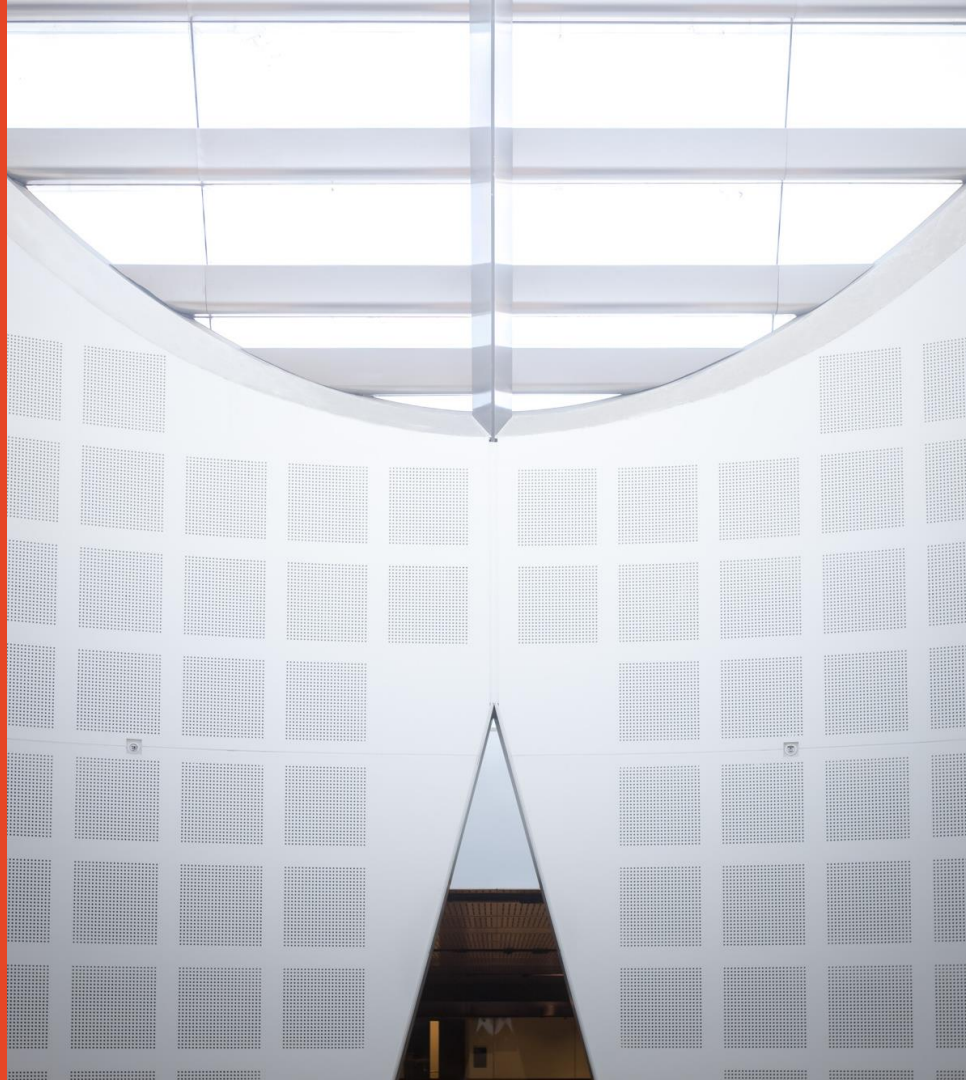
Agile Software Development Practices (SOFT2412/COMP9412)

