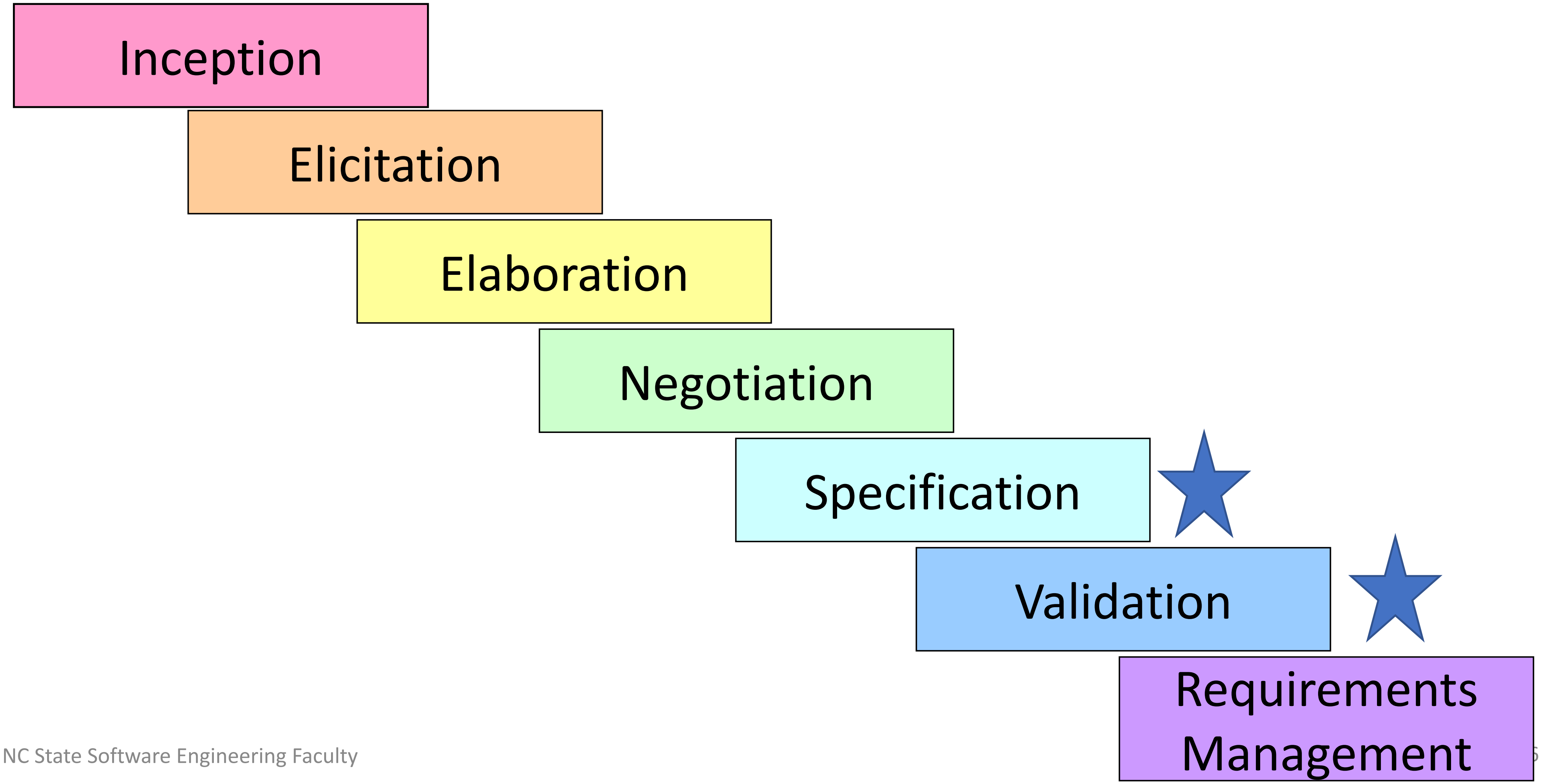
- This is inevitable as requirements may serve a dual function
  - May be the **basis for a bid for a contract** – therefore must be **open for interpretation**
  - May be the **basis for the contract** itself – therefore must be **defined in detail**
  - Both of these statements may be called requirements

© NC State Software Engineering Faculty

Requirements Engineering

- The **process of** establishing the services that the **customer requires from a system** and the **constraints** under which it operates and is developed
- Results in a specification of the system that stakeholders understands
  - natural language (e.g., use cases)
  - easy to understand pictures (UML Diagrams)
- Requirements may be **functional** or **non-functional**
- A stakeholder is a key representative of the groups who have a vested interest in your system and direct or indirect influence on its requirements.

Requirements Engineering Tasks



© NC State Software Engineering Faculty

REQUIREMENTS: Two Audiences

SOURCE CODE
Programming Language
(Java, C++, SQL)

Machines | Developers

REQUIREMENTS
Natural Language

Developers | Stakeholders

CSC326

SOURCE CODE
Programming Language
(Java, C++, SQL)

Machines | Students

REQUIREMENTS
Natural Language

Students | Teaching Staff

Requirements Engineering

- **Analysis**: studying user needs to generate system definition that users understand
  - Fact-finding
  - Communication
  - Fact-validation

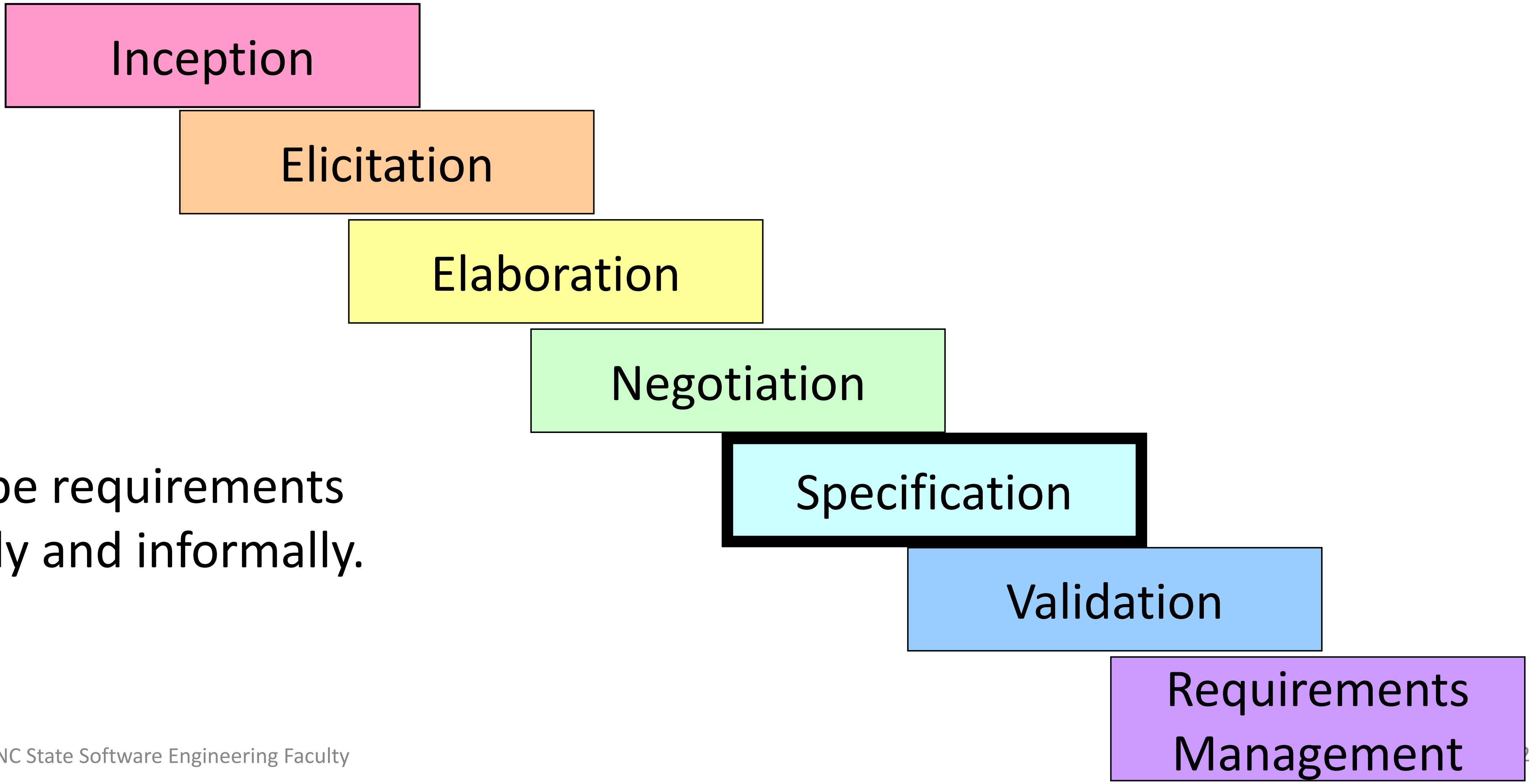
- **Modeling**: translating requirements the user understands to a form that software engineers understand
  - Representation
  - Organization

Types of Requirements

- Functional requirements: *requirements that specify a function that a system or system component must be able to perform*
  - **Example:** The watch shall display the time.
- Non-functional requirements: not specifically concerned with the functionality of a system but place restrictions on the product being developed
  - User visible aspects of the system not directly related to functional behavior
  - Usability; reliability; privacy; security; availability; performance
  - Best to translate non-functional to measurable.
    - ***Example:*** The response time must be less than 1 second
- Constraints ("Pseudo requirements"): not user-visible; imposed by the client that restricts the implementation of the system or the *development process*
  - ***Example:*** The implementation language must be Java.
  - ***Example:*** Unit tests must be written in JUnit.

