Software Engineering Project Workshop (SENG202)

Matthias Galster

Phase 1 – tasks (part 2)

July 16, 2020

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Reminders and heads-up

- Send us (one) weekly meeting time/location
  - Due: July 17, 5:00pm (remember penalty)
- Submit your first reflection
  - Due: Mondays, 5:00pm (remember penalty)
- Start/keep logging
  - Clockify
- Quiz (to be released today)
  - Due: July 17, 5:00pm (no late submissions accepted)

Deliverables

- Project setup checklist
- Design document
- Reflections and logs
- Presentation

Content of design document

Executive summary

1. Business and system context
2. Stakeholders and requirements
3. Acceptance tests
4. GUI prototypes
5. Deployment model
6. Detailed UML class diagram
7. Risk assessment
8. Project plan

References

Appendix

2. Stakeholders and requirements

2.1 Stakeholders

- Table with stakeholders

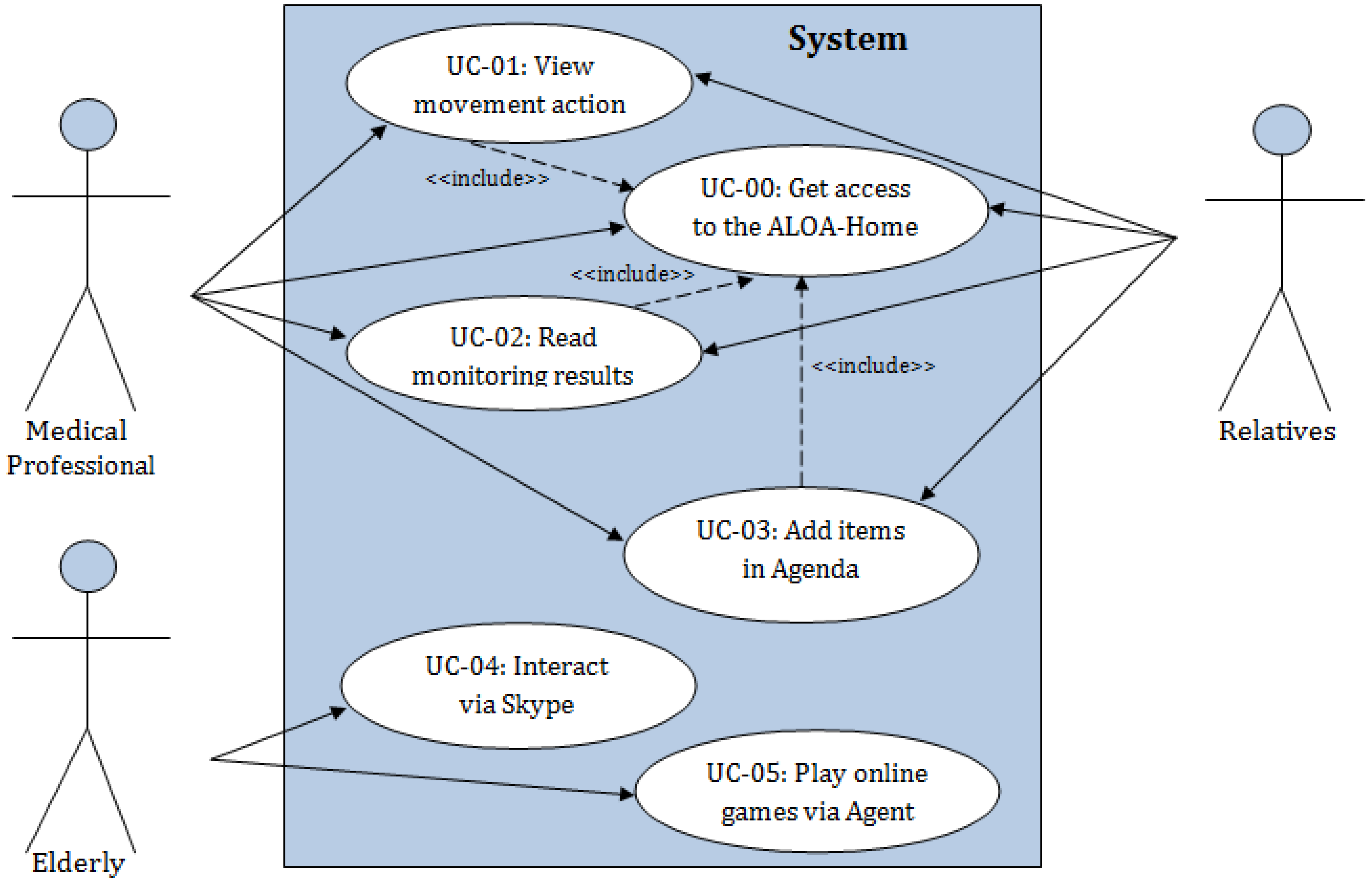
2.2 Requirements

- Use cases
    - Use case diagrams
    - Textual description of use cases
- Functional requirements
- Quality requirements