Requirements; Technologies for Expressing Requirements

Dr. Basem Suleiman

School of Information Technologies

THE UNIVERSITY OF SYDNEY

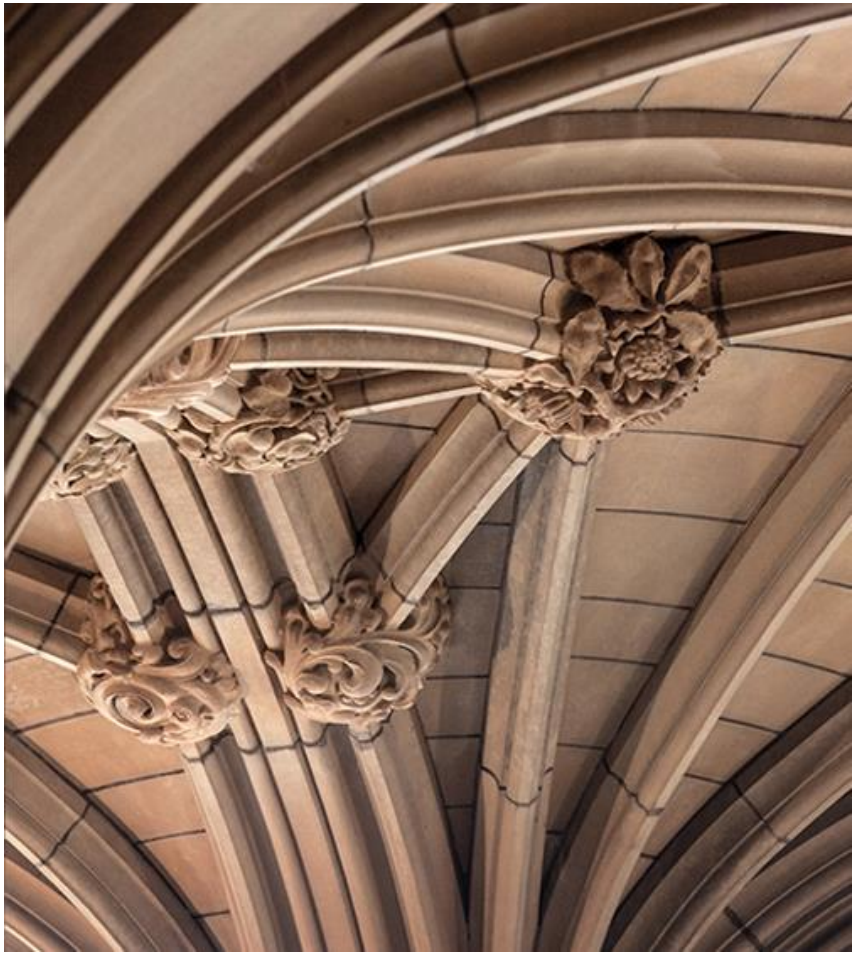


Copyright warning

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

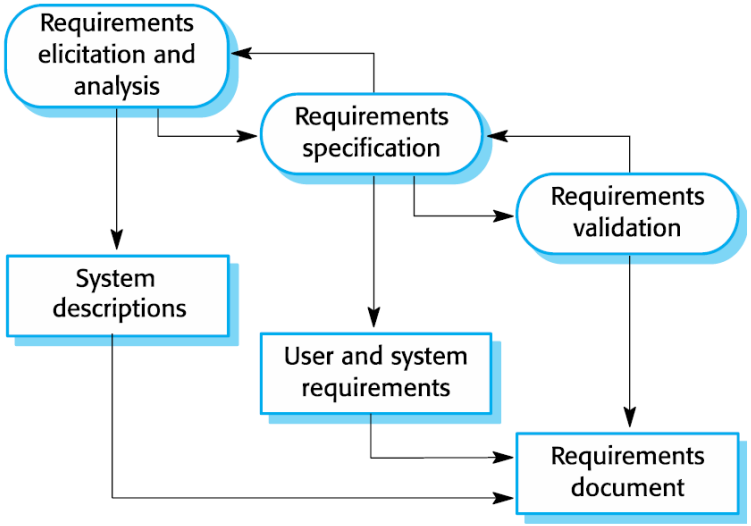
- Requirements, Analysis, Design, Code & Integrate, Test/QA, Deploy/Operate/Maintain
- Goal is to make Software Engineering predicable in budget and schedule
  - Requirements elicitation
  - Requirements documentation
  - Cost estimation
  - Scheduling and monitoring schedule
  - Change management for requirements, cost and schedule
  - Ensuring implementation matches requirement features
  - Risk analysis and management

Plan-and-document – Requirements

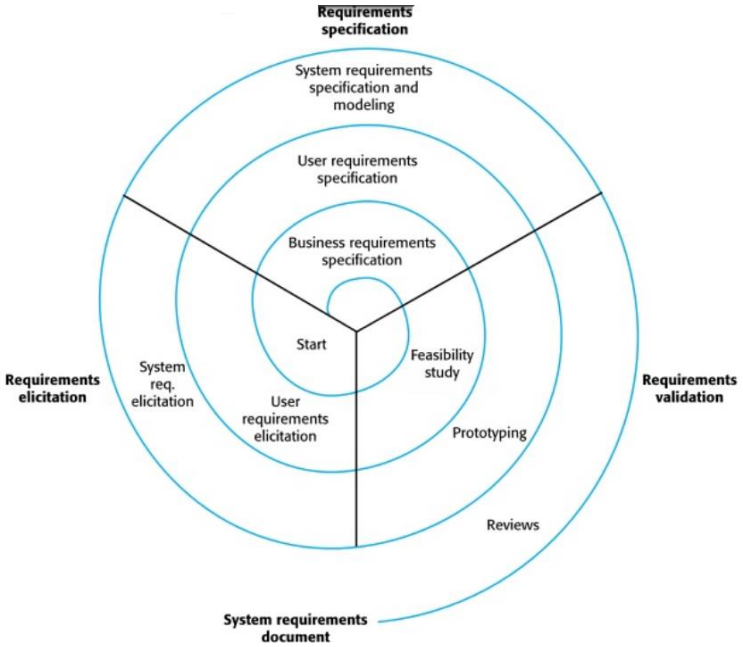
- In traditional methodologies, the requirements document is usually very detailed and extensive
- Often done in Word, to a detailed template
- The requirements are named and numbered, to support traceability
- See IEEE Standard 29148 for comprehensive details

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

Plan-and-Document – Requirements Engineering Process



Requirements engineering
main activities and deliverables



Requirements engineering
spiral view

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

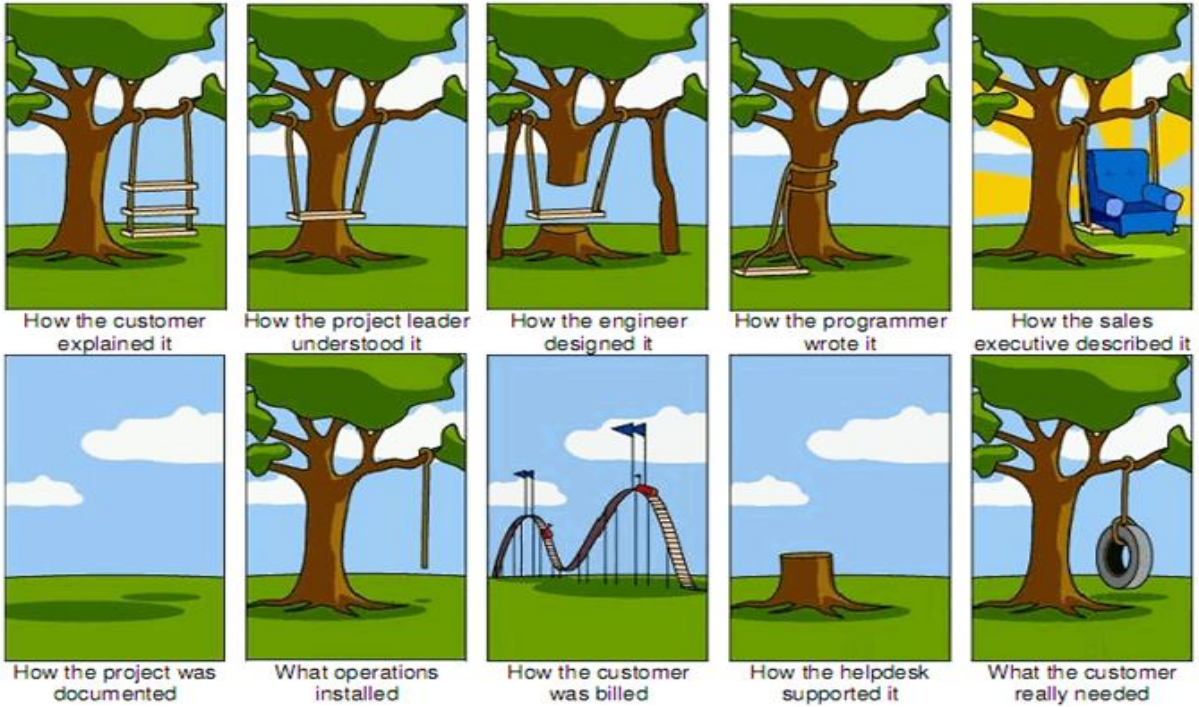
- Elicit what the software/system is supposed to be
- Functional
  - Details matter: exactly what information goes in and out
  - Interactions between features
  - Not only when things go well
- Non-functional
  - Include performance, security, usability, constraints on technologies

- Note that different stakeholders often have different thoughts on this
  - And one stakeholder can change their mind
  - And communicating details is hard

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: are all requirements necessary?
  - Consistency: do requirements conflict?
  - Completeness: are all requirements and constraints included?
  - Feasibility: can requirements be implemented?
- Estimate budget and schedule based on the SRS