Requirement Specification

Requirements Engineering Tasks



Describe requirements formally and informally.

Technically Speaking, "requirement" ≠ "specification"

- Requirement – understanding between customer and supplier
- Specification – what the software must do
- Requirements that are not in the SRS
  - Costs
  - Delivery dates
  - Acceptance procedures
  - etc

Specifications (Generally)

- A broad term that means definition
- From Pressman,
  - The invention and definition of a behavior of a solution system such that it will produce the required effects in the problem domain
- A statement of agreement (contract) between
  - producer and consumer of a service

Specifications (Generally)

- Document many aspects of projects:
  - What the users need (requirements)
  - The features offered by a system (functional spec)
  - The performance characteristics
  - The design of the software (e.g. architecture)
  - External behavior of a module
  - Internal structure of a module

Requirements Specifications

- Statement of user requirements
- Major failures occur through misunderstandings between the producer and the user
  - Client: “But I expected that the system would be able to handle that.”
  - Developer: “Well, that’s not the impression you gave me. I thought that you wanted it to operate like this.”

Specification Qualities

- Consistency
- Completeness
- Verifiable
- Traceable
- Precise, clear, unambiguous

Consistency

- Requirements shouldn’t contradict each other
- Especially problematic when multiple types of requirements documents are used
    - E.g., natural language for customers, semi-structured models for engineers
- Use consistent terms for major items in the requirements
    - Example: Doctor, Health Care Provider (HCP), Nurse, Nurse Practitioner

Completeness

- Internal completeness
  - The specification must define any new concept or terminology that it uses
    - Glossary helpful for this purpose
  - Roles should be described and then used consistently
- External completeness
  - The specification must document all the needed requirements to satisfy the stakeholder needs
    - Difficulty: when should one stop?

Dealing with Incompleteness

- Lacking a piece of information about a specific requirement?
  - Need to consult with stakeholders
  - Need to check an external interface description
  - Need to build a prototype
- Use TBD (To Be Determined) to flag all knowledge gaps
- Resolve all TBDs before signing off on a set of requirements

Verifiable, Traceable

- **Verifiable**: it is possible to verify that requirements have been met
    - A system test can be developed to determine that the requirement is fully implemented
- **Traceable**: requirements can be linked to subsequent design, code, and test artifacts
    - A developer should be able to find the portion of the design, code, and test that correspond to a requirement

Precise, Clear, Unambiguous

- Requirements should clearly describe the system functionality.
- Example (from a real safety-critical system)

The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.

Q: Can a message be accepted as soon as we receive 2 out of 3 identical copies of message or do we need to wait for receipt of the 3rd?

Guidelines for Writing Requirements

- 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.

Requirements Readability

- Label sections, subsections, and individual requirements consistently
- Use white space liberally
- Use visual emphasis (e.g., bold, underline, italics, and different fonts) consistently
- Include a table of contents and possibly an index to help readers find information they need
- Number all figures and tables, give them captions, and refer to them by number

IEEE Requirements Standard

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

$\big\}$ **Supporting information**

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)

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

3. Specific Requirements

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

3. Specific Requirements (Traditional Requirements Example)

- 3.1 External interfaces
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communication interfaces
- 3.2 System features
    - 3.2.1 System feature 1
        - 3.2.1.1 Purpose of feature
        - 3.2.1.2 Associated functional requirements
        - 3.2.3.1 Functional requirement 1
        - …
        - 3.2.3.n Functional requirement n
    - 3.2.2 System feature 2
    - …
    - 3.2.m System feature 3
    - …
- 3.3 Non-functional Requirements (performance, etc.)
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Functional requirements can be modeled in different ways

- *Traditional Requirements*
- Use Cases
- User Stories

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

Prototyping

Prototyping

- Prototypes may be implementations or mockups
- Prototypes for requirements validation demonstrate the requirements and help stakeholders discover problems
- Validation prototypes should be reasonably efficient and robust
- It should be possible to use them in the same way as in the required system (within their limits)
- User documentation and training may need to be provided

Prototyping Activities

- Choose prototype evaluators
  - The best evaluators are users who are fairly experienced and who understand the nature of prototypes.
  - End-users who do different jobs should be involved so that different areas of system functionality will be covered.
- Develop test scenarios
  - Careful planning is required to draw up a set of evaluation scenarios that provide broad coverage of requirements.
  - End-users shouldn’t just play around with the system as this may never exercise critical system features.

Prototyping Activities

- Execute scenarios
  - The users of the system may work, on their own, to try the system by executing the planned scenarios.
  - Alternatively, some form of observational or think-aloud protocol can be used
- Document problems
  - It is usually best to define some kind of electronic or paper problem report form which users fill in when they encounter a problem.

Prototyping Benefits

- Misunderstandings between software users and developers are exposed
- Missing services may be detected
- Confusing services may be identified
- Prototyping can also be used in the Requirements Analysis phase
- The prototype may serve as a basis for preparing a precise system specification

Prototyping Drawbacks

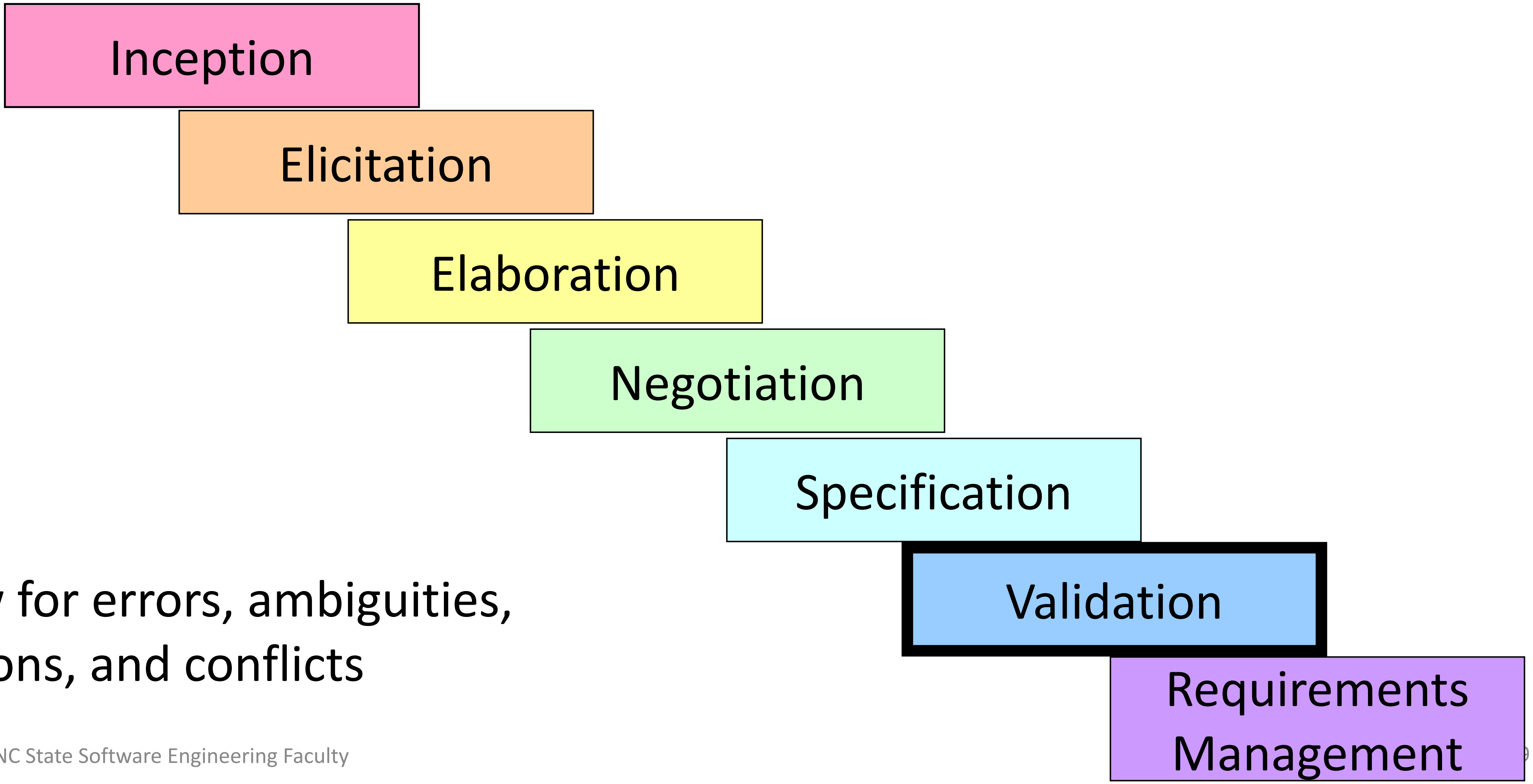
- Prototypes may lead users to think that system construction will be easier than anticipated
- There may be a temptation to morph the prototype into the delivered solution. Don’t do this! (“Plan to throw one away” – Brooks).

