

CSC216: Project 1

Project 1, Part 1: ReqKeeper - Requirements Management Software

[Home](#) > [Projects](#) > [Project 1: Scrum Backlog](#) > [Project 1, Part 1: ReqKeeper - Requirements Management Software](#)

[Project 1, Part 2: Scrum Backlog](#) ►

Project 1, Part 1: ReqKeeper - Requirements Management Software

Project 1 requires you to go through standard software development phases to design, implement, and test a complete Java program consisting at multiple source code files and JUnit test cases.

The project comes in two parts. Deliverables for Part 1 are:

1. Design document that includes a UML class diagram.
2. Black box test plan

Project 1 Part 1 Due Date: June 14 at 11:45 pm.

Requirements

When implementing new software, developers must keep track of a large number of requirements. Managing requirements helps developers decide what still needs to be implemented, what the priorities are, and make sure that nothing essential gets left out.

Software that enables developers to track the requirements are called *requirements management systems*. From quite small to very large software projects, requirements management systems can be vital to ensure that all developers know what is left to implement

with their software. Requirements management systems reflect the different steps organizations take when working to implement requirements. The process of managing requirements can be modeled as a finite-state machine (FSM).

For this project, you must implement a requirements management system called `REQKEEPER` that uses an FSM modified from the [Bugzilla](#) open-source bug tracking system. (It turns out that since an un-implemented requirement can be thought of as a type of bug, requirements tracking and bug tracking have a lot in common!) The FSM is illustrated in Figure 1. (The transitions are too detailed to fully label in the diagram. Instead each transition is given a name describing the type of action for moving to the next state. Descriptions of the transitions follow below.)