Software Requirements Mystery



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

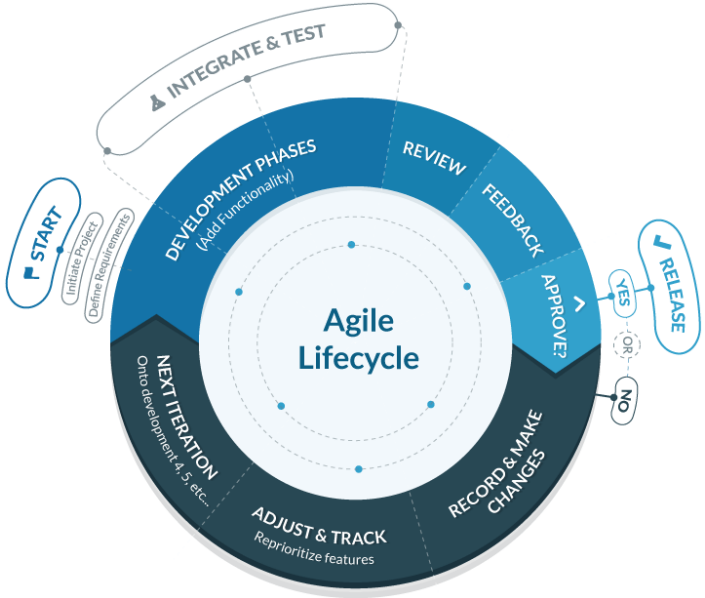
- Useless (unwanted) product features
  - Standish group's CHAOS report: annual survey repeatedly found that project teams felt that many features in the software they built were not used
    - 45% of features never used, 19% rarely used
  - Development teams would build software, and throw it over the wall to their users, and hope some of what they build would stick
    - Developers believe that software work exactly the way they intended it to work and users need to change their expectation to use it

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

Requirements in Agile Software Development

THE UNIVERSITY OF SYDNEY

Requirements in Agile Software Development?



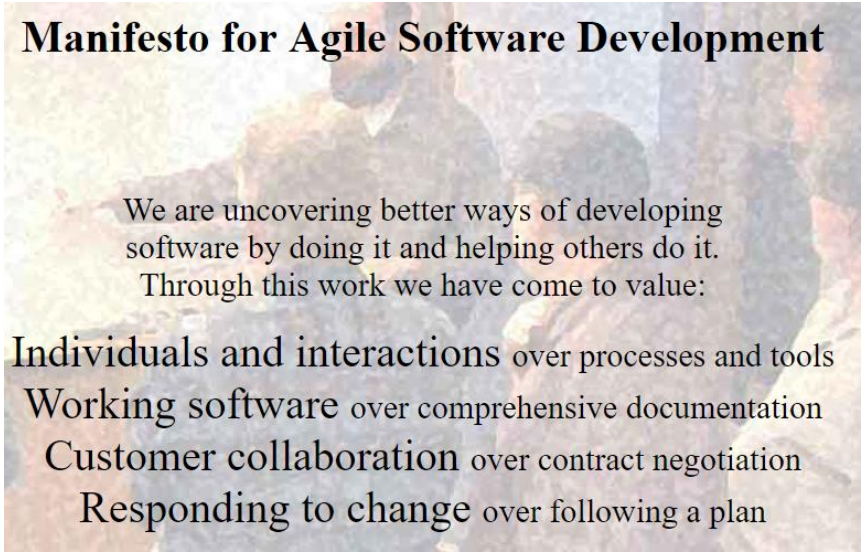
Agile model
Incremental & iterative development

Groups of 2, discuss:

- Where requirements fit in the agile software development?
- When requirements are captured?
- How requirements are identified, documented and managed?

https://blog.capterra.com/agile-vs-waterfall/

Agile Manifesto – Agile Values or Goals



That is, while there is value in the items on the right, we value the items on the left more.

Which agile value(s) relate to or have impact on how requirements are managed?

Agile Manifesto: http://agilemanifesto.org/

Agile Principles – Relation to Requirements?

Discuss - which Principles relate to requirements in Agile Software Development ?

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. Business people 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

Behaviour Driven Development (BDD)

THE UNIVERSITY OF SYDNEY

Behavior Driven Development (BDD)

- A conceptual approach for specifying application's behaviour (requirements) and communicating them clearly among stakeholders before and during development
- The goals of BDD requirements are:
  - Validation: build the right product
  - Verification: build the product right
- Business value (feature) → acceptance criteria → code to deliver this
- Uses simple notation (Given-When-Then canvas) which improves communication among domain experts, users, testers and developers

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

Requirements: Behavior Driven Development (BDD)

- Agile lifecycle involves:
  - Working closely and continuously with stakeholders to develop requirements and tests
  - Maintaining a working prototype while deploying new features (2-4 weeks iterations)
  - Continuously validating that the current system is what the stakeholders really want and what features to be added next
    - Maintenance mode as soon as the first set of features are developed

Requirements: Behavior Driven Development

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

BDD – User Stories

- Lightweight version of requirements that describe how the application is expected to be used from the user's point of view
  - We need feature X but we do not have enough details about it – put it as a user story
  - We got more details about feature Y – add it to the corresponding user story
  - Functionality that has value to the user and/or customer

- Helps stakeholders to plan and prioritize development
- Improve requirements clarity as it focuses on the behaviour of the application (not the implementation)

- Documents functional requirements as executable scenarios or examples

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

Agile Development – User Stories

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

- "As a <role>, I want <feature> so that <benefit>"
- Eg: As a potential student, I want to get information on a particular degree, so that I can make an informed decision about whether to apply

User Stories

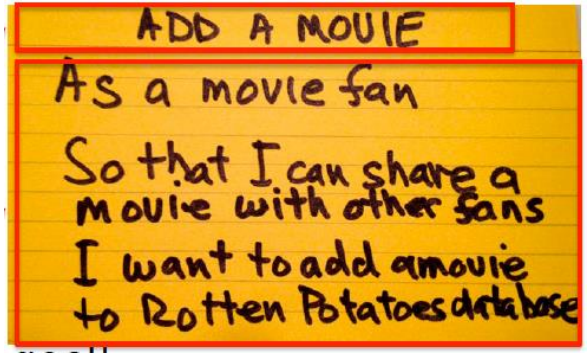
- Borrowed from Human Computer Interface (HCI) community
  - "3-by-5 cards" (3x5 inch index cards)
  - 1-3 non-technical sentences written jointly by the users/customers and developers
  - Must be small enough to implement in one iteration, testable and must have business value

- *Connextra* format
  - As a [stakeholder], I want a [feature], so that [benefit]