Paper Prototypes (Interface Mockups)

- Paper prototype of an interface
- Can be similar to “storyboards” (illustrations or images used to pre-visualize things like films or animations)
- Allow the reader to “walk through” the interface, seeing the different screens and layouts provided
- Use captions to explain what is shown

Requirements Validation

Requirements Engineering Tasks



Review for errors, ambiguities, omissions, and conflicts

Requirements Validation via Inspection

- A group of people read and analyze the requirements, look for problems, meet and discuss the problems and agree on actions to address these problems

Inspection Team Membership

- Inspections should involve a number of stakeholders of different backgrounds
  - People from different backgrounds bring different skills and knowledge to the inspection
  - Stakeholders should feel involved in the RE process and seek an understanding of the needs of other stakeholders
- Inspection team should ideally include (among others) a domain expert an end-user, and representatives of the client

Requirements Validation Considerations

- Correctness
  - How well do the requirements represent the client’s view?
- Completeness
  - Are all possible scenarios through the system are described, including exceptional behavior by the user or the system?
  - Are there any missing requirements or is there any information missing from requirement descriptions?
- Consistency
  - Are there any functional and/or non-functional requirements that contradict each other?
  - Do the descriptions of different requirements include contradictions?

Requirements Validation Considerations

- Clarity
  - Are there any ambiguities in the requirements?
  - Are the requirements expressed using terms that are clearly defined?
  - Could readers from different backgrounds make different interpretations of the requirements?
- Feasibility
  - Can the requirements be implemented and delivered given current technologies?
- Traceability
  - Are requirements unambiguously identified with links to related requirements and reasons why requirements are included?

© NC State Software Engineering Faculty

Requirements Validation Considerations

- Understandability
  - Can readers of the document understand what the requirements mean?
- Non-prescriptive
  - Doe the requirements describe what the customer wants and nothing about how the programmers will implement the system?
- Consistent Language
  - Is consistent language used throughout?
- Testable
  - Are requirements measurable and testable?
  - Are error flows and success criteria specified?
  - Can a system test describe what a fully implemented requirement should look like?

Requirements Validation Considerations

- Concise
  - Could the requirements description be written in a more concise manner?
- Organization
  - Is the document structured in a sensible way?
  - Are the descriptions of requirements organized so that related requirements are grouped?
- Conformance to Standards
  - Do the requirements and specification conform to defined standards?
  - Are any departures from standards justified?

© NC State Software Engineering Faculty

Requirements Modeling

Types of Requirements Statements

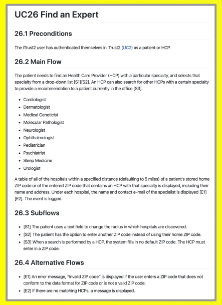
- Traditional
  - “The system shall…”
  - Very formal
- Use case based
  - Description of system from user perspective
- User story – married with acceptance test to supply the detail
  - Agile requirements

@ NC State Software Engineering Faculty

Spectrum of Appropriate Requirements Processes and Tools

$\longleftarrow$ *Requirements Process* $\longrightarrow$

more formal *less formal*



UC26 Find an Expert

26.1 Preconditions

The iTrust2 user has authenticated themselves in iTrust2 (UC2) as a patient or HCP.

26.2 Main Flow

The patient needs to find an Health Care Provider (HCP) with a particular specialty, and selects that specialty from a drop-down list [S1][S2]. An HCP can also search for other HCPs with a certain specialty to provide a recommendation to a patient currently in the office [S3].

- Cardiologist
- Dermatologist
- Medical Geneticist
- Molecular Pathologist
- Neurologist
- Ophthalmologist
- Pediatrician
- Psychiatrist
- Sleep Medicine
- Urologist

A table of all of the hospitals within a specified distance (defaulting to 5 miles) of a patient's stored home ZIP code or of the entered ZIP code that contains an HCP with that specialty is displayed, including their name and address. Under each hospital, the name and contact e-mail of the specialist is displayed [E1][E2]. The event is logged.

26.3 Subflows

- [S1] The patient uses a text field to change the radius in which hospitals are discovered.
- [S2] The patient has the option to enter another ZIP code instead of using their home ZIP code.
- [S3] When a search is performed by a HCP, the system fills in no default ZIP code. The HCP must enter in a ZIP code.

26.4 Alternative Flows

- [E1] An error message, "Invalid ZIP code" is displayed if the user enters a ZIP code that does not conform to the data format for ZIP code or is not a valid ZIP code.
- [E2] If there are no matching HCPs, a message is displayed.

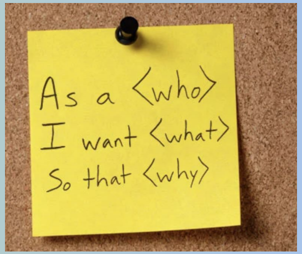
Traditional Requirements Use Cases

- Stable
- Larger
- Safety Critical
  - Examples:
    - Surgery Robots
    - Health Care
    - Nuclear Power Plant

Img: istock

- Plan-driven, well-documented

User Stories



As a <who>
I want <what>
So that <why>

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

TikTok

- Agile, Scrum, XP

Traditional 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 obsolete)
- **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;
  - or that (in the negative form) a certain course of action is deprecated but not prohibited
- **May** (== *is permitted to*): used to indicate a course of action permissible within the limits of the standard
- **Can** (== *is able to*): used for statements of possibility and capability, whether material, physical, or causal

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

Traditional Requirements

FR = Function Requirement

- FR2.4 Go to Jail

  When the player lands on the Go to Jail cell, the player *shall* be sent to the Jail cell. The player *shall* not receive $200 if she or he passes the Go cell on the way to the Jell cell.

- FR2.5 Buy Property

  When the player lands on a tradable cell, including properties, railroads, and utilities, she or he *shall* have a chance to buy that cell given that the cell is available. If the player clicks on the Buy button, the cell *shall* be sold to the player. See FR3 for the price rules on the properties, railroads, and utilities.

Traditional Requirements

NFR = Non-Functional Requirement

- NFR1.1 User Response

  The system shall respond to any user input within 0.01 seconds.

- Constraints
  - All code development shall be done with the Scala programming language
  - All testing shall be done using JUnit, Cucumber, and Selenium

Use Case Requirements

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

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

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

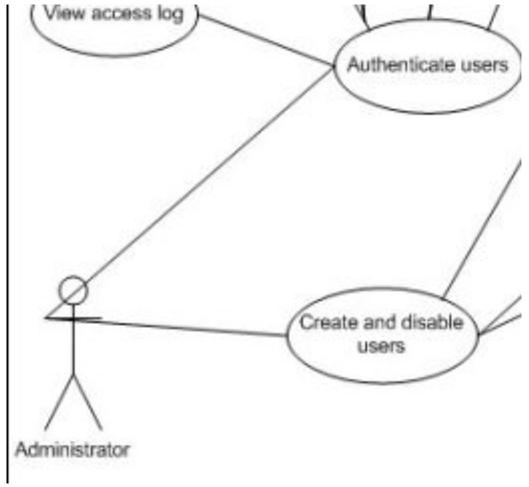
- Use case diagrams model the big picture

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

External System Behavior: Use Case Diagram

- Complete course of events in the system, <u>from the user's perspective</u>
- 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

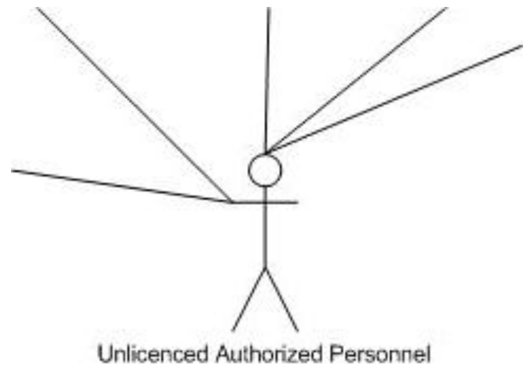
The collection of <u>all</u> use cases is <u>everything</u> that can be done to/with the system



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



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.

