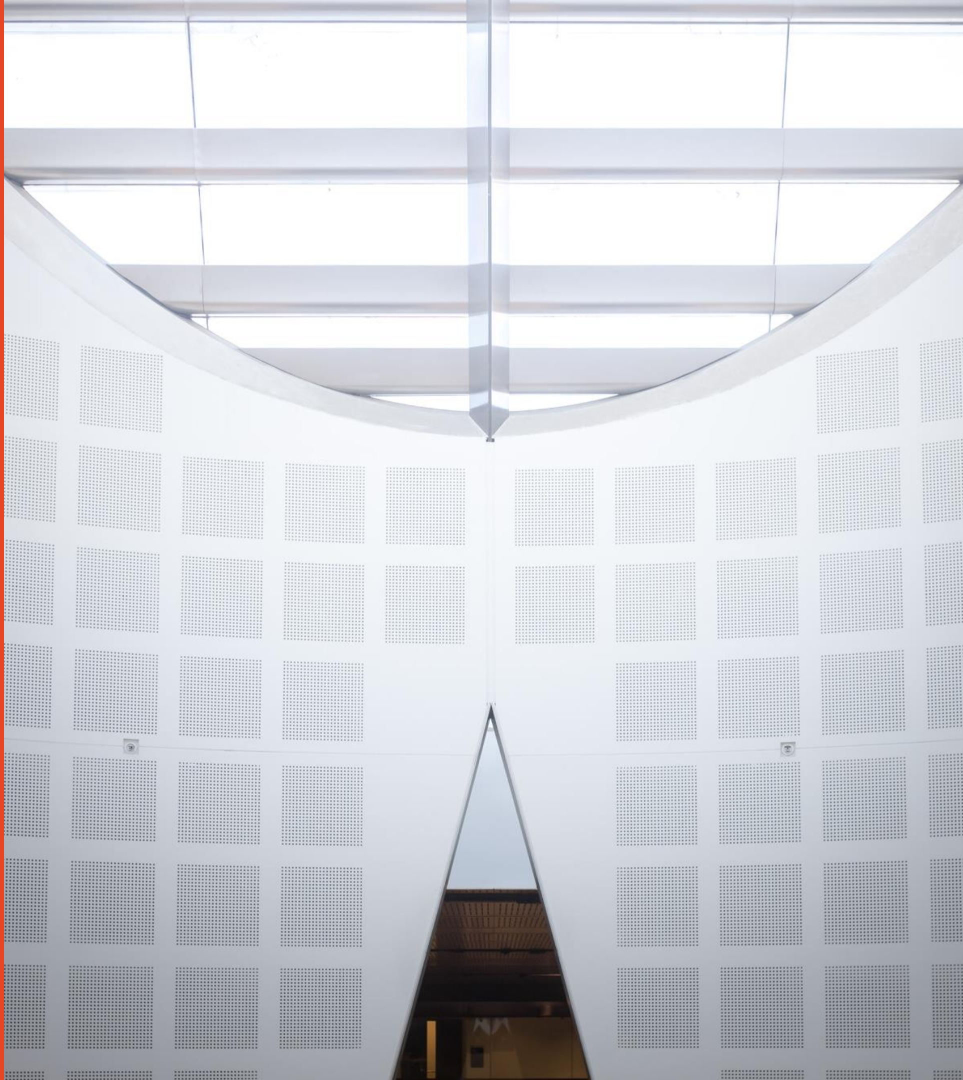
Agile Software Development Practices (SOFT2412/COMP9412)

Requirements; Technologies for Expressing Requirements

Dr. Basem Suleiman

School of Computer Science

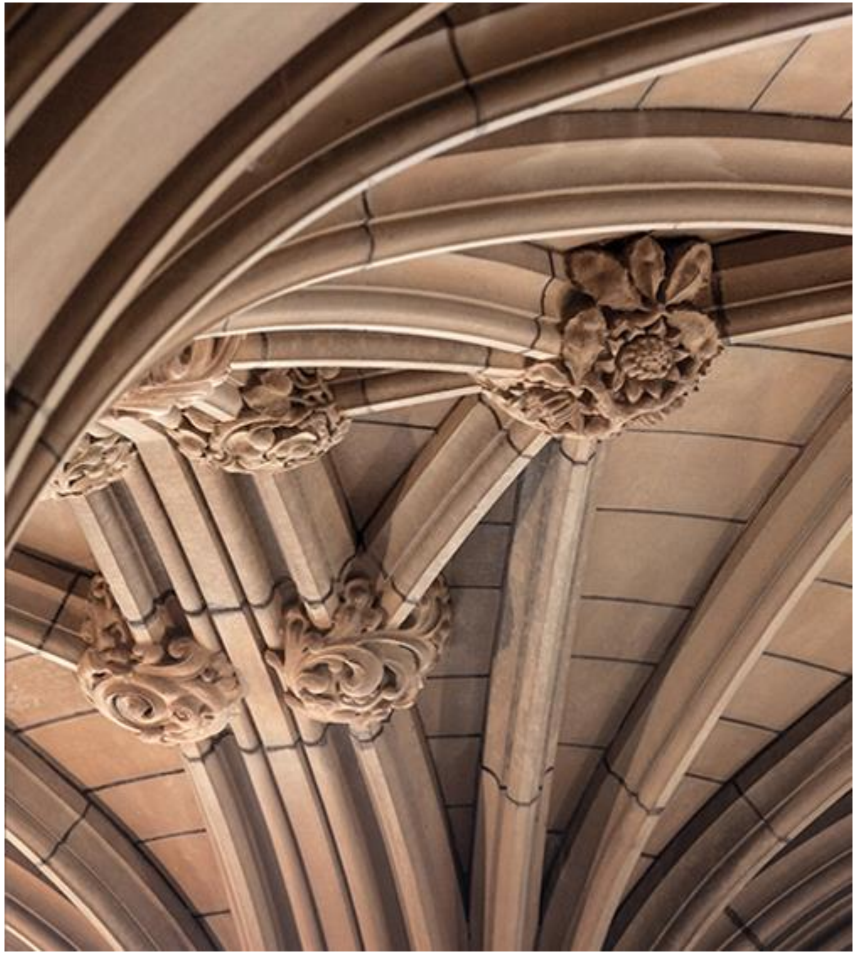
THE UNIVERSITY OF SYDNEY



Agenda

- Plan-Driven Software Development
  - Requirements Engineering
- Agile Software Development
  - Behavior-Driven Development
  - User Stories
  - User Interfaces
  - Scenarios
  - Storyboards
- Tools for Agile Software Development

Requirements in Plan-Driven Software Development



THE UNIVERSITY OF SYDNEY

Plan-and-Document Software Methodologies – Revisit

- Requirements, Analysis, Design, Code & Integrate, Test/QA, Deploy/Operate/Maintain