User Stories – Examples

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

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*

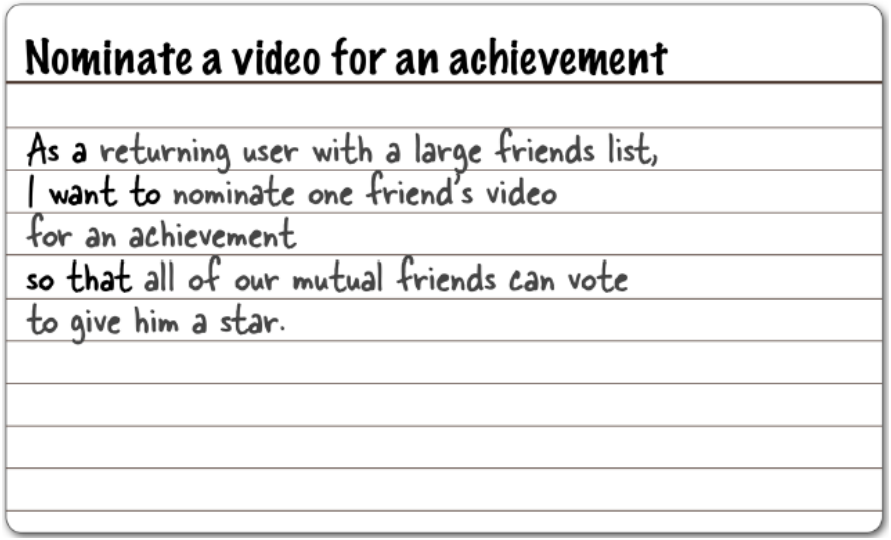


User Stories – Why?

– User stories help to build software features that users will use
  – 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



Common Mistakes

- Generic user, rather than meaningful role
- No evident business value
  - Less technical details (jargon)
  - especially: writing things the developer wants to do, but don't help the real goal of the organization
  - E.g., "wouldn't it be cool if…."
- How to write good user stories?

SMART User Stories

- **Specific**
  - Describe specific details that add value
- **Measurable**
  - User story should be testable; there are known expected results for some good inputs
- **Achievable**
  - Can be implemented in one Agile iteration
  - Subdivide big stories (features) into smaller ones – implement in one agile iteration
- **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

SMART User Stories

- **Timeboxed**
  - User stories should be time-boxed to estimate amount of time to implement it
  - Stop developing the story if allocated time is over
    - Divide the user story into smaller ones or reschedule what's left according to new estimate
    - If dividing won't help discuss with the customer the part of highest value

- Note: Minimum Viable Product (MVP) aims at implementing subset of the full set of features that when completed has business value in real world
    - Relevant stories and their combination make the software viable in the real world

SMART User Stories – Examples

Group Exercise: Evaluate the following features/stories using SMART. If any identified as not SMART, re-write using SMART technique.

- User can search for a movie
- RottenPotatose website should have a good response time
- User can search movie by title, year, genre, actor names, producers or any combination of those

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

Acceptance Criteria (Conditions of Satisfaction)

– Condition of Satisfaction (or Acceptance Criteria): an effective tool for helping developers to gauge completion/satisfaction of software features

  – Concrete definition of “Done” or “Completed”

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

Acceptance Criteria (Condition of Satisfaction)

- How will you know the work is “Done”?
- E.g.,
  - Given: Potential student Jane who has created an account
  - When: Jane asks for information about BCST degree
  - Then: Jane is shown the page at http://sydney.edu.au/courses/bachelor-of-computer-science-and-technology

Common mistakes

- Missing crucial details, such as
  - which user,
  - what status is at the time,
  - parameters of the event,
  - what gets returned

User Stories and Epics

- **Epics** are larger bodies of work that can be broken into smaller tasks (user stories)
  - Often encompass multiple teams, on multiple projects
  - Almost always delivered over a set of Sprints

- **User stories** short/brief requirements or requests written from the end user perspective
  - Who, what and why
  - Defines details of an epic

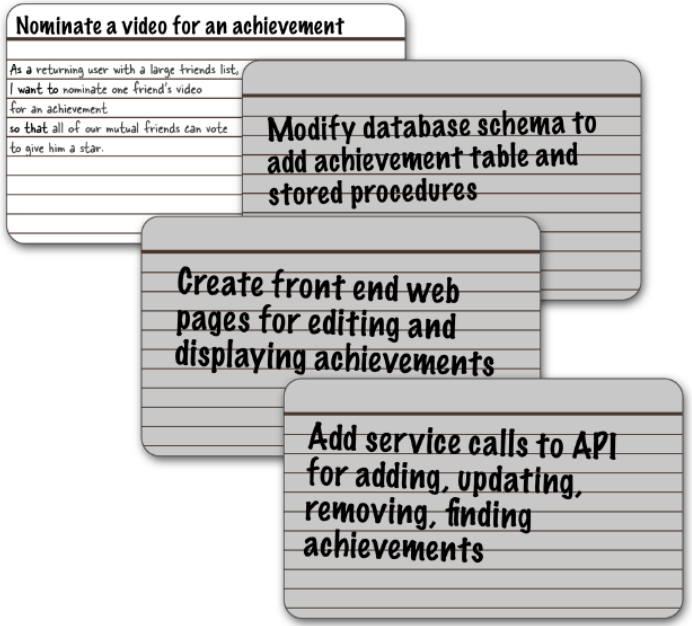
Scrum – User Stories

Sprint planning meeting – part 2:

1. Break the stories down into tasks that team members will carry out during the Sprint

- Team members work together to identify list of individual tasks to
- Each task on a separate card

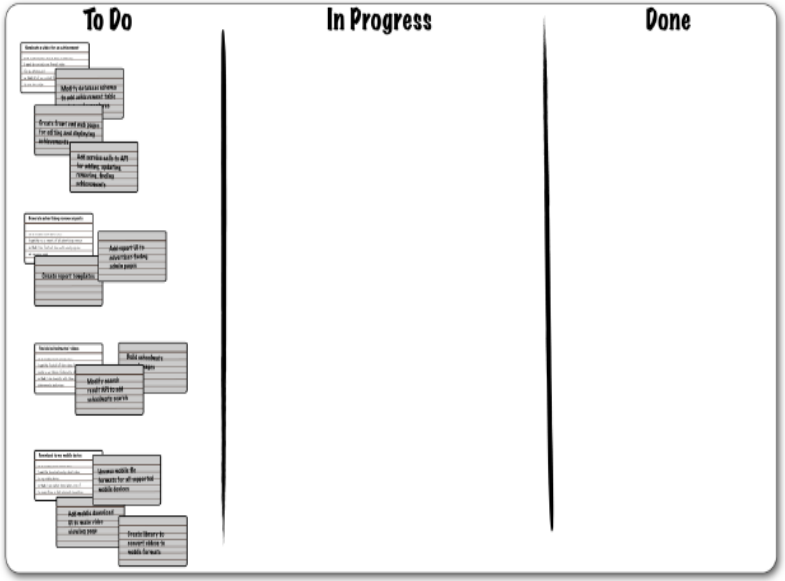
- Uncompleted stories to be put on a single card to plan out the story