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

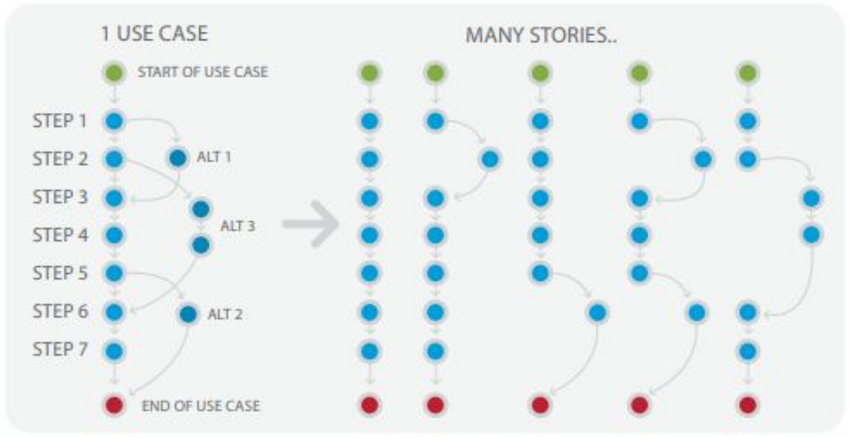


FIGURE 8:
THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES

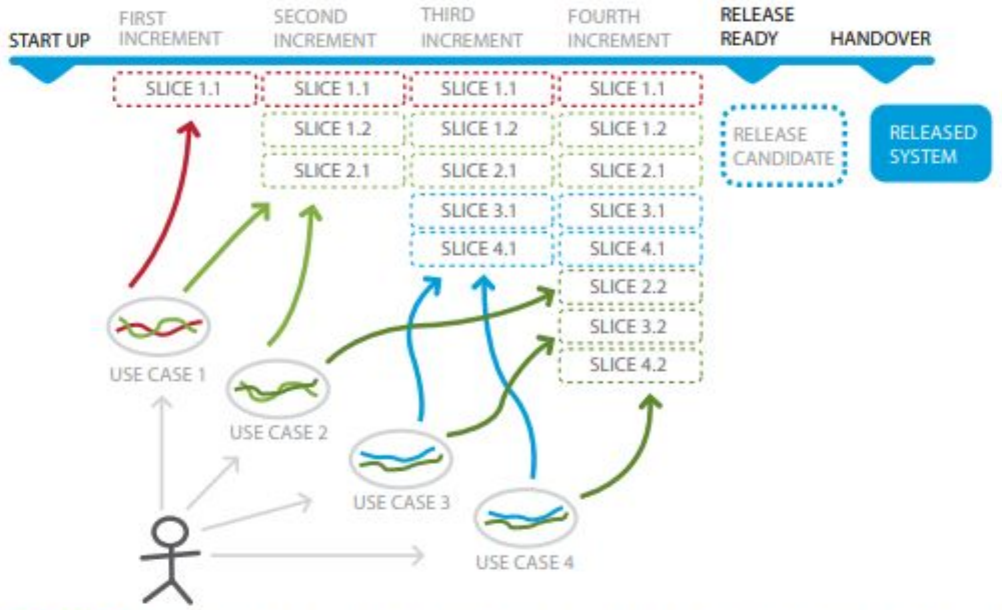
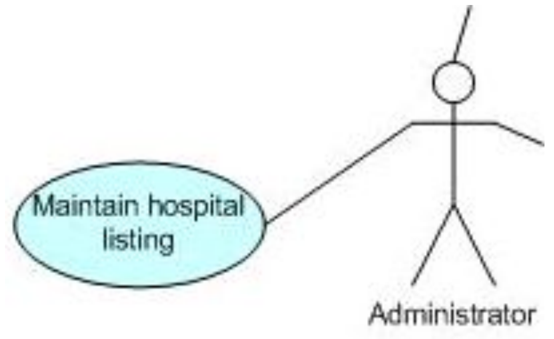


FIGURE 4: USE CASES, USE-CASE SLICES, INCREMENTS, AND RELEASES

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

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.



Use Case
Template for Flow of Events

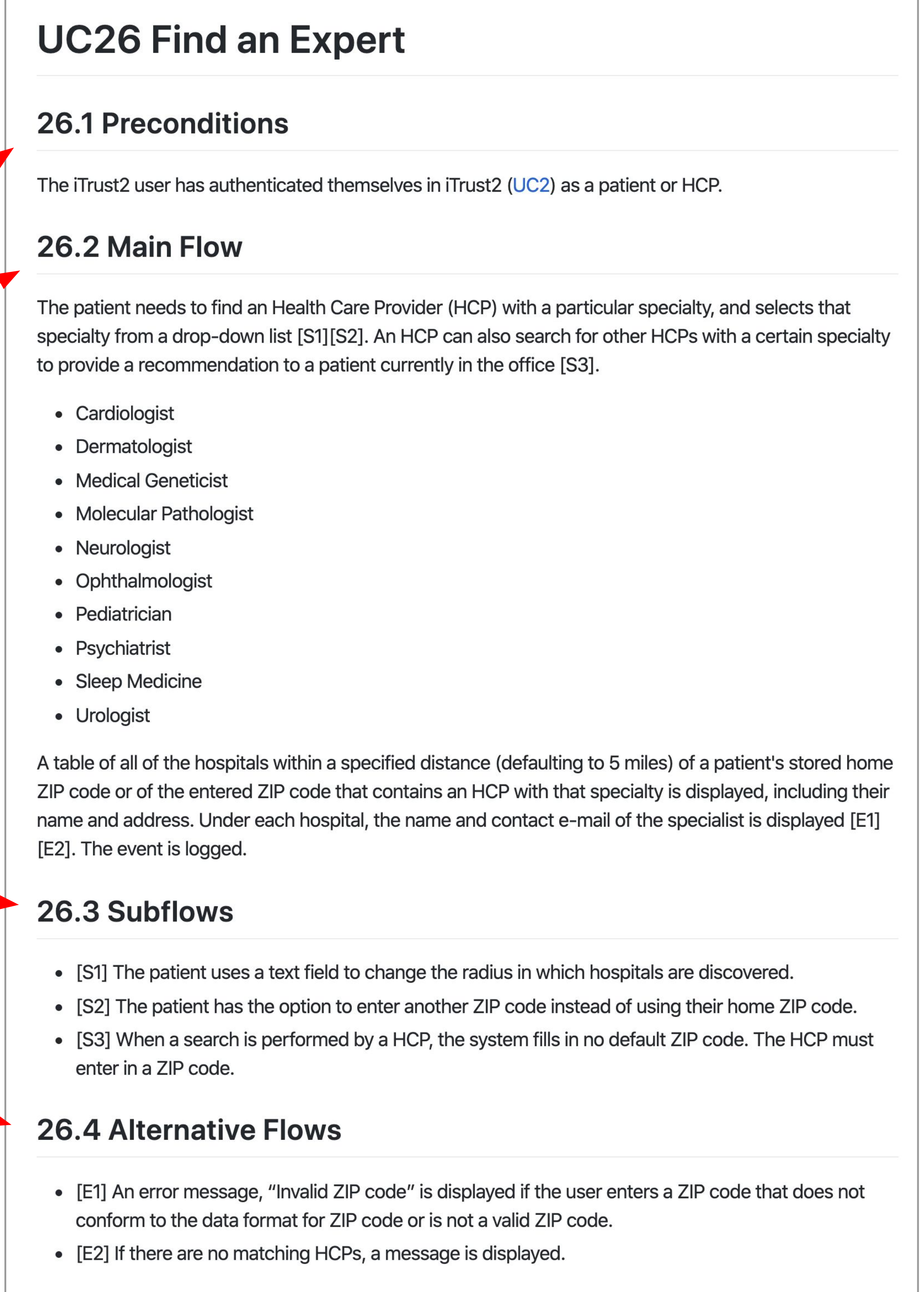
UC-X Use Case Name

X.1 Preconditions

X.2 Main Flow

X.3 Subflows

X.4 Alternative Flows



UC26 Find an Expert

26.1 Preconditions

The iTrust2 user has authenticated themselves in iTrust2 (UC2) as a patient or HCP.

26.2 Main Flow

The patient needs to find an Health Care Provider (HCP) with a particular specialty, and selects that specialty from a drop-down list [S1][S2]. An HCP can also search for other HCPs with a certain specialty to provide a recommendation to a patient currently in the office [S3].

- Cardiologist
- Dermatologist
- Medical Geneticist
- Molecular Pathologist
- Neurologist
- Ophthalmologist
- Pediatrician
- Psychiatrist
- Sleep Medicine
- Urologist

A table of all of the hospitals within a specified distance (defaulting to 5 miles) of a patient's stored home ZIP code or of the entered ZIP code that contains an HCP with that specialty is displayed, including their name and address. Under each hospital, the name and contact e-mail of the specialist is displayed [E1][E2]. The event is logged.

26.3 Subflows

- [S1] The patient uses a text field to change the radius in which hospitals are discovered.
- [S2] The patient has the option to enter another ZIP code instead of using their home ZIP code.
- [S3] When a search is performed by a HCP, the system fills in no default ZIP code. The HCP must enter in a ZIP code.

26.4 Alternative Flows

- [E1] An error message, “Invalid ZIP code” is displayed if the user enters a ZIP code that does not conform to the data format for ZIP code or is not a valid ZIP code.
- [E2] If there are no matching HCPs, a message is displayed.

© NC State Software Engineering Faculty

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

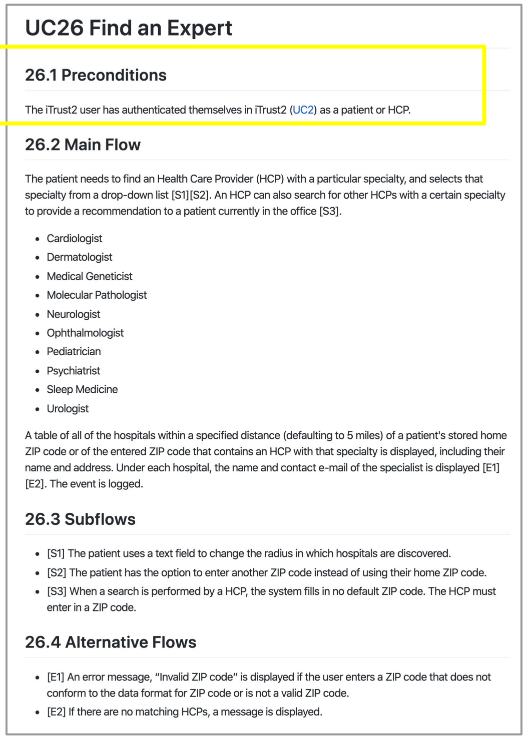
Use Cases Covers multiple related scenarios or stories!!!

- Related functionality
- Subflows detail scenarios

Additional Information (e.g., iTrust2)

- Logging
- Data Format
- Acceptance Scenarios (ex. UC8)

Use Case Elements



UC26 Find an Expert

26.1 Preconditions

The iTrust2 user has authenticated themselves in iTrust2 (UC2) as a patient or HCP.

26.2 Main Flow

The patient needs to find an Health Care Provider (HCP) with a particular specialty, and selects that specialty from a drop-down list [S1][S2]. An HCP can also search for other HCPs with a certain specialty to provide a recommendation to a patient currently in the office [S3].