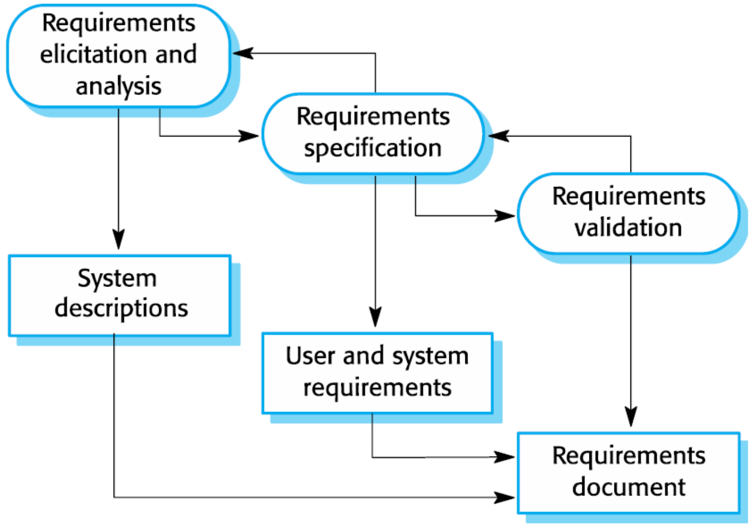
- Goal is to make Software Engineering predicable in budget and schedule
  - Requirements elicitation
  - Requirements documentation
  - Cost estimation
  - Scheduling and monitoring schedule
  - Change management for requirements, cost and schedule
  - Ensuring implementation matches requirement features
  - Risk analysis and management

Plan-and-document – Requirements

- In traditional methodologies, the requirements document is usually very detailed and extensive
- Often done in Word, to a detailed template
- The requirements are named and numbered, to support traceability
- See IEEE Standard 29148 for comprehensive details

http://mmf.nsu.ru/sites/default/files/iso-iec-ieee-29148-2011.pdf

Plan-and-Document – Requirements Engineering Process


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

Requirements engineering
main activities and deliverables

Ian Sommerville. 2016. Software Engineering (10th ed. Global Edition). Pearson

Requirements Elicitation – Techniques

- Interviewing stakeholders
  - Information discussions and/or formal questions

- Cooperatively create scenarios
  - Initial state, happy and sad paths, concurrent and final states

- Create Use Cases
  - User and system interactions to realize functions (using UML case diagrams)

Requirements Elicitation

- Functional
  - Details matter: exactly what information goes in and out
  - Interactions between features
  - Not only when things go well

- Non-functional
  - Include performance, security, usability

- Note that different stakeholders often have different thoughts on this

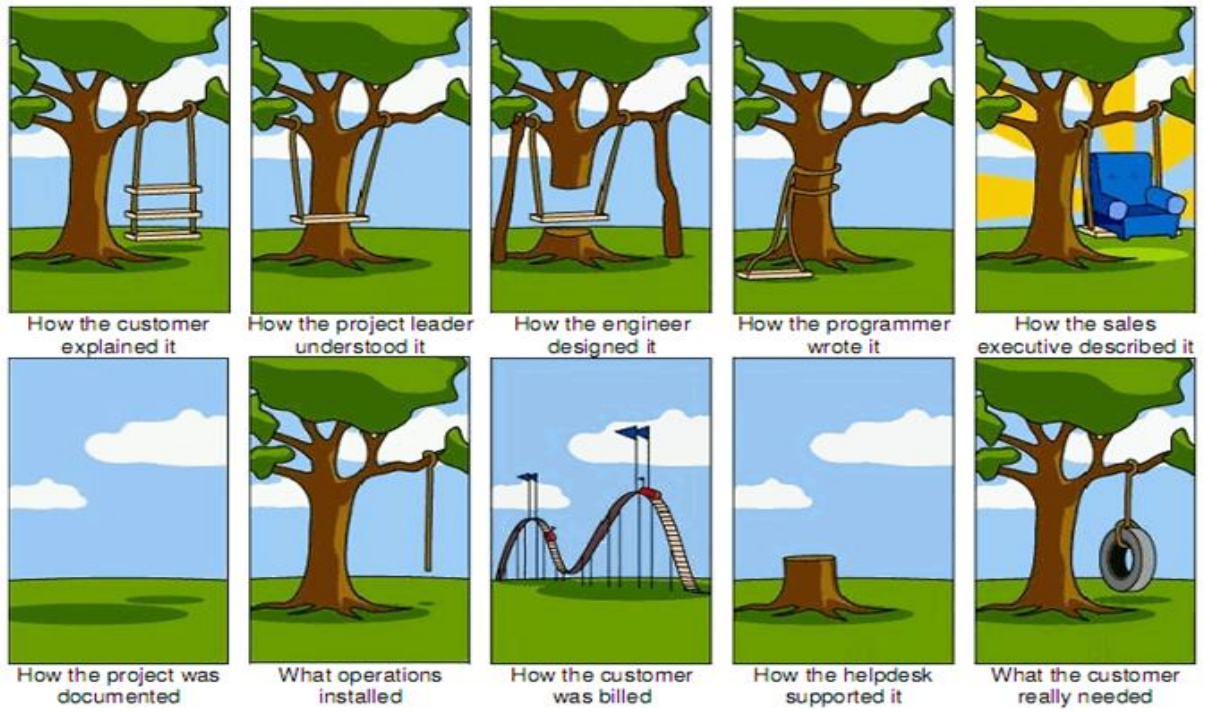
Requirements Documentation

- Software Requirements Specifications (SRS) process
  - 100s of pages, IEEE 830-1998 standard recommended practice for SRS

- Stakeholders to read SRS document, build basic prototype, or generate test cases to check:
  - Validity
  - Consistency
  - Completeness
  - Feasibility

- Estimate budget and schedule based on the SRS

Software Requirements Mystery


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

https://cdn-images-1.medium.com/max/1600/1*xzS-UkYtNOgzPvpkHGwRbQ.png

Why Software Projects Fail?

- Over-budget, over-time
- Hard to maintain and evolve

- **Useless (unwanted) product features**
  - Project teams felt that *many features in the software they built were not used*
    - 45% of features never used, 19% rarely used
  - Development teams would build software, and *throw it over the wall* to their users, and hope some of what they build would stick

https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

Requirements in Agile Software Development

THE UNIVERSITY OF SYDNEY

Agile Manifesto – Revisit

- “We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value”
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

- The items on the left are more valued than those at he right

Agile Manifesto: http://agilemanifesto.org/

Agile Manifesto – Revisit

- “We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value”

  - Individuals and interactions over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan

- The items on the left are more valued than those at he right

Agile Manifesto: http://agilemanifesto.org/

Agile Principles –Requirements

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	9. Continuous attention to technical excellence and good design enhances agility.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	**10. Simplicity--the art of maximizing the amount of work not done--is essential.**
3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**	7. Working software is the primary measure of progress.	**11. The best architectures, requirements, and designs emerge from self-organizing teams.**
4. Businesspeople and developers must work together daily throughout the project.	8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Alliance: http://www.agilealliance.org

Requirements in Agile Software Development

- Work continuously with stakeholders to develop requirements and tests
  - Stakeholders: any party that has interest in the product/software being build (e.g., customers, end users, domain experts, sales, government agencies)

- Iterative development
  - Short iterations 2-4 weeks each focused on core features
  - Maintain working prototype while adding new features
  - Check with stakeholders what's next to validate building the right software

Requirements in Agile Development

- No "standard way" to do requirements in Agile development
  - Normal lightweight "Software Requirements Document"
  - Based on user stories
  - Iteratively elaborate mall set of the functional requirements (priority-based)
  - Create some acceptance tests at the same time as you write the requirements

Behaviour Driven Development (BDD)