Planning and Running a Sprint

Sprint planning meeting – part 2:

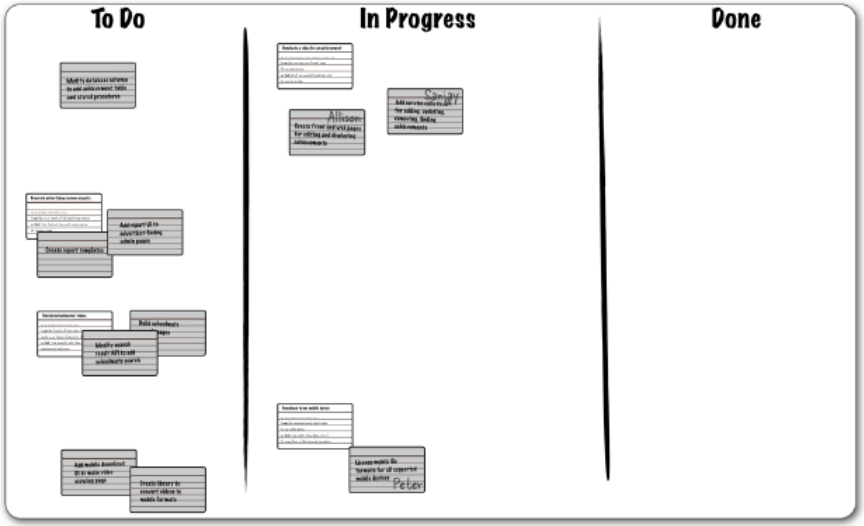
2. Group the stories and their tasks together
   - Add them to the "To-Do" column of the **Task Board**



Planning and Running a Sprint

Sprint planning meeting – part 2:

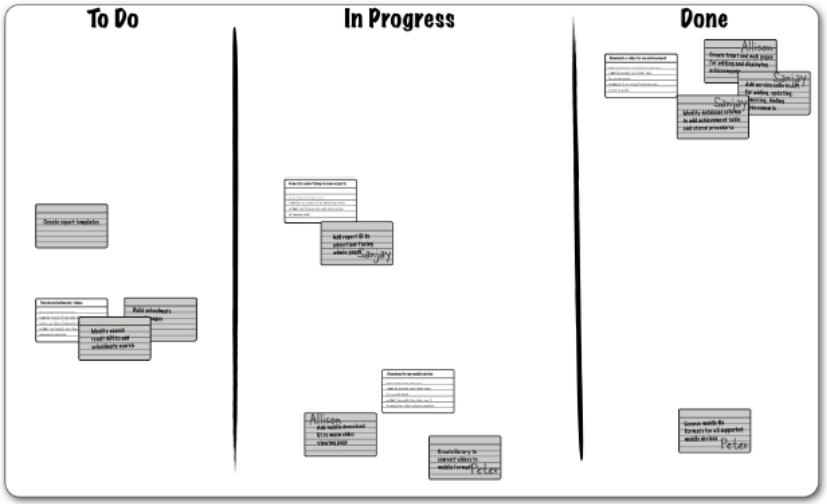
3. Team members to work on the tasks
   - Each team member works on one task at a time by writing their name on it and moving it to the “In-Progress” column
   - Team members may need to add additional tasks to finish a story. They should add it to the board and inform team members about it during the Daily Scrum



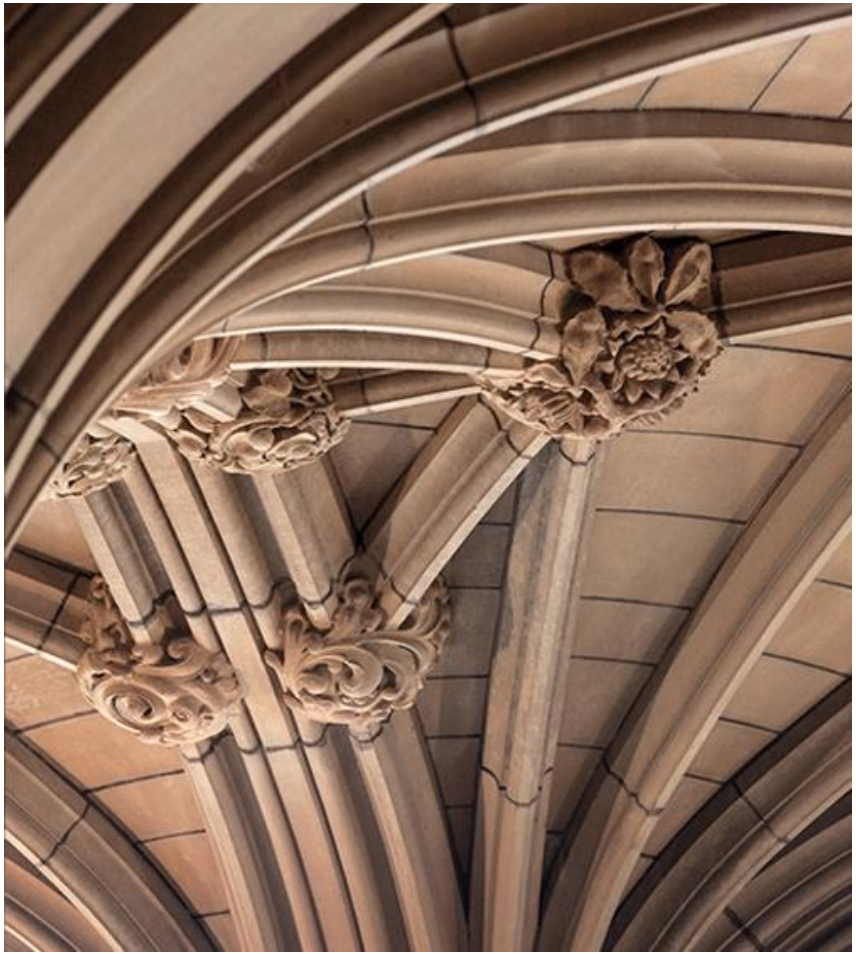
Planning and Running a Sprint

Sprint planning meeting – part 2:

4. Finishing a task/story
   - Task completed → "Done" and pulls another task
   - The team member who finishes the Final task of a story will have to verify that all of the conditions of satisfaction are completed and move it to the "Done" column