- Cardiologist
- Dermatologist
- Medical Geneticist
- Molecular Pathologist
- Neurologist
- Ophthalmologist
- Pediatrician
- Psychiatrist
- Sleep Medicine
- Urologist

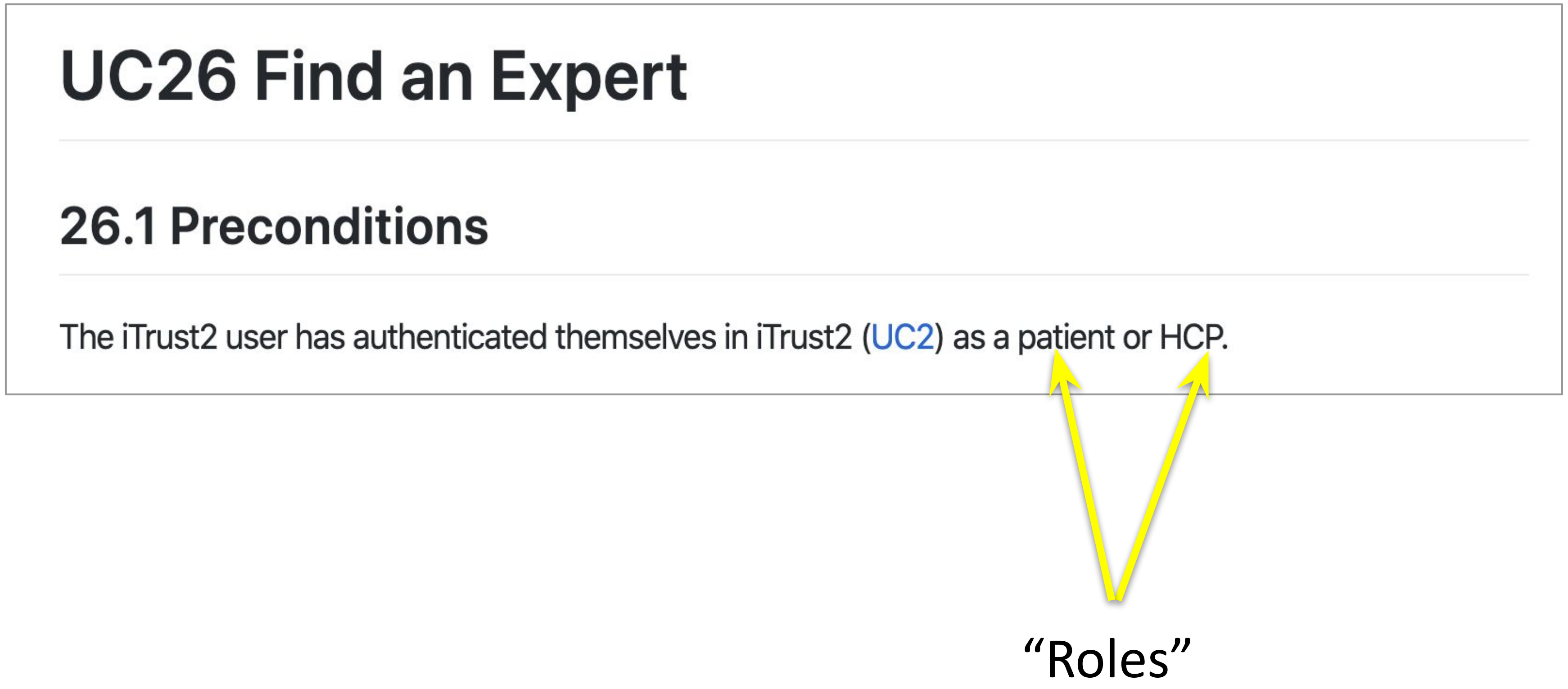
A table of all of the hospitals within a specified distance (defaulting to 5 miles) of a patient's stored home ZIP code or of the entered ZIP code that contains an HCP with that specialty is displayed, including their name and address. Under each hospital, the name and contact e-mail of the specialist is displayed [E1] [E2]. The event is logged.

26.3 Subflows

- [S1] The patient uses a text field to change the radius in which hospitals are discovered.
- [S2] The patient has the option to enter another ZIP code instead of using their home ZIP code.
- [S3] When a search is performed by a HCP, the system fills in no default ZIP code. The HCP must enter in a ZIP code.

26.4 Alternative Flows

- [E1] An error message, "Invalid ZIP code" is displayed if the user enters a ZIP code that does not conform to the data format for ZIP code or is not a valid ZIP code.
- [E2] If there are no matching HCPs, a message is displayed.

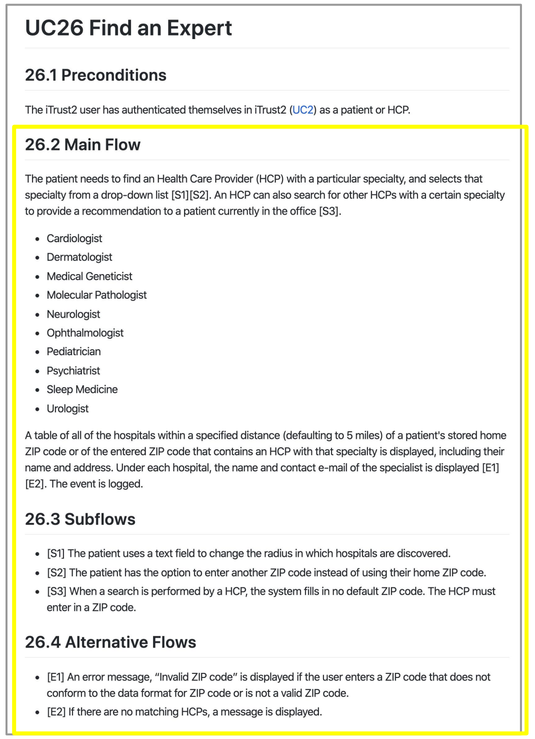


UC26 Find an Expert

26.1 Preconditions

The iTrust2 user has authenticated themselves in iTrust2 (UC2) as a patient or HCP.

Use Case



UC26 Find an Expert

26.1 Preconditions

The iTrust2 user has authenticated themselves in iTrust2 (UC2) as a patient or HCP.

26.2 Main Flow

The patient needs to find an Health Care Provider (HCP) with a particular specialty, and selects that specialty from a drop-down list [S1][S2]. An HCP can also search for other HCPs with a certain specialty to provide a recommendation to a patient currently in the office [S3].

- Cardiologist
- Dermatologist
- Medical Geneticist
- Molecular Pathologist
- Neurologist
- Ophthalmologist
- Pediatrician
- Psychiatrist
- Sleep Medicine
- Urologist

A table of all of the hospitals within a specified distance (defaulting to 5 miles) of a patient's stored home ZIP code or of the entered ZIP code that contains an HCP with that specialty is displayed, including their name and address. Under each hospital, the name and contact e-mail of the specialist is displayed [E1][E2]. The event is logged.