User Stories

Behavior-Driven Development (BDD)

- A conceptual approach for specifying application's behaviour and communicating them clearly among people
- Business value (feature) → acceptance criteria → code to deliver it
- *Given-When-Then* canvas
  - Improves communication among domain experts, users, testers and developers

https://www.agilealliance.org/glossary/bdd

Requirements – BDD

- User stories to elicit functional requirements
- Low Fidelity User Interfaces (UIs) and Storyboards to elicit UIs
- Transfer user stories into acceptance tests

BDD – User Stories

- Short and concise description of how the application is expected to be used from the user's point of view
  - Functionality that has value to the user and/or customer
  - We need feature X but we do not have enough details about it?
  - We got more details about feature Y?
- Helps stakeholders to plan and prioritize development
- Improve requirements clarity
- Documents functional requirements as executable scenarios/examples

https://www.agilealliance.org/glossary/bdd

Agile Development – User Stories

- A common way to express very high-level requirements
- Almost universal in Scrum and other agile methods
- A short piece of text (can be written on a small piece of *cardboard*)

User Stories

- Borrowed from Human Computer Interface (HCI) community
  - "3-by-5 cards" (3x5 inch index cards)
  - 1-3 non-technical sentences written jointly by the stakeholders and developers
  - Small enough to implement in one iteration, testable and must have business value

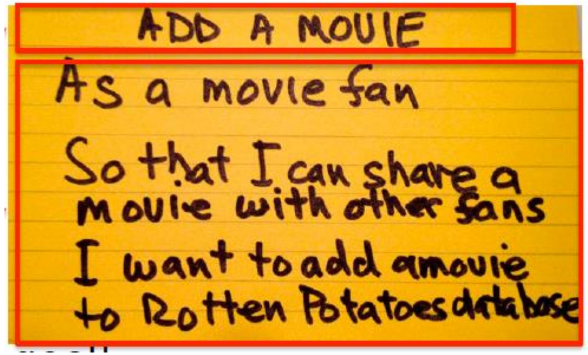
- *Connextra* format:

  As a [stakeholder], I want a [feature], so that [benefit]

User Stories – Examples

Feature
Kind
Goal
Task

1 *Feature Name* F
2 *As a [Kind of Stakeholder],* K
3 *So that [I can achieve some goal],* G
4 *I want to [do some task]* T



ADD A MOVIE

As a movie fan

So that I can share a movie with other fans

I want to add a movie to Rotten Potatoes database

1 *Feature: Add a movie to RottenPotatoses*
2 *As a movie fan*
3 *So that I can share movie with other movie fans*
4 *I want to add a movie to RottenPotatoses database*

User Stories – Why?

– User stories help to build software features that are likely needed

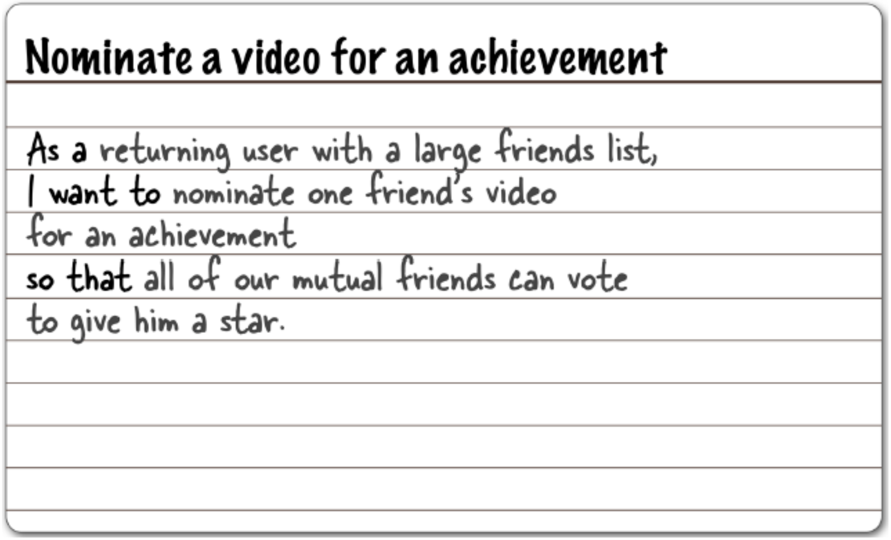
  – *CHAOS report; 45% and 19% of features in the software were never and rarely used respectively*

Effective as it specifies:

– Who the user is

– What the user wants to do

– Why the user wants to do it



Nominate a video for an achievement

As a returning user with a large friends list,
I want to nominate one friend's video
for an achievement
so that all of our mutual friends can vote
to give him a star.

User Stories – Common Mistakes

- Generic user, rather than meaningful role
- No evident business value
  - Less technical details (jargons)
  - E.g., "wouldn't it be cool if…."
- How to write good user stories? ??

SMART User Stories

S – **Specific**
  – Describe specific details that add value

M – **Measurable**
  – User story should be testable; there are known expected results for some good inputs

A – **Achievable**
  – Can be implemented in one Agile iteration
  – Subdivide big stories (features) into smaller ones

SMART User Stories

R – **Relevant**

- Must have a *business value* relevant to one or more stakeholders
- Use "*Five Whys*" technique to help drilling down to uncover real business value

T – **Timeboxed**

- To estimate amount of time to implement it
- Stop developing the story if allocated time is over
  - Divide it into smaller ones or reschedule what's left to new estimate
  - If dividing won't help discuss with the customer the part of highest value

SMART User Stories – Relevant (Five Whys)

– **Group Exercise**: apply the "five whys" technique to uncover real business value/need in the following feature/story

Add a Facebook Linking feature to a Ticket-Selling application

SMART User Stories – Examples

- **Group Exercise**: apply ~~the~~ "five whys" technique to uncover real business value/need in the following feature/story

Add a Facebook Linking feature to a Ticket-Selling application

- Why add the Facebook feature? I think more people will go with friends and enjoy the show more
- Why does it matter if they enjoy the show more? I think we will sell more tickets
- Why do you want to sell more tickets? Because the theatre make more money
- Why does theatre need to make more money? So that we do not go out of business
- Why does it matter that theatre is in business next year? If not, I have no job

User Stories – “Done”

– When we can say the following user story is “Done”?

1 *Feature: Add a movie to RottenPotatoses*
2 *As a movie fan*
3 *So that I can share movie with other movie fans*
4 *I want to add a movie to RottenPotatoses database*

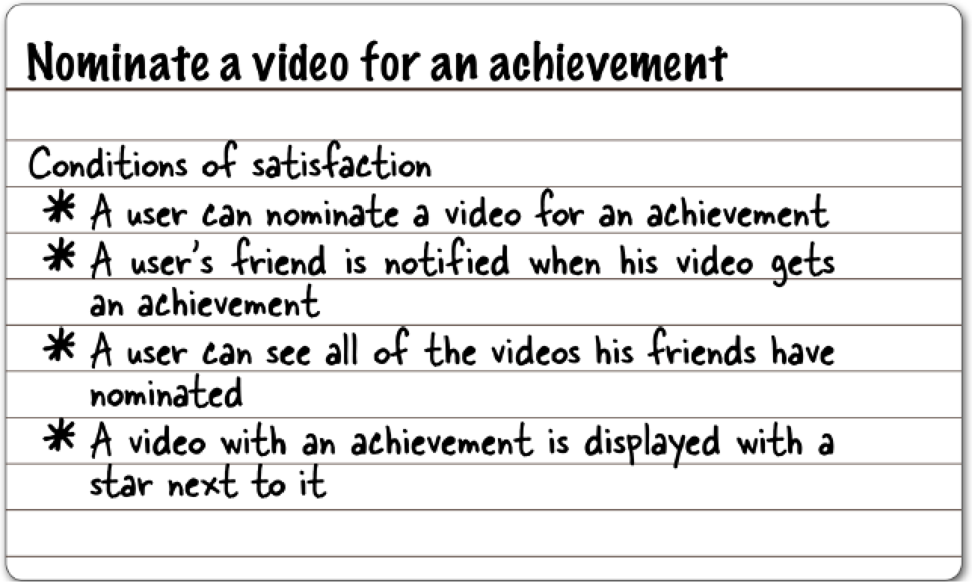
When all of the relevant criteria defined in the story is met!