User Interfaces, Scenarios and Storyboard

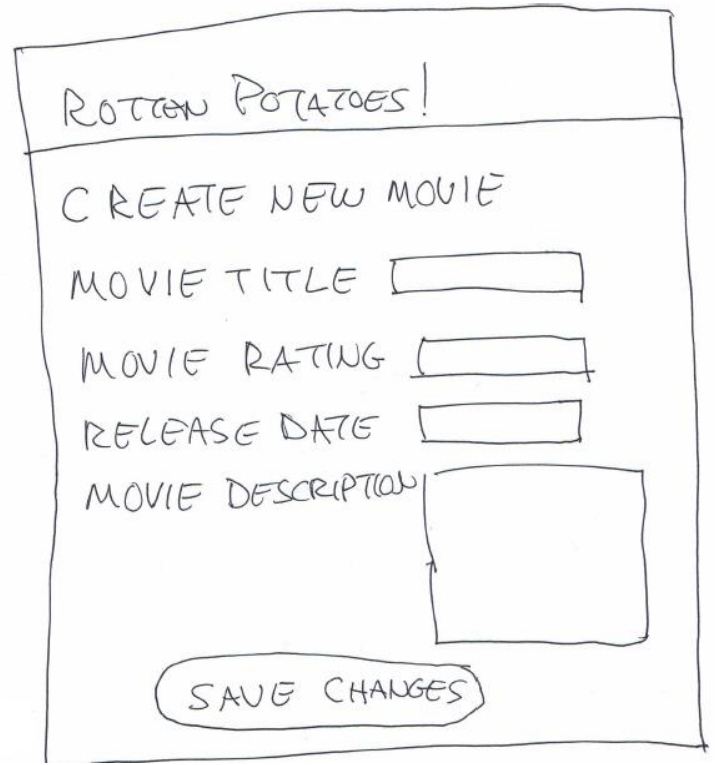


THE UNIVERSITY OF SYDNEY

User Interface (UI) Sketches

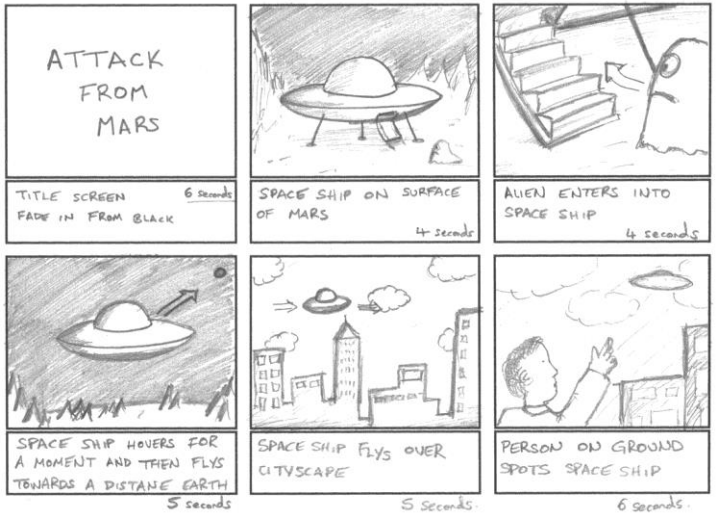
- **Low-Fidelity (Lo-Fi) UIs**
  - A rough (UI) sketch or mock-up of a corresponding user story
  - Low-tech approach to UIs and the paper prototype sketches
    - Pencil-and-paper to quickly produce
  - Shows how a UI looks like and how sketches work together as a user interacts with the UIs
  - Effective way for communication and engaging nontechnical stakeholders
- **High-Fidelity (Hi-Fi) UIs**
  - Includes higher level of details that more closely matches the design of the actual UI (also used for documentation)

Lo-Fi Example – Add Movie



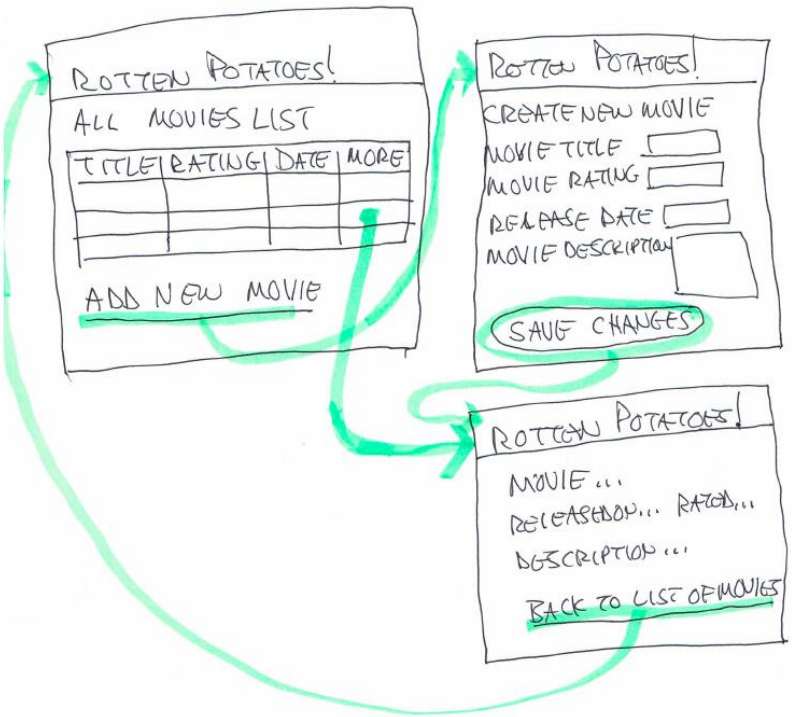
Scenarios and Storyboards

- Originated in filmmaking industry to visualize a film's senses in a sequence
- A scenario is a written description of the system interactions from one or more user's perspectives
- A storyboard is similar to scenario but it visualizes the interactions
- A user story refer to a single feature, and a feature usually has one or more scenarios that show different ways the feature is used
- A scenario consists a few steps (typically 3 to 8 steps)