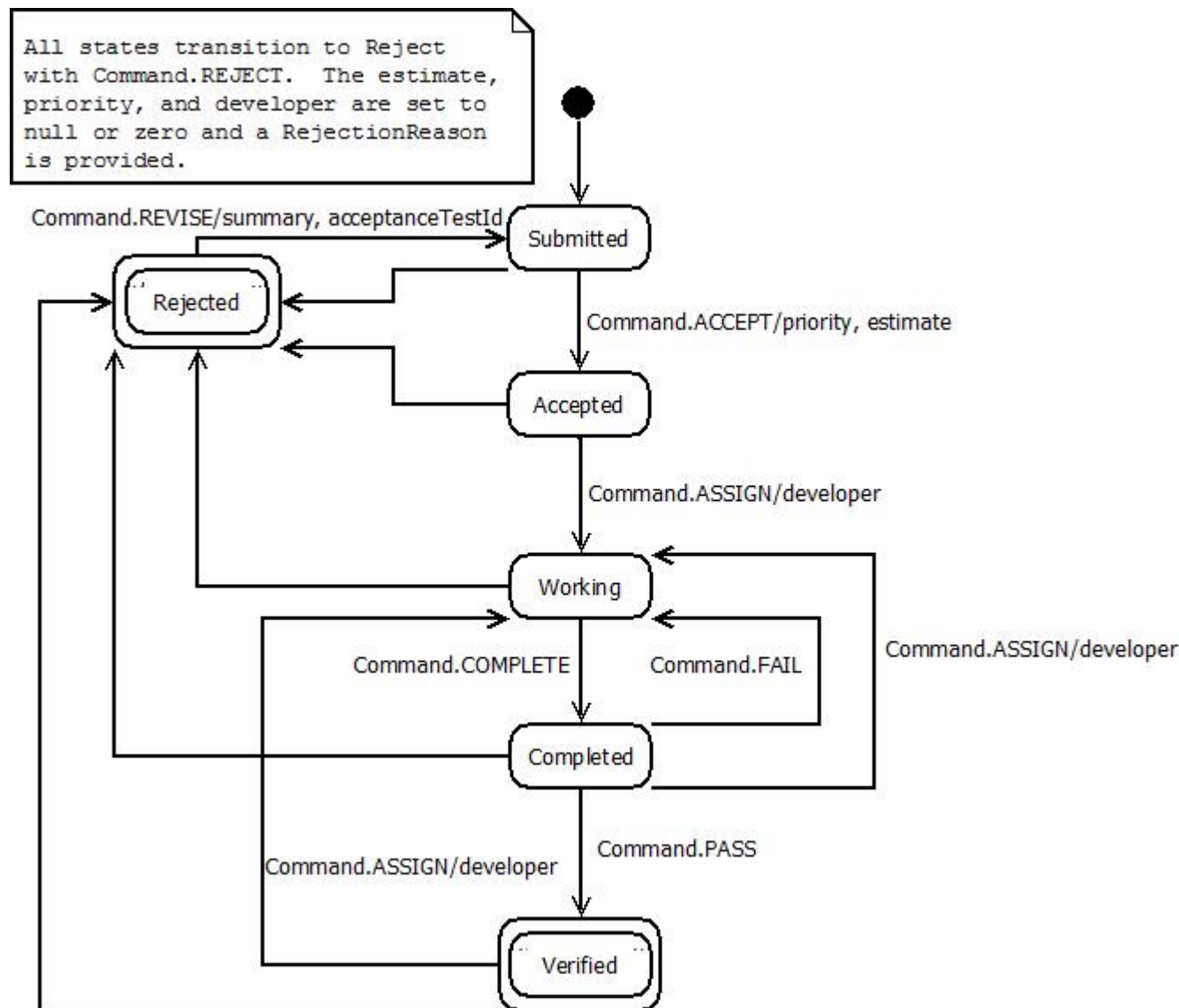


Figure 1: Requirements State Diagram for ReqKEEPER

A requirement in ReqKEEPER can be in one of the following states: **Submitted**, **Accepted**, **Working**, **Completed**, **Verified**, and **Rejected**. The meanings of these states and their corresponding transitions are described as follows. A requirement may be rejected when in any of the states, except rejected. This could be for several reasons: the requirement is a duplicate of another one already in the system, the requirement is infeasible, the requirement is too large, the requirement is out of scope, or the requirement is inappropriate.

1. **Submitted.** Requirements enter REQKEEPER into the Submitted state. All requirements in the Submitted state have a status of Submitted, which means the requirement has not been accepted by a developer or manager for future implementation in the system. For a requirement to transition out of the Submitted state, the requirement must either 1) be accepted by a developer or manager, or 2) be rejected (meaning that requirement is not in a condition to be accepted as a development task yet). If the requirement is accepted, it will never return to the Submitted state unless it is first rejected, possibly through some other path in the FSM.
 1. *SubmittedA.* Transition from Submitted to Accepted. The transition occurs when a developer or manager accepts the requirement. This can occur on requirements just entering the system or on requirements that have returned to the Submitted state through other paths. An estimate of the time to complete a requirement and a priority are provided on the transition.
 2. *SubmittedB.* Transition from Submitted to Rejected. This occurs if the developer rejects the requirement because of any of the above-mentioned reasons.
2. **Accepted.** Requirements in Accepted state have been accepted as tasks that need a developer to work on them, but they have not yet assigned a developer. Transition out of the Accepted state occurs when the requirement is assigned to the developer responsible for implementing the requirement or when the requirement is rejected.
 1. *AcceptedA.* Transition from Accepted to Working. A developer responsible for implementing the requirement is associated with the requirement. The requirement retains the provided developer unless the requirement is reassigned to another developer.
 2. *AcceptedB.* Transition from Accepted to Rejected. This occurs if the developer rejects the requirement because of any of the above-mentioned reasons.
3. **Working.** Requirements in Working have developers. In this state, the developer is working to implement the requirement. The resolution is one of the following two options:
 - Complete: the requirement has been implemented.
 - Reject: the developer won't implement the requirement. The transitions from Working depend on the resolution:
 1. *WorkingA.* Transition from Working to Completed. The requirement's implementation is complete and it is ready for testing. The requirement then moves to the Completed state for further testing.
 2. *WorkingB.* Transition from Working to Rejected. This occurs if the developer rejects the requirement because of any of the above-mentioned reasons.
4. **Completed.** A completed requirement has been implemented by the developer and is passed along for verification, which is done through testing. The requirement can transition out of the Completed state in several ways:
 1. *CompletedA.* From Completed to Verified. The software passed its associated test.
 2. *CompletedB.* From Completed to Working. The test fails leading to additional rework of the requirement.
 3. *CompletedC.* From Completed to Working. A (possibly different) developer is assigned responsibility for implementing the requirement.