User Stories – Acceptance Criteria

- Effective way for developers to gauge completion/satisfaction of features
  - Concrete definition of “Done” or “Completed”
  - aka *Condition of Satisfaction*

- Condition of Satisfaction to be written at the back of the user story index card

User Stories – Acceptance Criteria Example


Nominate a video for an achievement

Conditions of satisfaction

* A user can nominate a video for an achievement
* A user's friend is notified when his video gets an achievement
* A user can see all of the videos his friends have nominated
* A video with an achievement is displayed with a star next to it

Scrum Artifacts – Product Backlog

– Features and sub-features (items) needed to build the product (the "Plan" for multiple iterations)

  – User stories (features, functions), enhancements and fixes identified from previous Sprints

– Maintained by the Product Owner in collaboration with customers and team

– The source of the product requirements

  – The items ordered by priority – value to the customer

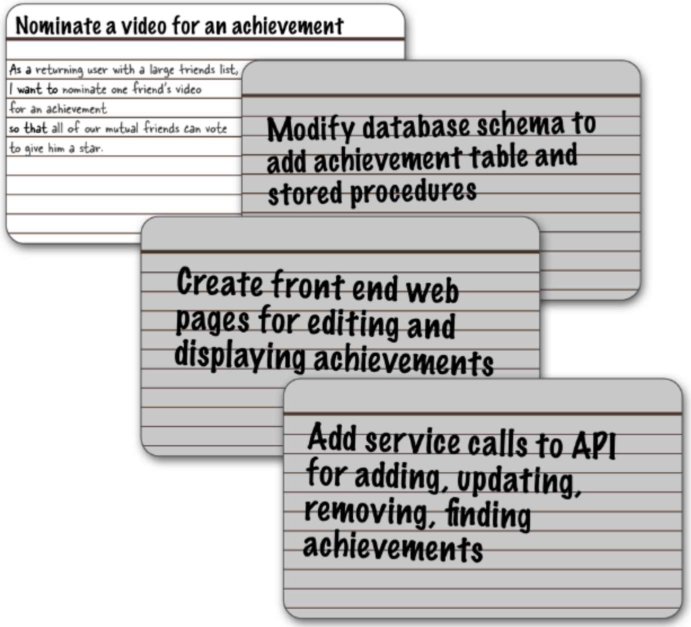
Scrum Events – Sprint Planning (Revisit)

- Identify the Sprint Goal (items from the "Product Backlog")
- Identify work to be done to deliver this
- Two-parts meeting (SM, PO and Dev team)
  - **Before meeting:** PO prepares prioritized list of most valuable items
  - **Part 1** *(max. 4 hours)*: PO & Dev team select items to be delivered at the end of the sprint (value-based) and on the team's estimate of effort
  - **Part 2** *(max. 4 hours)*: Dev team (with the PO's help) figure out the individual tasks they'll use to implement those items
- *Output*: Sprint Backlog (the items selected by the team for development)

Scrum – User Stories

Sprint Planning Meeting (part2):

1. Break the defined stories down into tasks
   - By team members
   - Each task on a separate card
   - Uncompleted stories - put on a single card to plan out the story



Nominate a video for an achievement

As a returning user with a large friends list, I want to nominate one friend's video for an achievement so that all of our mutual friends can vote to give him a star.

Modify database schema to add achievement table and stored procedures

Create front end web pages for editing and displaying achievements

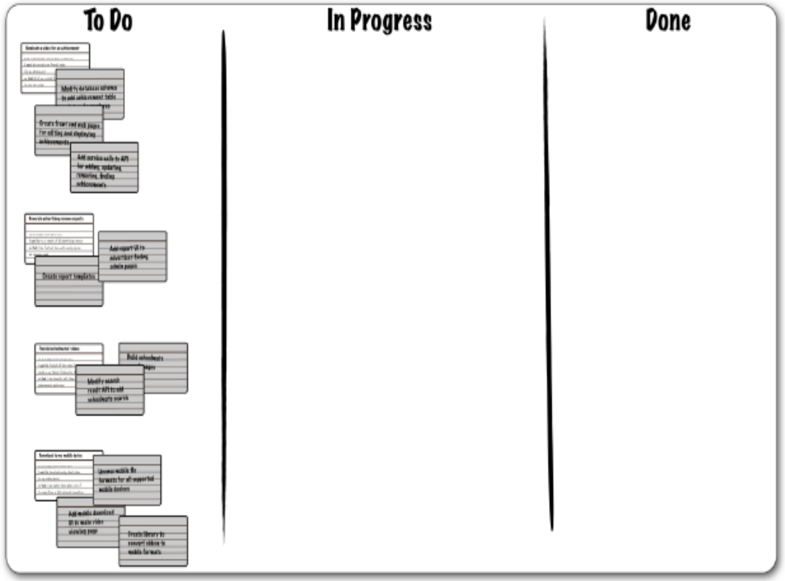
Add service calls to API for adding, updating, removing, finding achievements

Planning and Running a Sprint

stories and tasks

Sprint Planning Meeting (part 2):

2. Group the stories and their tasks together
   - Add them to the "*To-Do*" of the ***Sprint Backlog*** ***(Task Board)***



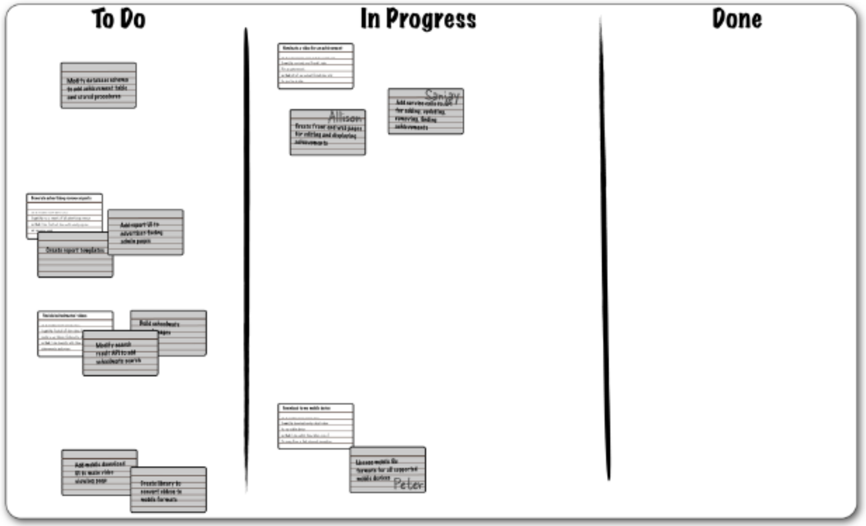
Planning and Running a Sprint

During the Sprint (development)