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

Storyboard Example – Add New Movie



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

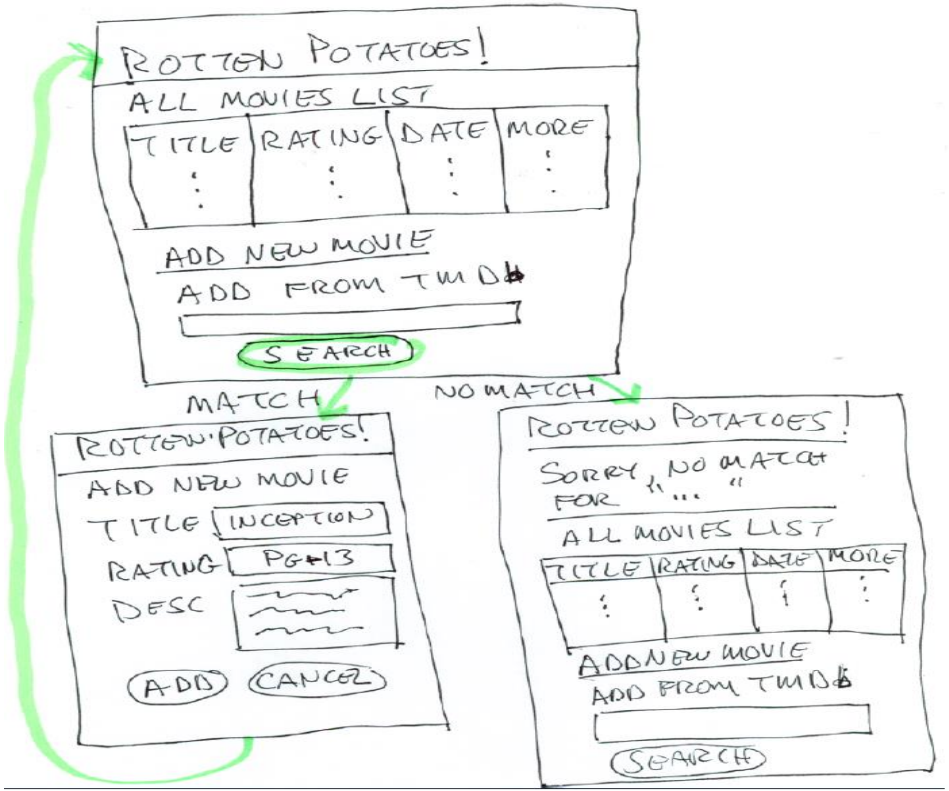
Scenario Example – Add a Movie

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”

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

Scenario: Try to add existing movie (happy path)
GIVEN: I am on the RottenPotatoes home page
THEN: I should see "Search TMDb for a movie"
WHEN: I fill in Search terms with "Inception"
AND: I press "Search TMDb"
THEN: I should be on the "Search Results" page
AND: I should not see "Not found"
AND: I should see "Inception"

….

Scenario: Try to add non-existent movie (sad path)
GIVEN: I am on the RottenPotatoes home page
THEN: I should see "Search TMDb for a movie"
WHEN: I fill in Search terms with "Movie that does not exist"
AND: I press "Search TMDb"
THEN: I should be on the RottenPotatoes home page
AND: I should see "'Movie that does not exist' was not found in TMDb"

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 "Search TMDb for a movie"

Scenario: Try to add existing movie (happy path)

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

….
THEN: I should be on the "Search Results" page
AND: I should not see "Not found"
AND: I should see "Inception"

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 TMDb"
THEN: I should be on the RottenPotatoes home page
AND: I should see "'Movie that does not exist' was not found in TMDb"

Implicit vs. Explicit Scenarios

- **Explicit requirements** are explicitly specified as user stories developed by stakeholders in BDD (and as formal specification in a plan-and-document process)
- **Implicit requirements** are not specified in the user stories
  - E.g., by default movies should be listed in a chronological order by release date
- 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

Imperative Scenarios

- **Imperative Scenarios** tend to have complicated WHEN statements and lots of AND steps
  - Ensures that the details of UIs match customer expectations, e.g., filling in a form, clicking a button, etc.
  - Tedious to write such scenarios

- Suppose we want to write a feature that specifies that movie should appear in alphabetical order on the list of movie page
  - "Zorro" should appear after "Apocalypse Now" even if the former was added first

Imperative Scenario – Movie in Alphabetical Order

Discuss:

- How well is this scenario documented?
- What is the main focus of the scenario?
- If the focus is not what it is suppose to be, what should be the main focus instead?

Feature: Movie should appear in alphabetical order, not added order
Scenario: view movie list after adding 2 movies (imperative)

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 "Zorro"
AND: I select "PG-13" from "Rating"
AND: I press "Save Changes"
THEN: I should be 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 "Apocalypse Now"
AND: I select "R" from "Rating"
AND: I press "Save Changes"
THEN: I should be on the RottenPotatoes home page
THEN: I should see "Apocalypse Now" before "Zorro" on the RottenPotatoes home page sorted by title

Declarative Scenarios

- **Declarative Scenarios** domain language-oriented scenarios focus on the feature being described, rather than the low-level details, 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 user interface implementation
  - Less dependent on the details of the UI; describes the state of the world rather than the sequence of steps to get to that state
- 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

Declarative Scenario – Movie in Alphabetical Order

Feature: Movie should appear in alphabetical order, not added order

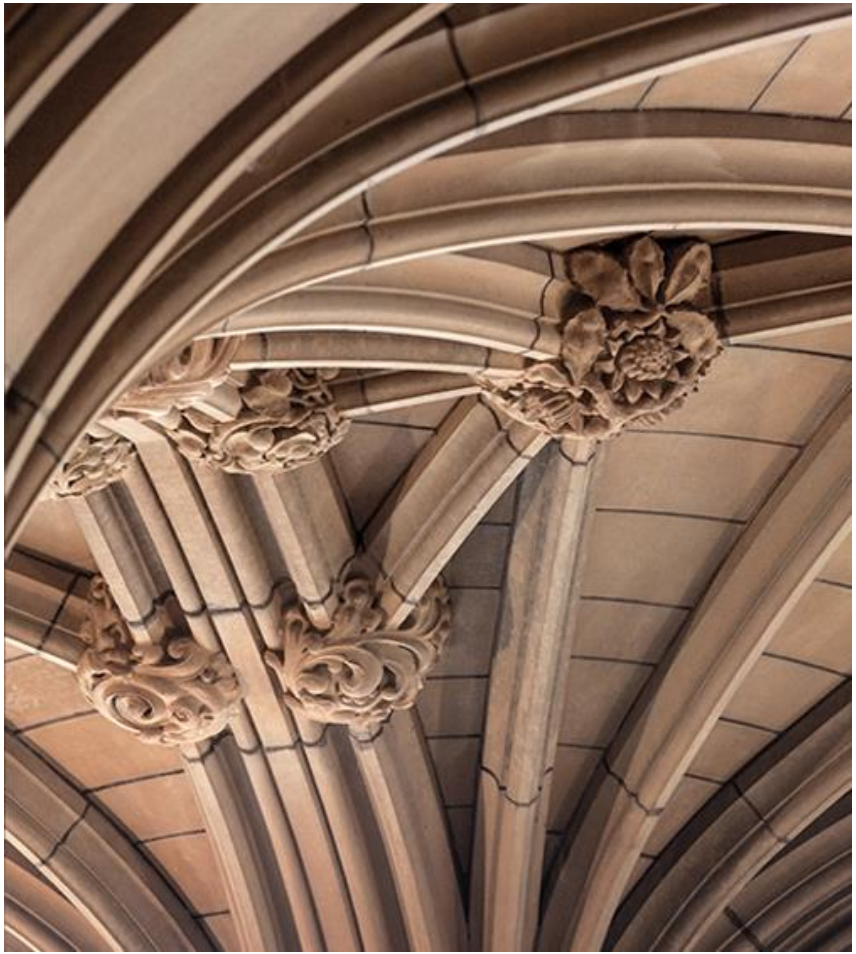
Scenario: view movie list after adding 2 movies (declarative)