4. **CompletedD.** From Completed to Rejected. This occurs if the developer rejects the requirement because of any of the above-mentioned reasons.
5. **Verified.** A final state. From here the requirement can be reopened if it fails any subsequent test or if its implementation is found not to be fully correct. It can also move to the Rejected state. There are two transitions:
 1. *VerifiedA.* From Verified to Working. Assign the requirement to another developer for revisions.
 2. *VerifiedB.* From Verified to Rejected. Reject a requirement, indicating that it will no longer be addressed because of any of the above-mentioned reasons.
6. **Rejected.** A final state. From here, the requirement can be revised, which puts it back in to the Submitted state. There is one transition:
 1. *RejectedA.* Transition from Rejected to Submitted. Re-submits the requirement back to the initial state.

We will now show the remaining requirements for REQKEEPER through a series of Use Cases (UCs). Your design must accomplish these requirements with the State Pattern. Use cases divide system behavior into related scenarios around a core piece of functionality. These scenarios tie directly to the user experience with the GUI. They describe the different paths that a user can take when interacting with the system. *These paths translate into black box tests for the program.*

Use Case 1 (UC1): requirement XML File Interactions

Precondition: A user has started the requirement management system and the GUI (shown in Figure 2) has opened.

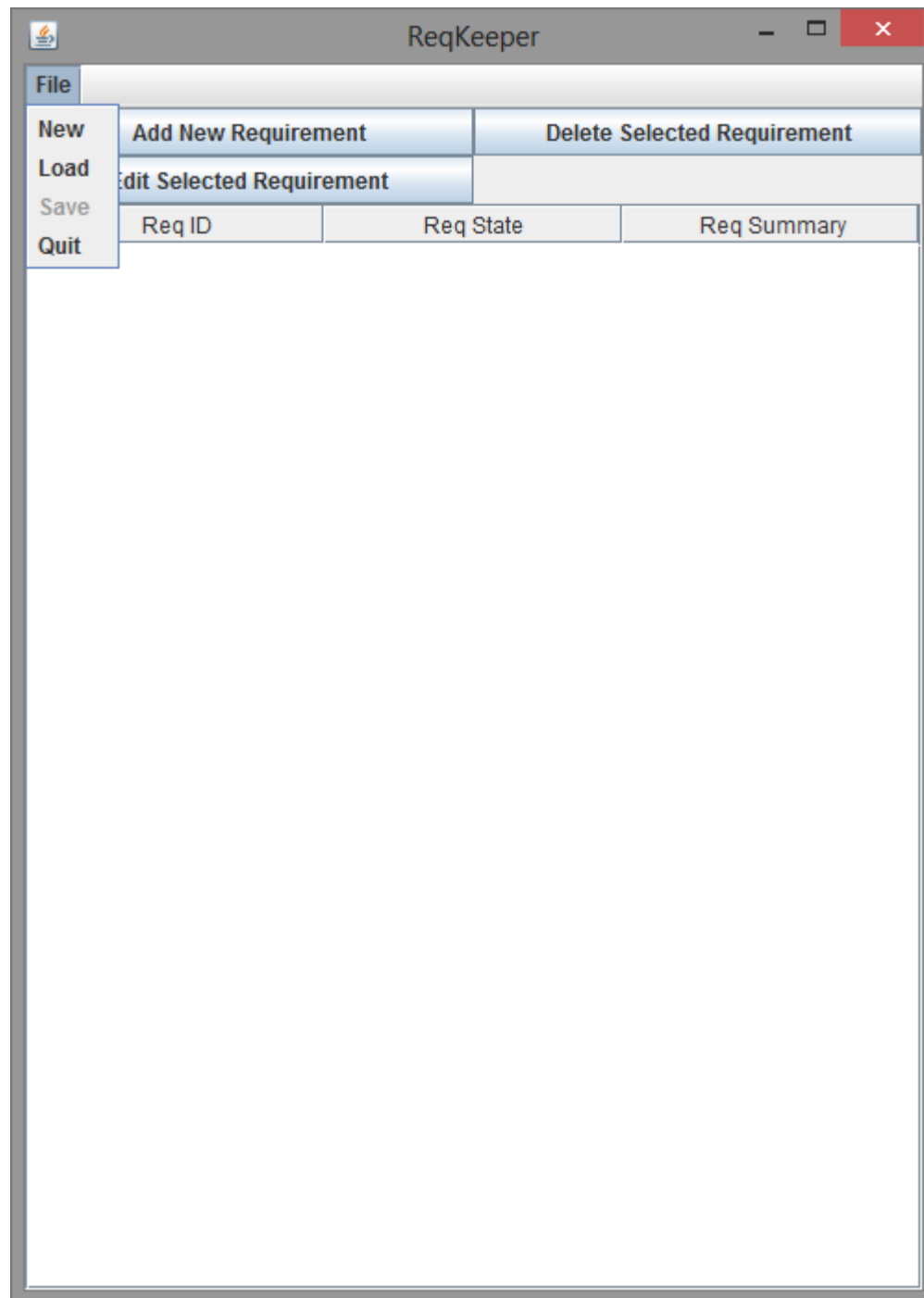


Figure 2: ReqKeeper GUI with File menu showing options described in UC1.

Main Flow: A user selects an option from the file menu: create a new requirement file [S1], load a requirement file [S2], save a requirement file [S3], and quit the application [S4]. A new, empty requirements file [S1], is created when the GUI is first loaded.

Subflows:

[S1] The user selects **New** from the file menu. A new, empty list of requirements, is created [UC2].

[S2] The user selects **Load**. A file chooser opens, where the user browses for the appropriate file. The file must conform to the standards listed below. These standards are encoded in the [RequirementTrackerXML 3rd party library](#) that you will be provided as part of the project. The library contains a [RequirementReader](#) class that processes requirement XML files. The library contains a [RequirementWriter](#) class that writes XML files. The requirements read from the file are listed on the main application window [UC2].