3. Team members to work on the tasks

- Each team member works on one task at a time by moving it to the "*In-Progress*" (under their name)

- Team members can add additional tasks to the board to finish a story
  - Communicate it in the *Daily Scrum*

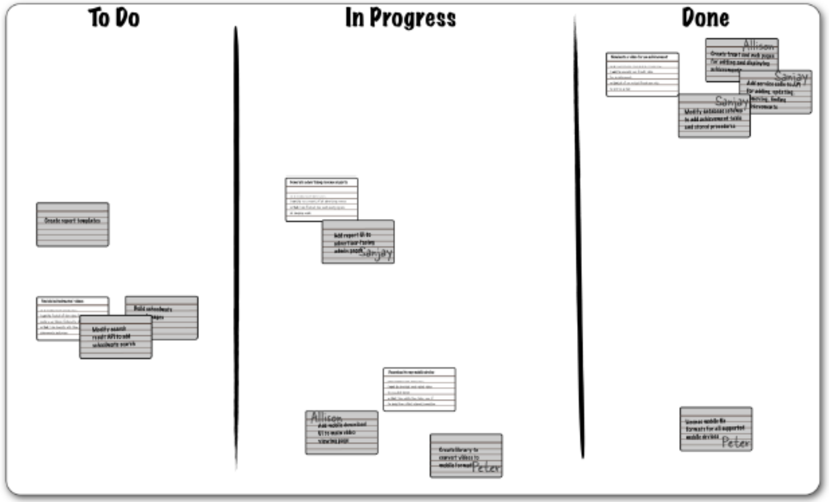


Planning and Running a Sprint

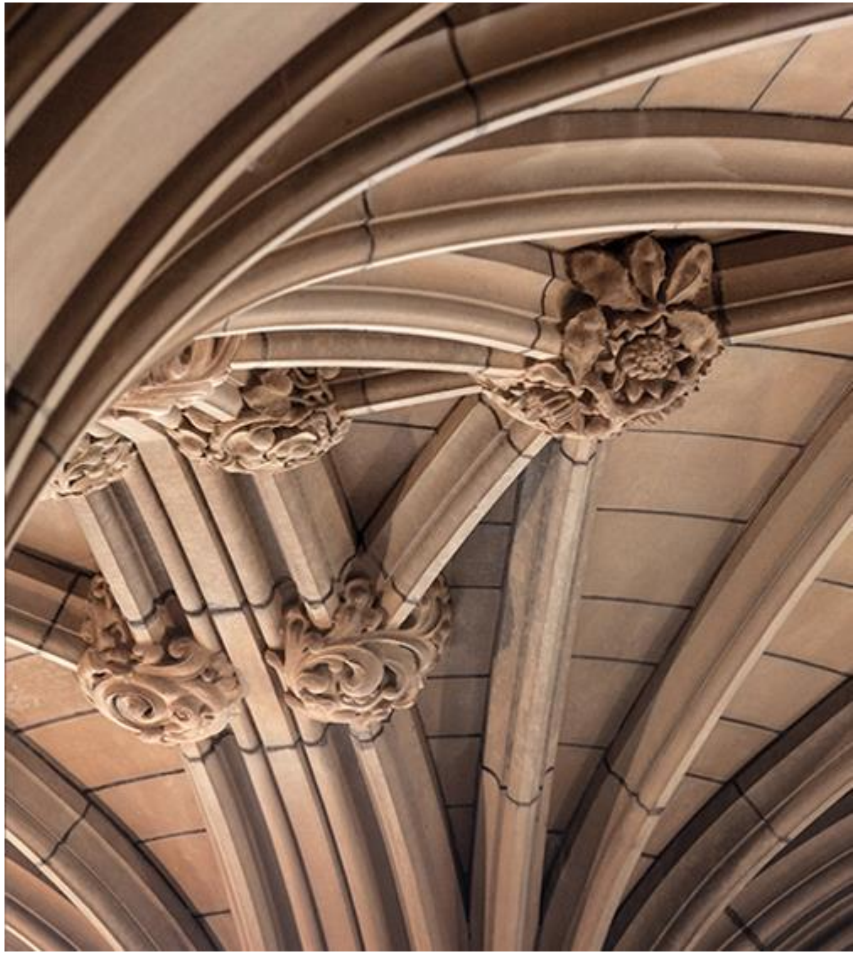
During the Sprint (development)

4. Finishing a task/story

- Task completed → "*Done*" and pulls another task
- The team member to finish the story's final task verifies that all the conditions of satisfaction are completed with the PO and move it to the "*Done*"



User Interfaces, Scenarios and Storyboard

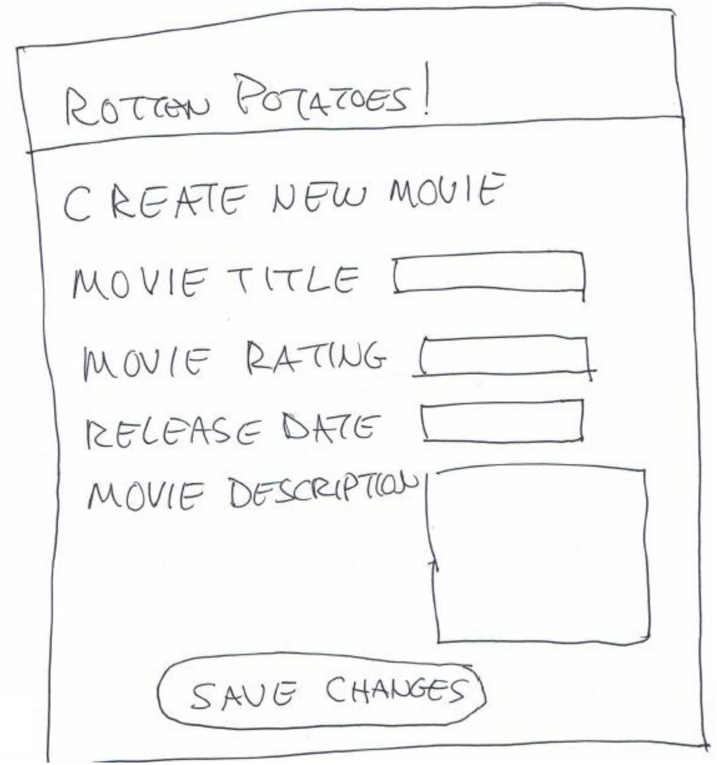


User Interface (UI) Sketches

- **Low-Fidelity (Lo-Fi) UIs**
  - Rough (UI) sketch or mock-up of a user story
  - Low-tech approach to UIs and the paper prototype sketches
    - Pencil-and-paper
  - Shows how a UI looks like and how sketches work together as a user interacts
  - Effective for engaging nontechnical stakeholders