GIVEN: I have added “Zorro” with rating “PG-13”
AND: I have added “Apocalypse Now” with rating “R”
THEN: I should see “Apocalypse Now” before “Zorro” on the RottenPotatoes home page sorted by title

The Declarative version

Less verbose, easier to maintain and easier to understand as it describes the state of the application (behaviour)

Tools for Agile Software Development



THE UNIVERSITY OF SYDNEY

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

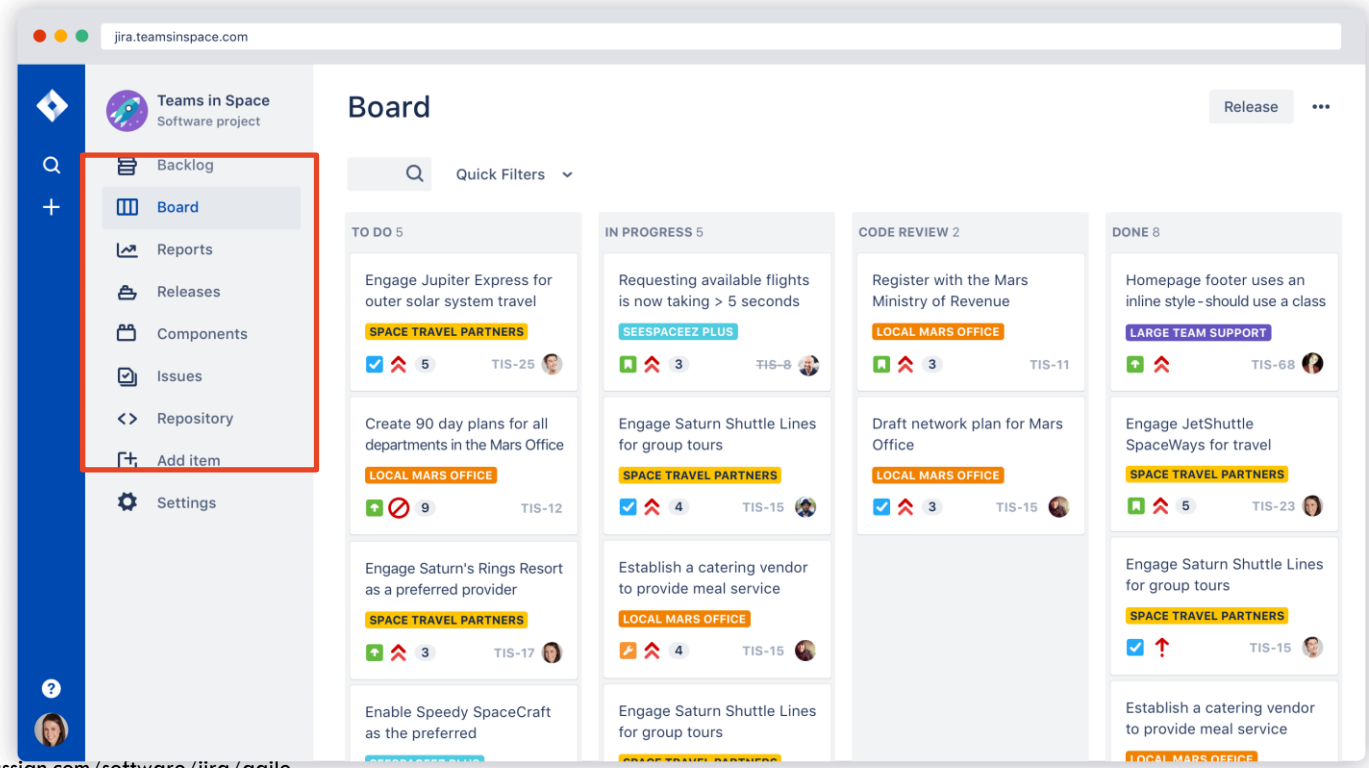
Tool Support for Agile SW Development

- Jira agile is a software tool for planning, tracking and managing software development projects
  - Supports different agile methodologies including 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

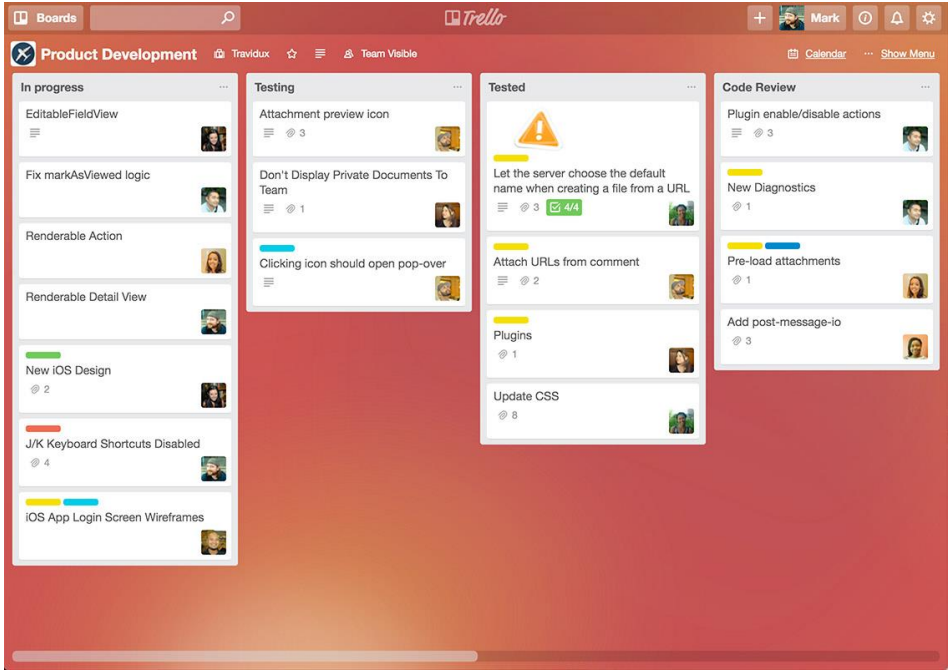
Jira Agile – Scrum Boards & other support



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

- 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.