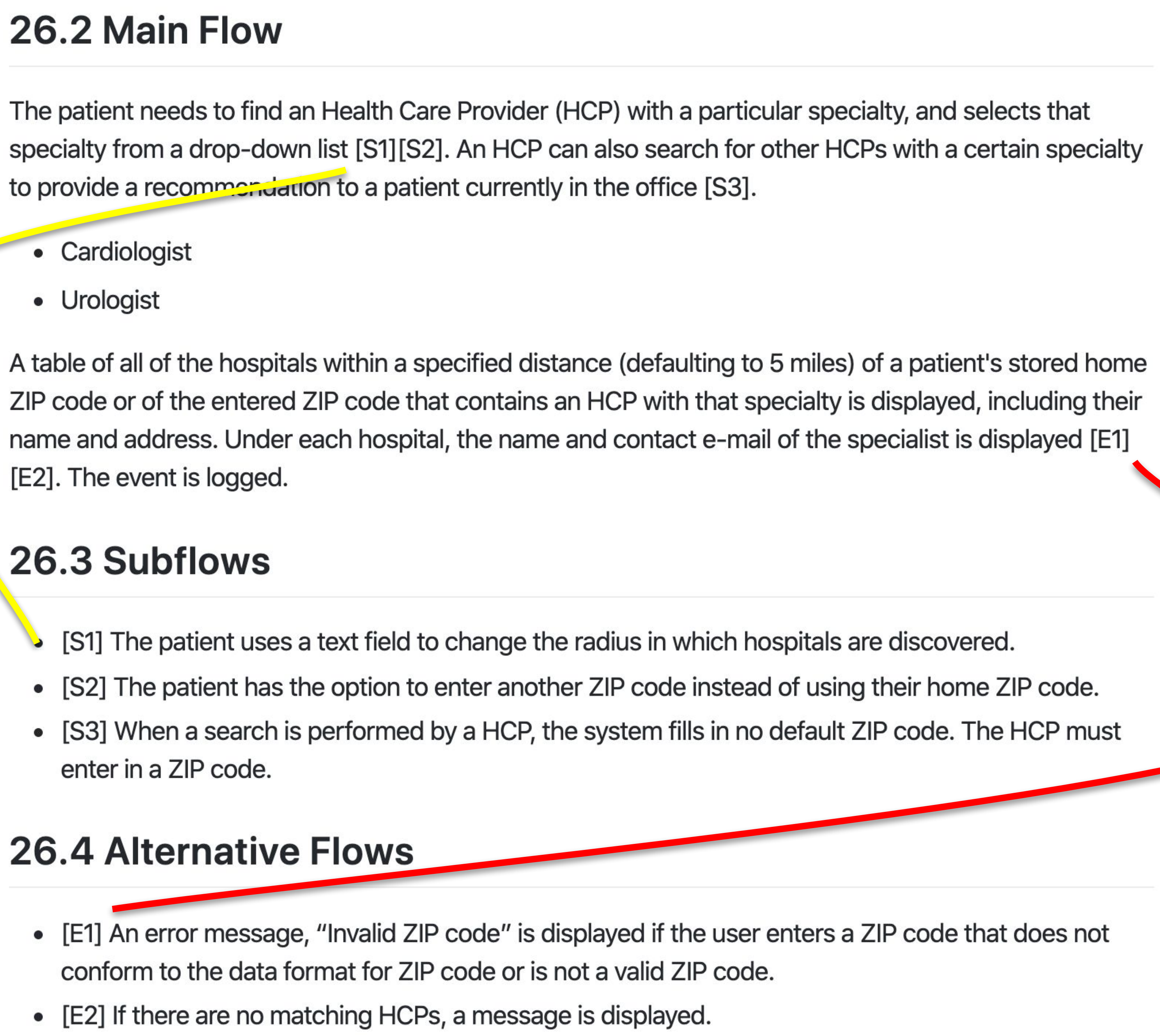
26.3 Subflows

- [S1] The patient uses a text field to change the radius in which hospitals are discovered.
- [S2] The patient has the option to enter another ZIP code instead of using their home ZIP code.
- [S3] When a search is performed by a HCP, the system fills in no default ZIP code. The HCP must enter in a ZIP code.

26.4 Alternative Flows

- [E1] An error message, “Invalid ZIP code” is displayed if the user enters a ZIP code that does not conform to the data format for ZIP code or is not a valid ZIP code.
- [E2] If there are no matching HCPs, a message is displayed.

[S1]



26.2 Main Flow

The patient needs to find an Health Care Provider (HCP) with a particular specialty, and selects that specialty from a drop-down list [S1][S2]. An HCP can also search for other HCPs with a certain specialty to provide a recommendation to a patient currently in the office [S3].

- Cardiologist
- Urologist

A table of all of the hospitals within a specified distance (defaulting to 5 miles) of a patient's stored home ZIP code or of the entered ZIP code that contains an HCP with that specialty is displayed, including their name and address. Under each hospital, the name and contact e-mail of the specialist is displayed [E1] [E2]. The event is logged.

26.3 Subflows

- [S1] The patient uses a text field to change the radius in which hospitals are discovered.
- [S2] The patient has the option to enter another ZIP code instead of using their home ZIP code.
- [S3] When a search is performed by a HCP, the system fills in no default ZIP code. The HCP must enter in a ZIP code.

26.4 Alternative Flows

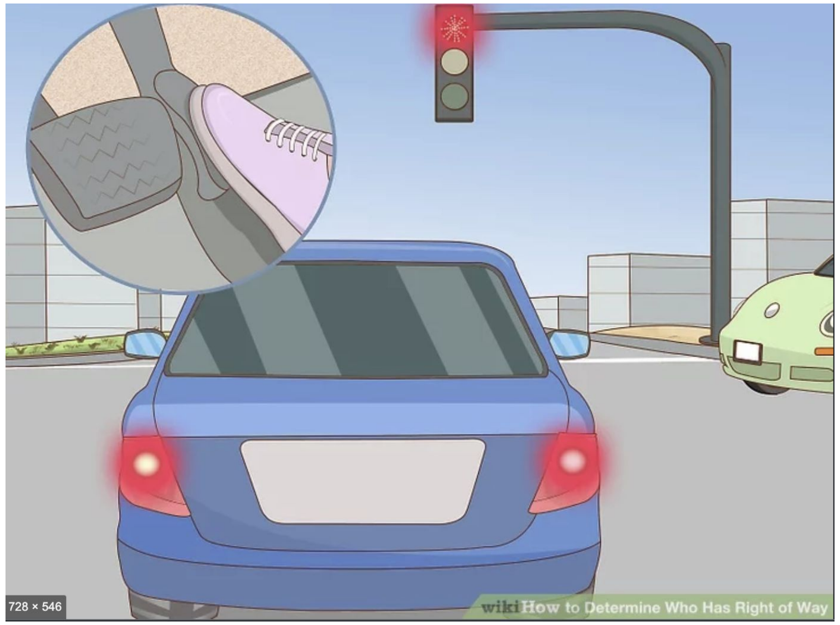
- [E1] An error message, “Invalid ZIP code” is displayed if the user enters a ZIP code that does not conform to the data format for ZIP code or is not a valid ZIP code.
- [E2] If there are no matching HCPs, a message is displayed.

[E1]

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.



1. 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].

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].

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].

Pinto revisited :*"Clear Intersection"*

Src: Top Secret! (1986)

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 gun it.
- **[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.

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!

User Stories

Assumption:
In the beginning, neither the customer nor the developer has full knowledge of requirements.

© NC State Software Engineering Faculty

Process:
Frequent, personal interactions with customers and/or stakeholders

Goal:
Frequent Delivery of software

Result:
Faster delivery of software that meets customer needs

Driver of Requirements

- Domain Expert
- Study the Market
- Talk to Customers
- Understand the Business
- Run Focus Groups
- Competitive Analysis
- Go to Trade Shows

- …. Also take suggestions from the development team, technical support, professional services, etc. and understand feasibility/limitations

© NC State Software Engineering Faculty

In other words: Agile processes are highly collaborative

Features ≣ User Stories

User Roles

- A user role represents a population of user types and their intended interactions with the system.
- Stories should not be written from a single perspective or stories will be forgotten.
- Roles in iTrust2
  - HCP – eventually could have specializations, different types, etc.
  - Patient – eventually could have dependents
  - Admin
  - ….

© NC State Software Engineering Faculty

Structure of an Epic or Story

- As an [X], I want to [Y] so I can [Z].
  - X = user role (who)
  - Y = desired functionality (what)
  - Z = benefit (why)

There's a new feature in CoffeeMaker – extending the types of ingredients that a customer can include in their recipes!

Create Recipes
As a ***customer***, I want to ***add new ingredients to create more custom drink recipes***, so I ***can enjoy a yummy beverage during work***.

Acceptance: Abi adds apple cider as an ingredient in the CoffeeMaker system.

© NC State Software Engineering Faculty

Writing a User Story

1 As an [X], I want to [Y] so I can [Z].

X = user role (who)
Y = desired functionality (what)
Z = benefit (why)

micro-story

2 Acceptance Criterion:

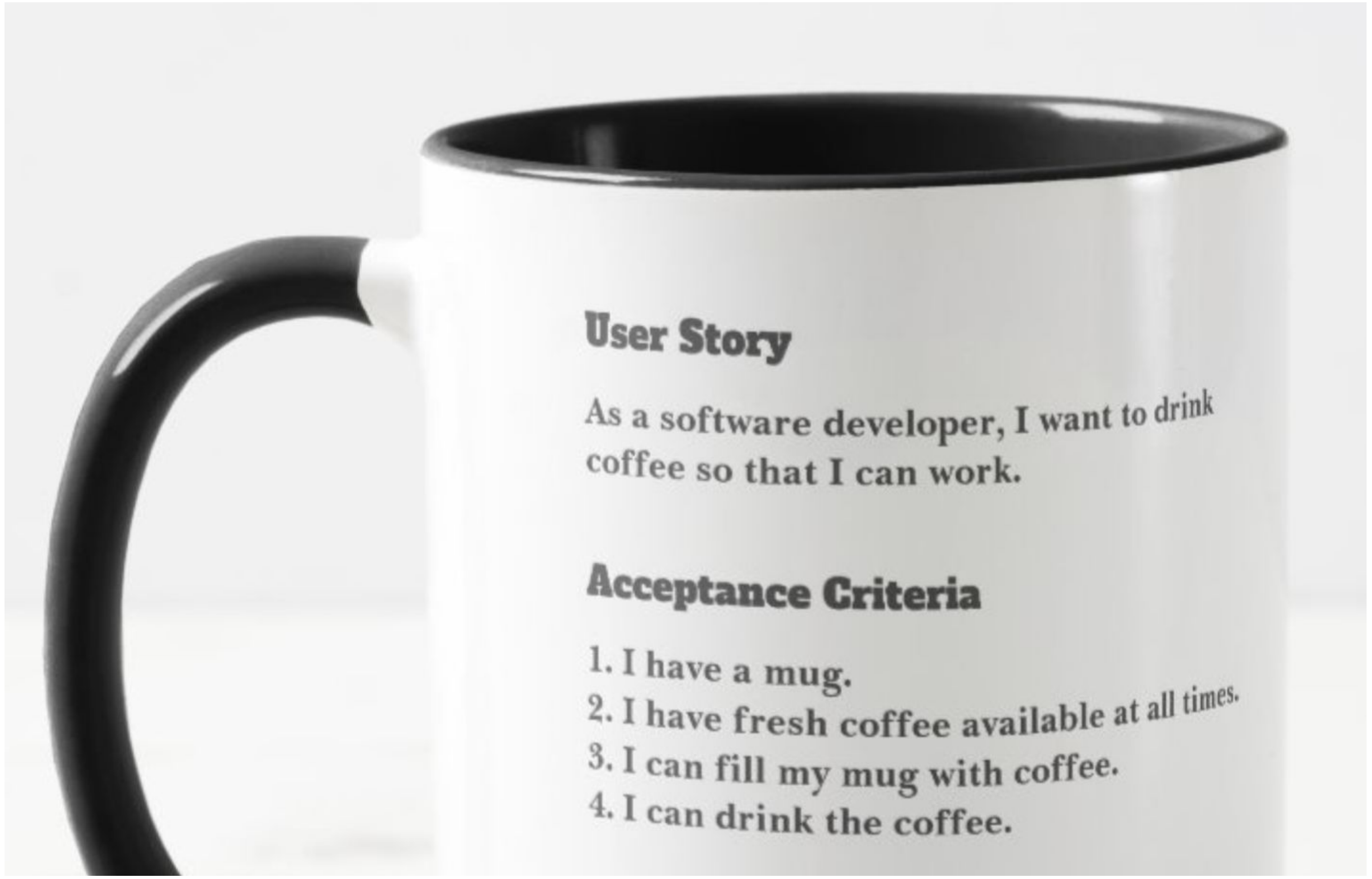
- ❏ A
- ❏ B
- ❏ C

$\}$ checklist ✓

They put it on a coffee mug...