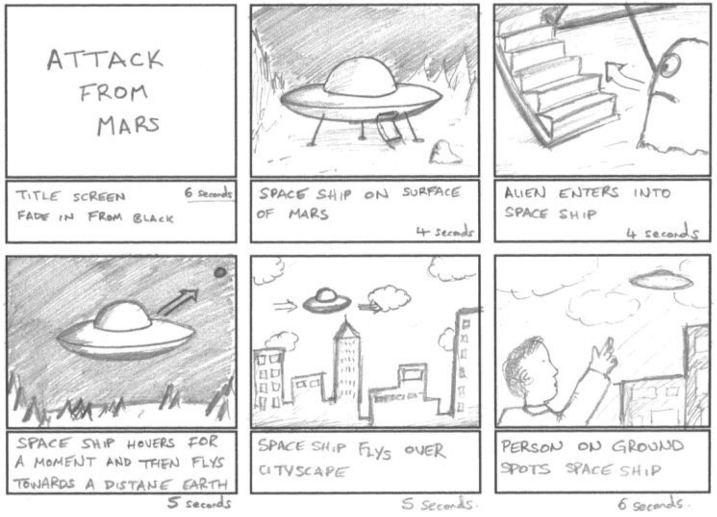
- **High-Fidelity (Hi-Fi) UIs**
  - Higher level of details that more closely matches the design of the actual UI (also used for documentation)

Lo-Fi Example – Add Movie



Scenarios and Storyboards

- Originated in filmmaking industry
- ***Scenario:*** written description of the system interactions from user's perspectives
  - Few steps (typically 3 to 8 steps)
- ***Storyboard:*** similar to scenario but it visualizes the interactions
- **User story** refer to a single feature
  - A feature usually has one or more scenarios that show different ways the feature is used


ATTACK FROM MARS
TITLE SCREEN FADE IN FROM BLACK
6 seconds
SPACE SHIP ON SURFACE OF MARS
4 seconds
ALIEN ENTERS INTO SPACE SHIP
4 seconds
SPACE SHIP HOVERS FOR A MOMENT AND THEN FLYS TOWARDS A DISTANE EARTH
5 seconds
SPACE SHIP FLYS OVER CITYSCAPE
5 seconds
PERSON ON GROUND SPOTS SPACE SHIP
6 seconds

https://www.mrstruitt.net/brainstormscriptstoryboard.html

Scenarios Format

Scenario: brief description of the scenario
GIVEN: description of context info. or pre-condition
WHEN: description of the action/event
THEN: description of the outcome

Simplified format of a scenario structure using GIVEN, WHEN and THEN phrases

S
G
W
t

Scenarios Format

Scenario: brief description of the scenario
GIVEN: description of context info. or pre-condition
WHEN: description of the action/event
THEN: description of the outcome

Simplified format of a scenario structure using GIVEN, WHEN and THEN phrases

Scenario: brief description of the scenario
GIVEN: description of context info. or pre-condition
AND: more context info. or pre-condition
WHEN: description of the action/event
AND: more action/event
THEN: description of the outcome
AND: more outcome
AND: more outcome

More complicated version of a scenario structure AND Multiple pre-conditions, actions/events and outcomes could be combined

Scenario Example – Add a Movie

Feature: User can manually add a movie

Scenario: Add a movie

GIVEN: I am on the RottenPotatoes home page

WHEN: I follow “Add new movie”

THEN: I should be at the “Create New Movie” page

WHEN: I fill in “Title” with “Men in Black”

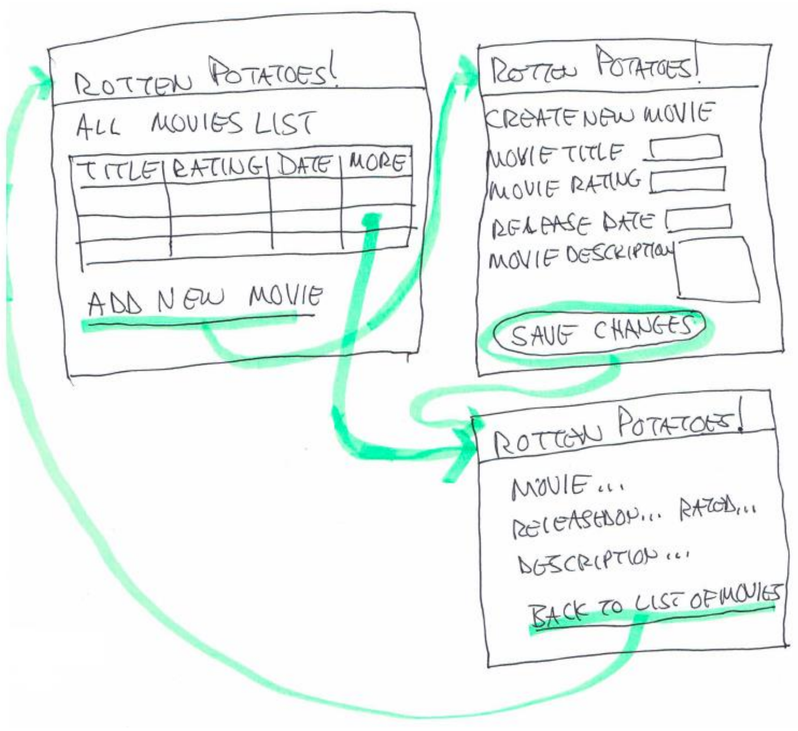
AND: I select “PG-13” from “Rating”

AND: I press “Save Changes”

THEN: I should be on the RottenPotatoes home page

AND: I should see “Men in Black”

Storyboard Example – Add New Movie



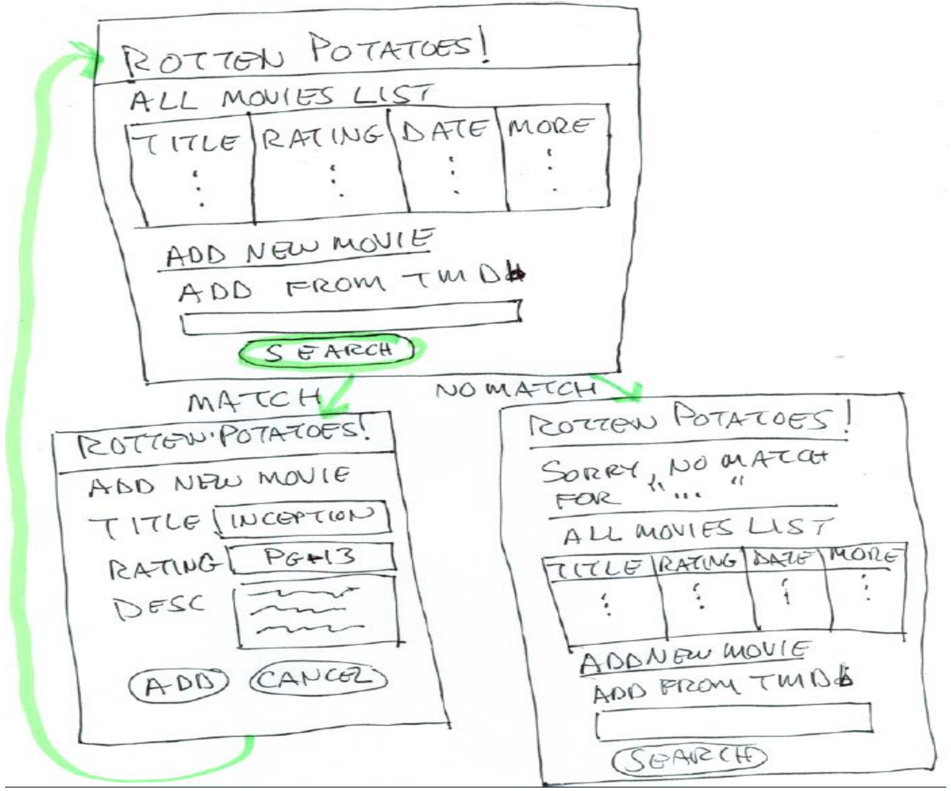
Scenario Example – Search for a Movie

- Search “The Open Movie Database” (TMDb) to find information about a movie we are interested in adding to RottenPotatoes
  - TMDb Web Service that has an API allow to access its information