Use cases

- What is a use case?
  - Describes system behaviour
  - Stimulates discussion
- Start with high-level use case(s), then refine
  - Core things that the application needs to do
  - Derived from business vision and goal
- Need to be prioritized
- Tools
  - dia, umbrella, argoUML, Visio, etc.

Use case diagrams – example


System
UC-01: View movement action
<<include>>
UC-00: Get access to the ALOA-Home
<<include>>
UC-02: Read monitoring results
<<include>>
UC-03: Add items in Agenda
UC-04: Interact via Skype
UC-05: Play online games via Agent
Medical Professional
Relatives
Elderly

Textual description of a use case

- ID
- Name
- Actor(s)
- Precondition(s)
- Basic flow + alternative flow/exceptional flow
- Post-condition(s)

Hint

High level use cases can help identify more detailed functional requirements. Use cases should cover the core things the application needs to do. Create use cases which describe the functionality of the system. Combine the graphical use case diagram with the structural natural language description. At an early stage it is often sufficient to have one high-level use case diagram that shows all actors and major use cases. This use case diagram can then be refined later. In your report use cases must include use case diagrams, a textual structured description of use cases and a brief description of each actor.

Functional requirements

ID	Description	Stakeholder	Priority	Use case(s)

- Hints for phrasing requirements
  - E.g., Karl E. Wiegers
    - http://www.processimpact.com/articles/qualreqs.html
- Principles
  - SMART (Specific, Measurable, Agreed, Realistic, Time-bound)
  - INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable)

Quality requirements

- Commercial, e.g.,
  - There should be no licensing issues with product
  - The product should not violate any privacy laws in the country it is used in
- Technical, e.g.,
  - Design time, e.g.,
    - Portability (description must be concrete enough)
    - Maintainability (description must be concrete enough)
  - Runtime, e.g.,
    - Security
    - Performance
- Same table as before (without use case[s])

Key drivers

Quality requirements

	Weight*	Usability	…	
Citizen Police officer …	0.6 0.4	50 0		
Total		30		

e.g., sum must be 100

Stakeholders

Weighted sum for usability (i.e., 0.6 x 50 + 0.4 * 0)

* Weights and scores for quality requirements must be consistent with priority of stakeholders and requirements.

Hint

Requirements are usually negotiated with stakeholders. Throughout your project, prioritization of requirements will become necessary since it is not feasible to implement all possible features within the given time and resource budget. Thus, you may come up with an initial feature list that is then cut down to what is feasible given your technical constraints and budget. Requirements cover functional aspects (i.e., what the system should do) and quality requirements (how the system does things, e.g., in terms of performance, usability, reliability). The most important quality attributes are key drivers. Usually, there are around 3-5 key drivers in a system.