1. micro-story

2. checklist ✓



User Story

As a software developer, I want to drink coffee so that I can work.

Acceptance Criteria

1. I have a mug.
2. I have fresh coffee available at all times.
3. I can fill my mug with coffee.
4. I can drink the coffee.

I am your customer

Coffee at WakeUp
As a user, when I wake up in the morning, I want my coffee to be ready.

Acceptance: alarm goes off, coffee is 190F

Clean the coffee machine
As a machine, I want to be cleaned within 30 minutes of brewing.

Acceptance: after brewing, ping the user to clean me until I've been cleaned

Coffee after Lunch
As a user, after I eat lunch, I want my coffee to be ready.

Acceptance: after lunch appt on calendar, coffee is 190F

Coffee after Morning Workout
As a user, after I work out in the morning, I want my coffee to be ready.

Acceptance: after my fitbit is done charging, coffee begins

Attributes of an epic or user story (INVEST)

- Independent
- Negotiable
- Valuable to purchasers or users
- Estimate-able
- Small*
- Testable

* Only applies to user stories

INVEST

- Independent: Avoid dependencies between stories in the same iteration. Dependencies lead to prioritization and planning problems and cascading failure risks.
- Negotiable: Details negotiated via conversation between the product owner and the development team.
- Valuable to purchasers or users: Make someone other than the development team happy by showing interim progress the team can get feedback on.

© NC State Software Engineering Faculty

Estimate-able

- What if the team cannot begin to estimate business scenario or user story because the technology or functionality is too new to them?
  - Run a spike: a brief "end-to-end executable experiment"
  - Perform a research task
- Purpose: To learn, to "buy information" so that you can estimate
- Spikes and research tasks become user stories
  - Timeboxed/bounded resource allocation
  - Accountability

INVEST

- Small: A user story must be small enough to meet all done criteria in one iteration (i.e. finish a “potentially shippable” feature)

- Testable: Demonstrating acceptance tests is a strong indication the functionality has been developed. Watch for “untestable” requirements being non-functional requirements or not specified well enough for the team to know what to do.

© NC State Software Engineering Faculty

Confirmation . . . The Acceptance Test

Setup: Jamie’s account balance is $100, Jamie’s ATM card is valid, machine has $5000.

Operation: Jamie inserts card, correctly keys PIN 0124, requests $20.

Verify: Machine dispenses $20, Jamie’s balance is $80, card is returned.

Non-functional requirements and constraints

- Non-functional requirements are requirements which are not specifically concerned with the functionality of a system but place restrictions on the product being developed.
    - Response time must be less than one second with up to 100 concurrent users.
    - Role-based access will be used to restrict the functionality available for each user role.
- Constraints are a type of non-functional requirement that restricts the implementation of the system or the development process. Constraints have no direct effect on the users' view of the system.
    - All graphing and charting will be done using a third-party library.
    - The software will be written in Java.
- Create a story for these as a reminder they may apply to all stories. If applicable to a story, the estimate for a story must consider the non-functional requirement and/or constraint.
    - Often placed in "Iteration 0" (discussed later)

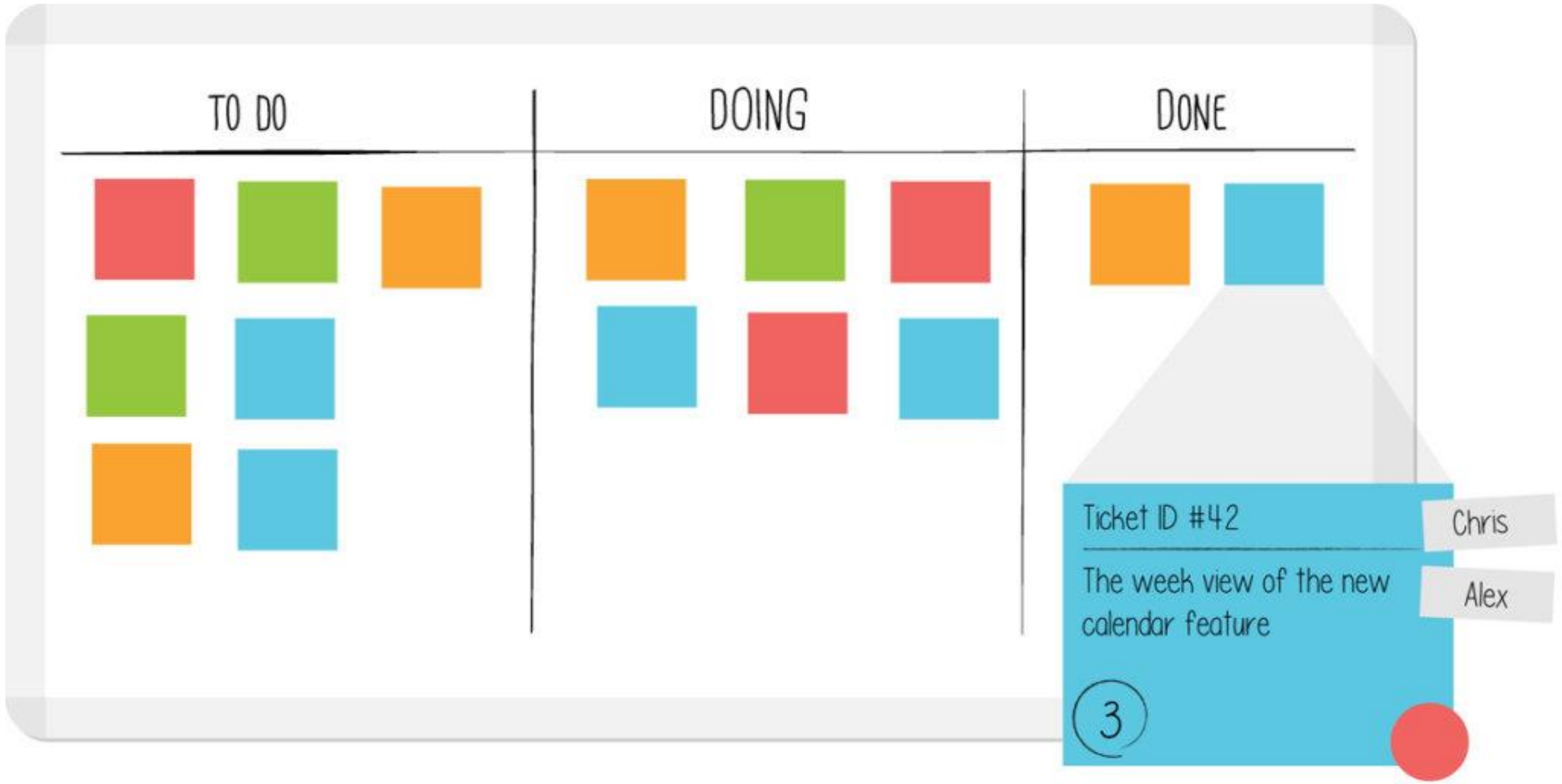
Non functional story examples

- As a customer, I want to be able to run your product on all versions of Windows from Windows XP on so that I don't have to upgrade all my systems.
- As the CTO, I want the system to use our existing orders database rather than create a new one so that we don't have one more database to maintain.
- As a user, I want the site to be available 99.999% of the time I try to access it so that I don't get frustrated and find another site to use.
- As a user, I want the driving directions to be the best 90% of the time and reasonable 99% of the time so I don't get lost.