- Develop 2 scenarios with corresponding Lo-Fi Uis that integrate RottenPotatoes with TMDb

- Generate automated test cases for the two scenarios

Searching the Movie Database – UI Storyboard



Searching the Movie Database – Scenario

Feature: User can add a movie by searching for it in the Movie Database TMDb

As a movie fan **so that I can** add new movies without manual tedium **I want to** add movies by looking up their details in TMDb

Background: Start from the Search form on the home page
GIVEN: I am on the RottenPotatoes home page
THEN: I should see "Add From TMDb"

Scenario: Try to add existing movie (happy path)

WHEN: I fill in Search terms with "Inception"
AND: I press "Search"

….
THEN: I should be on the "Add New Movie" page
AND: I should not see "Not found"
AND: I should see the "Inception" movie details

Scenario: Try to add non-existent movie (sad path)
WHEN: I fill in Search terms with "Movie that does not exist"
AND: I press "Search"
THEN: I should be on the RottenPotatoes home page
AND: I should see "Sorry No match for Movie that does not exist"

Implicit vs. Explicit Scenarios

- **Explicit requirements** explicitly specified as user stories developed by stakeholders in BDD
- **Implicit requirements** are not specified in the user stories
- Explicit requirements correspond to acceptance tests and implicit requirements correspond to integration tests
- Tools, such as *cucumber*, allows creating both acceptance and integration tests if user stories are written for both explicit and implicit requirements

2)

Imperative Scenarios

- Tend to have complicated WHEN statements and lots of AND steps
  - Ensures that UIs details match customer expectations, e.g., filling in a form
  - Tedious to write such scenarios and not a good practice

- Scenarios should rather focus on the application's behaviour

3)

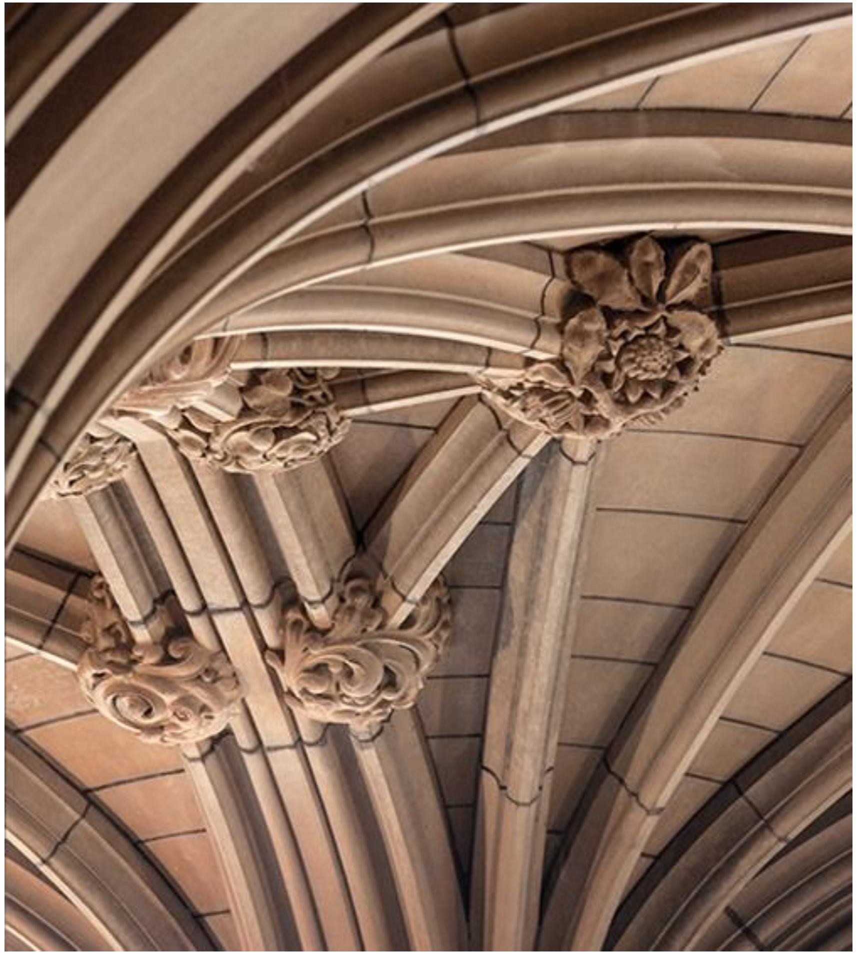
Declarative Scenarios

- Focuses on the feature being described by using the step definitions to make a domain language for the application
  - A domain language is informal but uses terms and concepts specific to your application rather than generic terms and concepts specific to the UI
- By experience, user stories should be written in a domain language that you will have to develop via your step definitions for the application you build

informal language

Tools for Requirements in Agile Development

User Stories, Backlog, Sprint Backlog



Automated Tests

- In BDD requirements and tests are combined together
- Some tools, e.g., *Cucumber*, *JBehave*, convert written scenarios to automated tests
- **Acceptance Tests**
  - To ensure the customer/user is satisfied with the application behaviour
- **Integration Tests**
  - To ensure that the interfaces between modules have consistent assumptions and communicate correctly