Content of design document

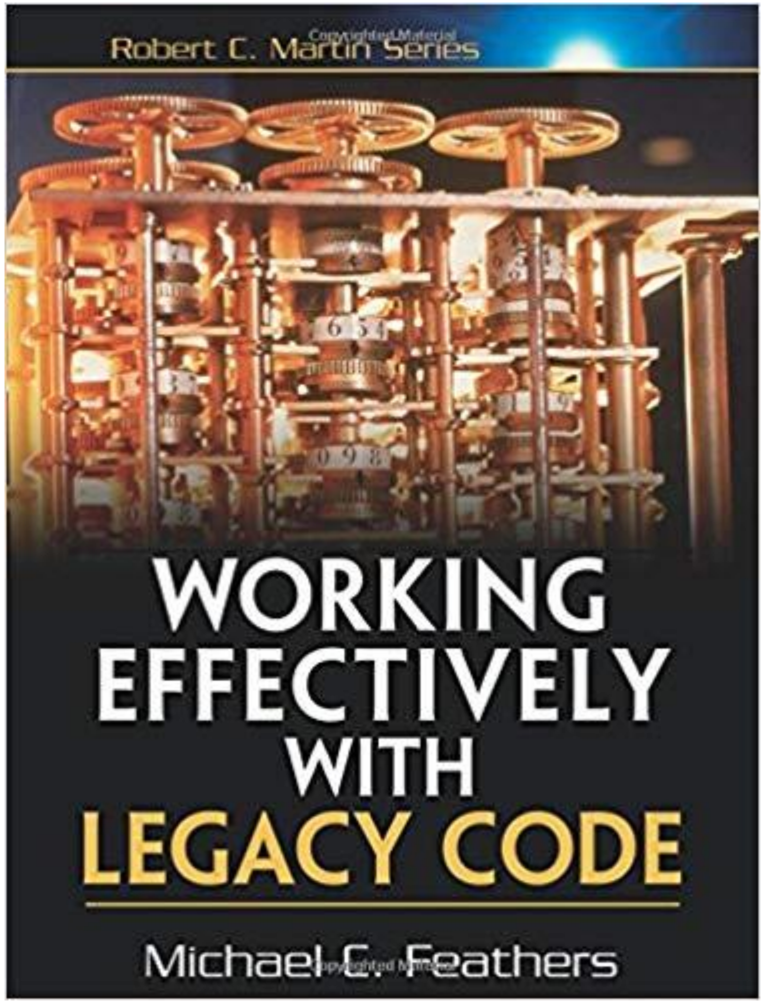
Executive summary

1. Business and system context
2. Stakeholders and requirements
3. Acceptance tests
4. GUI prototypes
5. Deployment model
6. Detailed UML class diagram
7. Risk assessment
8. Project plan

References

Appendix

Legacy code is code without tests



Acceptance tests

- Based on key features and end user requirements
  - Often described together with requirements
  - Determine if requirements of a specification (or contract) are met
  - Negotiated with customer

- Conducted by customer
  - User or client instead of “technical people”
  - Often performed prior to transfer of ownership
  - Conducted to test software in the “real world”, with actual use of system

- Black box or white box texting?

Template (minimalistic)

ID	Description	Responsibility	Acceptance criteria	Criticality	Use case
	See next slide	Who is responsible for testing?	When is test considered successful	How critical is it that the test passes	Related use case

- Above is one example – other templates more comprehensive
  - Expected and actual test results
  - Information about relationships between tests
  - Subsystems involved in tests
  - Etc.

Description of acceptance test

- Template inspired by behavior-driven development (BDD)
  - Detailed enough for a tester to actually perform the test
    - Given <condition>: initial context for the example
    - When <activity>: event/action actor in the system or stakeholder performs
    - [Then <result>: expected outcome of that action (acceptance criteria)]
    - Optional "And"

- Disclaimer: not really BDD – there is more to it
  - Features/user stories
    - "In order to <goal> as <role> I want to be able to <activity> then <result>"
  - Acceptance criteria of the form
    - "Given-When-Then"
  - Unit testing, tools, frameworks, etc.

See also https://dannorth.net/introducing-bdd/, http://behaviourdriven.org/, https://inviqa.com/bdd-guide

Description – examples

- **Given** a product is priced at £15, **when** I add it to the basket, **then** the total basket price should be £21

- **Given** a customer previously bought a black sweater from me, **and** I currently have three black sweaters left in stock, **when** she returns the sweater for a refund, **then** I should have four black sweaters in stock

- **Given** that a customer buys a blue garment, **and** I have two blue garments in stock, **and** three black garments in stock, **when** she returns the garment for a replacement in black, **then** should have three blue garments in stock, **and** two black garments in stock

Content of design document

Executive summary

1. Business and system context
2. Stakeholders and requirements
3. Acceptance tests
4. GUI prototypes
5. Deployment model
6. Detailed UML class diagram
7. Risk assessment
8. Project plan

References

Appendix

GUI prototypes

- To understand features, interactions, how system will be used
- Screenshots, paper drawings, whiteboard drawings
  - No implementation required
  - Tools like https://moqups.com/
  - Research literature, e.g., Rettig, CACM 37(4), pp. 21-27 (on Learn)
  - Include brief textual description
  - Danger: potential time sink
- Not about best design but to support thought process
- Beware effort for automating UI tests (TestFX, javafx.scene.robot)

Reminder

Executive summary

1. Business and system context
2. Stakeholders and requirements
3. Acceptance tests
4. GUI prototypes
5. Deployment model
6. Detailed UML class diagram
7. Risk assessment
8. Project plan

References

Appendix

Documents are "linear", how you work is not