The RequirementReader will throw an exception if any of the following standards are violated when processing a requirement XML file [E1]:

- The file matches the [req_tracker.xsd](#) schema.

- The requirement id must be greater than or equal to 0.

- The requirement's state cannot be null.

- The requirement's summary cannot be null.

- The requirement's acceptance test id cannot be null.

- The requirement's priority cannot be less than 0 or greater than 3.

- If the requirement is in the Accepted, Working, Completed, or Verified states, it must have a non-null estimate and a non-zero priority.

- If the requirement is in the Working, Completed, or Verified states, it must have a non-null developer.

- The requirement cannot be in the Rejected state with a null rejection reason.

- If the requirement is in the Submitted or Rejected states, the priority is 0.

[S3]The user selects **Save**. This option is available only if there is an active requirement list for the project. Otherwise, the option is not enabled. A file chooser opens, where the user browses for the name of the file to save to or creates a new file. The RequirementWriter class from the RequirementTrackerXML library writes the XML file. The RequirementWriter will throw an exception if the file cannot be written due to an XML problem or a file permissions/quota problem [E2].

[S4] The user selects **Quit**. A file chooser opens, where the user browses for the name of the file to save to or creates a new file. The RequirementWriter class in the RequirementTrackerXML library writes the XML file. The RequirementWriter will throw an exception if the file cannot be written due to an XML problem or a file permissions/quota problem [E2]. The program exits if the file is successfully written.

Alternative Flows:

[E1] If the file cannot be loaded, a dialog opens with the message “Unable to load requirement file.” The user clicks **OK** and is returned to the main application window [UC1].

[E2] If the file cannot be saved, a dialog opens with the message “Unable to save requirement file.” The user clicks **OK** and is returned to the main application window [UC2].

Use Case 2 (UC2): Requirement List

Precondition: A user has selected a requirement list to display [UC1]. The requirement listing is shown in Figure 3.