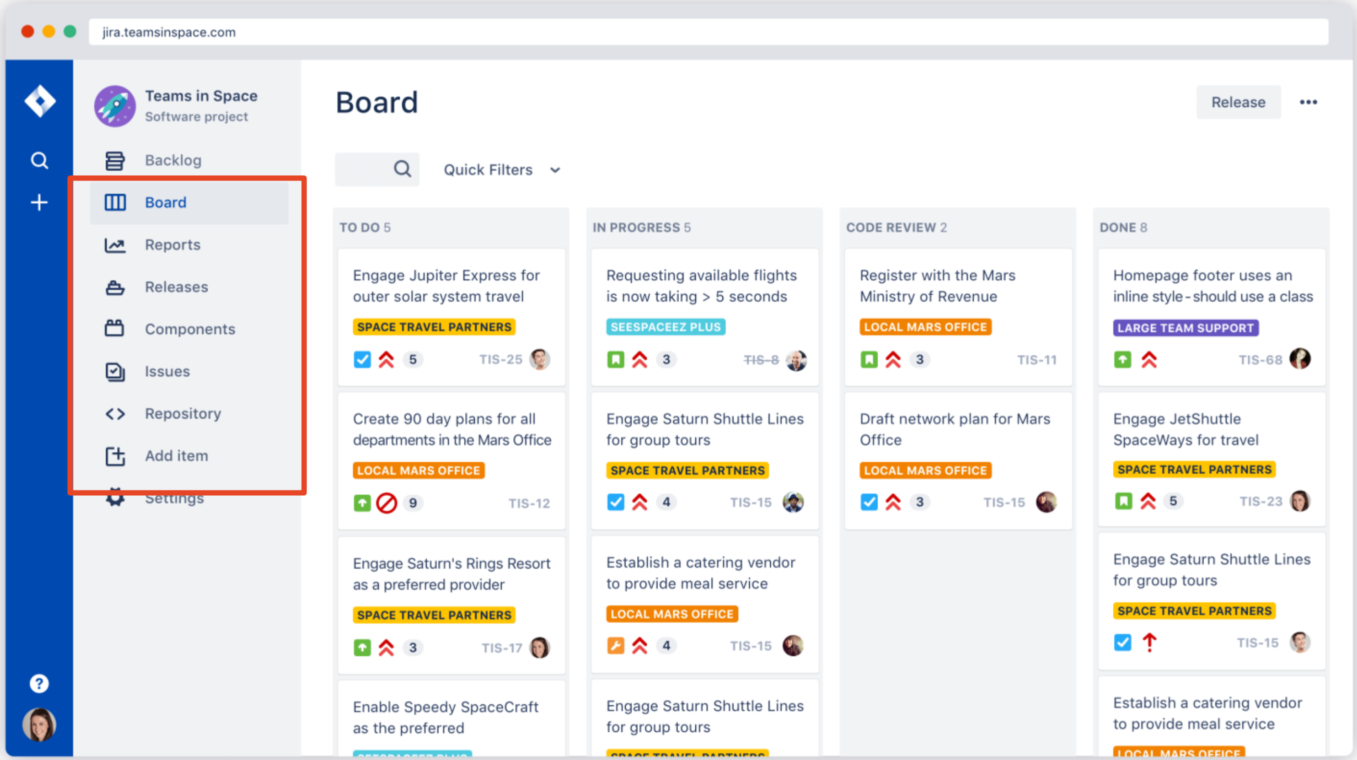
Tool Support for Agile SW Development – Requirements

- Jira agile is a software tool for planning, tracking and managing software development projects
  - Supports different agile methods (e.g., Scrum and Kanban)

- Jira Software supports Scrum Sprint planning, stand ups (daily scrums), Sprints and retrospectives
  - Including backlog management, project and issue tracking, agile reporting
    - E.g., Burndown and velocity charts, Sprint report
  - Scrum boards visualize all the work in a given Sprint

https://www.atlassian.com/software/jira/agile

Jira Agile – Scrum Boards & other Support



TO DO 5

Engage Jupiter Express for outer solar system travel
SPACE TRAVEL PARTNERS
5 TIS-25

Create 90 day plans for all departments in the Mars Office
LOCAL MARS OFFICE
9 TIS-12

Engage Saturn's Rings Resort as a preferred provider
SPACE TRAVEL PARTNERS
3 TIS-17

Enable Speedy SpaceCraft as the preferred

IN PROGRESS 5

Requesting available flights is now taking > 5 seconds
SEESPACEEZ PLUS
3 ~~TIS-8~~

Engage Saturn Shuttle Lines for group tours
SPACE TRAVEL PARTNERS
4 TIS-15

Establish a catering vendor to provide meal service
LOCAL MARS OFFICE
4 TIS-15

Engage Saturn Shuttle Lines for group tours

CODE REVIEW 2

Register with the Mars Ministry of Revenue
LOCAL MARS OFFICE
3 TIS-11

Draft network plan for Mars Office
LOCAL MARS OFFICE
3 TIS-15

DONE 8

Homepage footer uses an inline style - should use a class
LARGE TEAM SUPPORT
TIS-68

Engage JetShuttle SpaceWays for travel
SPACE TRAVEL PARTNERS
5 TIS-23

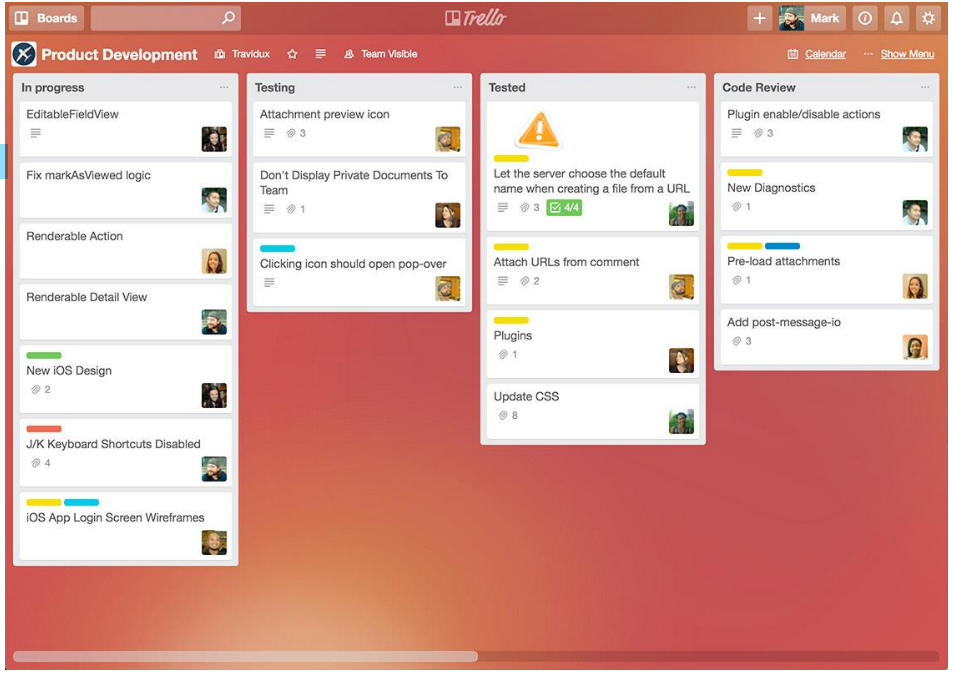
Engage Saturn Shuttle Lines for group tours
SPACE TRAVEL PARTNERS
TIS-15

Establish a catering vendor to provide meal service

https://www.atlassian.com/software/jira/agile

Tool Support – Trello

- Task/project management focus
- Tasks and project work are logged using a three-part hierarchy: Boards, Lists and Cards
- Lack of pre-built workflows
- Does not offer most of the agile software development features
    - E.g., Scrum and sprint planning, a backlog of user stories, detailed project reporting, issue tracking, and code repositories.



https://trello.com/

References

- Ian Sommerville 2016. Software Engineering: Global Edition (3rd edition). Pearson, Englad
- Armando Fox and David Patterson 2015. Engineering Software as a Service: An Agile Approach Using Cloud Computing (1st Edition). Strawberry Canyon LLC
- Andrew Stellman, Margaret C. L. Greene 2014. Learning Agile: Understanding Scrum, XP, Lean and Kanban (1st Edition). O'Reilly, CA, USA.
- Agile Manifesto. [http://agilemanifesto.org/]
- Agile Alliance. [http://www.agilealliance.org]
- Attlassian. JIRA Agile [https://www.atlassian.com/software/jira/agile]

Tutorial: Expressing Requirements (User Stories)
Lecture: Scrum – Planning and Estimation