© NC State Software Engineering Faculty

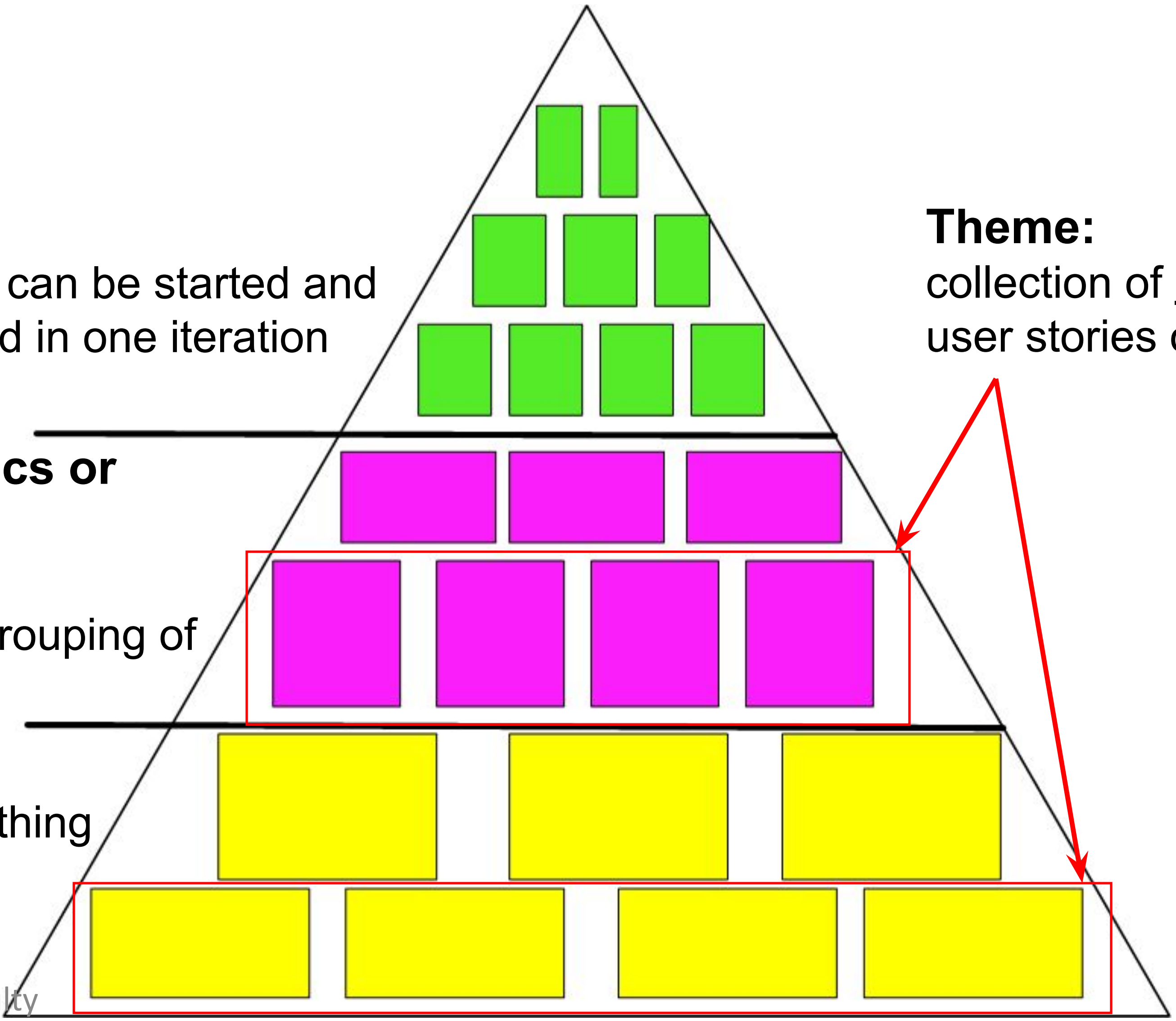
Adapted from http://blog.mountaingoatsoftware.com/non-functional-requirements-as-user-stories

Kanban Boards (Agile Planning)



https://leankit.com/learn/kanban/kanban-board/

Agile Requirements Granularity



Sprint:
Stories = can be started and completed in one iteration

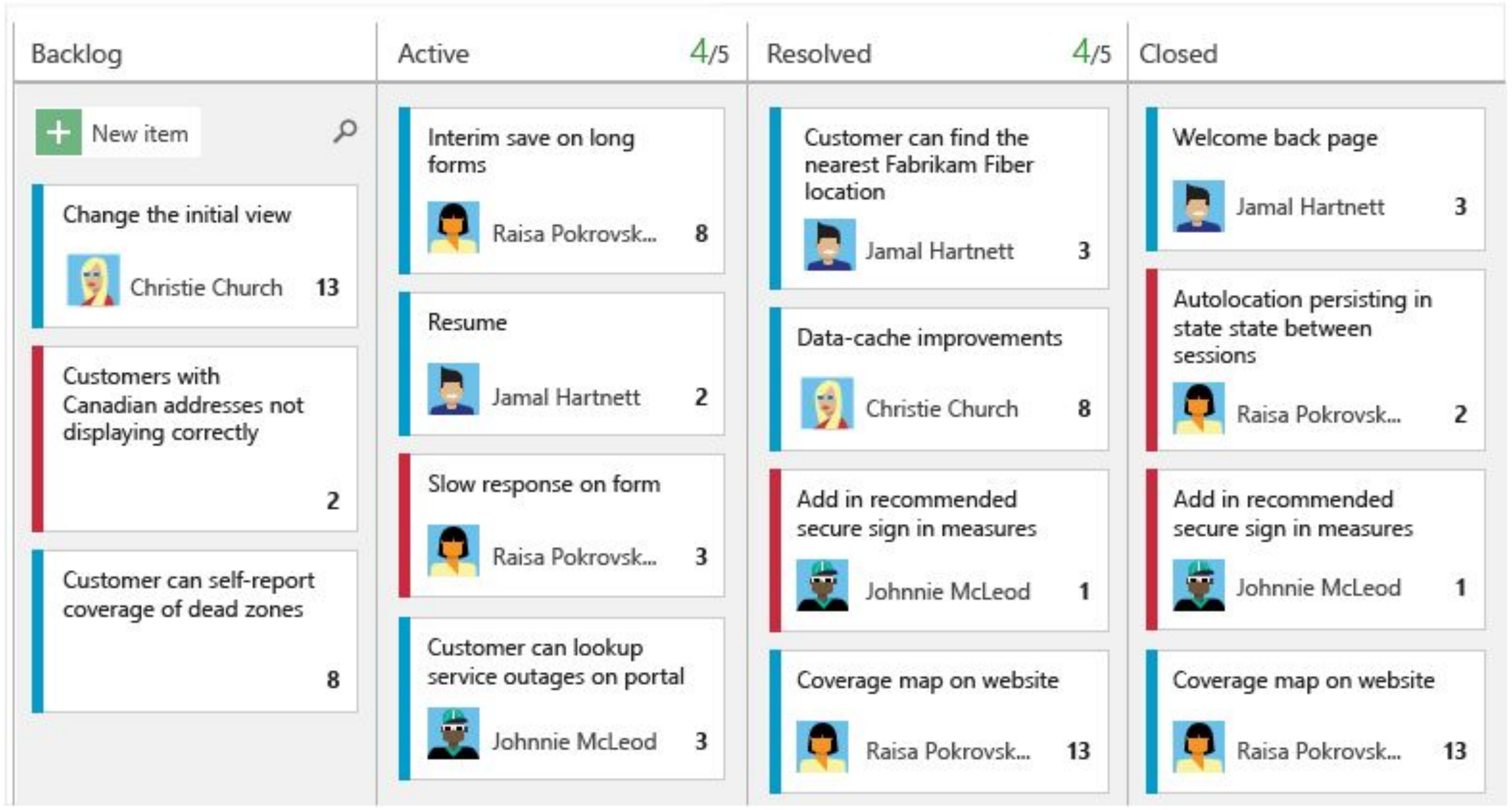
Theme:
collection of <u>related</u> user stories or epics

Release: epics or stories
Epic = larger requirement; grouping of stories

Product backlog
Project plan, everything the team plans to deliver/fix

© NC State Software Engineering Faculty

Kanban board in Visual Studio Team Services (VSTS)



Backlog

New item

Change the initial view
Christie Church 13

Customers with Canadian addresses not displaying correctly
2

Customer can self-report coverage of dead zones
8

Active 4/5

Interim save on long forms
Raisa Pokrovsk... 8

Resume
Jamal Hartnett 2

Slow response on form
Raisa Pokrovsk... 3

Customer can lookup service outages on portal
Johnnie McLeod 3

Resolved 4/5

Customer can find the nearest Fabrikam Fiber location
Jamal Hartnett 3

Data-cache improvements
Christie Church 8

Add in recommended secure sign in measures
Johnnie McLeod 1

Coverage map on website
Raisa Pokrovsk... 13

Closed

Welcome back page
Jamal Hartnett 3

Autolocation persisting in state state between sessions
Raisa Pokrovsk... 2

Add in recommended secure sign in measures
Johnnie McLeod 1

Coverage map on website
Raisa Pokrovsk... 13

Kanban Tools

1. GitHub Projects: https://github.com/features/project-management/
2. Trello
3. Jira
4. LeanKit
5. Visual Studio Team Services

Summary – User Stories

- Customer plays prime role in requirements
- Epics and stories are not “fully” documented requirements, but a token for further conversation between the development team and the customer.
  - The details of epics and stories emerge on a “just-in-time” basis during iteration planning.
- Epics are too large to finish within an iteration.
- Stories can be completed in one iteration but must still be useful to a user or purchaser (i.e. are not technical tasks)

Use Case vs. User Story

- Use cases written by developers
  - Fully documented, archived
  - Read by developers
  - Read by stakeholders
  - Read by customers
- User stories are written by customers (maybe with developer assistance)
  - Low ceremony
  - Read by developers
  - Read by stakeholders
  - Read by customers

Overall Summary

⟵ *Requirements Process* ⟶

more formal *less formal*

- Use Cases and User Stories are ways to model and document requirements
- Traditional Requirements ⟵ Very Formal, used in safety critical domains.
- Use cases ⟵ More Formal, used in a variety of domains, including safety critical.
- User stories ⟵ Less Formal, used in fast-to-market domain, part of *Agile.*

© NC State Software Engineering Faculty

Important Takeaways

- It’s **least costly** to identify issues in requirements, compared to other phases

- **Elicitation** involves understanding what the customer wants and why

- **Modeling** involves translating customer requirements to software requirements

© NC State Software Engineering Faculty