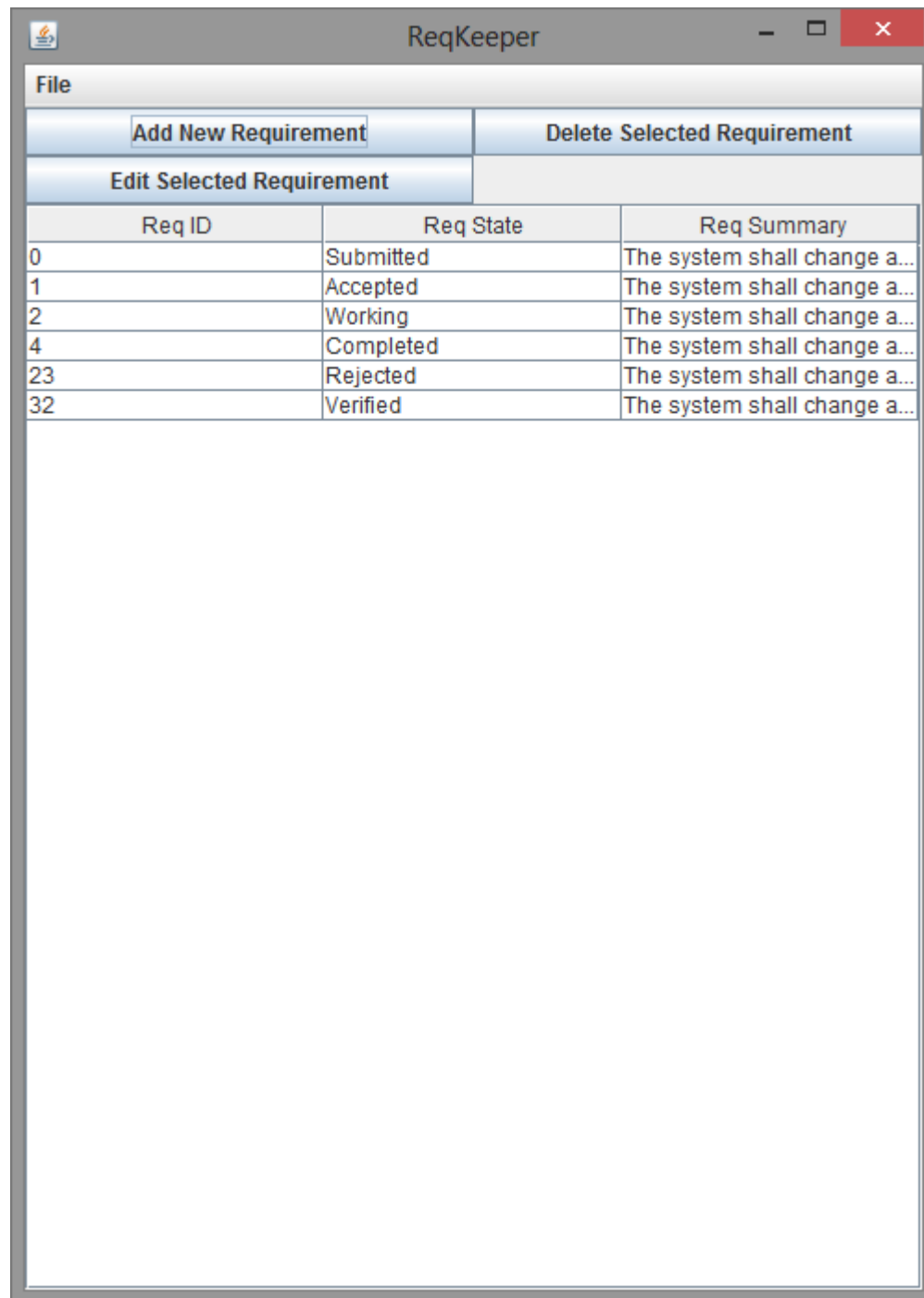


Figure 3: ReqKeeper GUI with a listing of 6 requirements read in from an XML file.

Main Flow: A user can add a new requirement [S1], delete the selected requirement [S2], or edit the selected requirement [S3].

Subflows:

[S1] To enter a new requirement in the system, the user clicks **Add New requirement**. A window opens for the user to enter the requirement summary and an acceptance test id (for subsequent test tracking in a separate system). The user enters the information and clicks **Add** to complete the add activity. When the requirement is added to the system, it is assigned a unique requirement id which is 1 + the greatest requirement id already used. The new requirement enters the Submitted state. If the user clicks **Cancel** instead, no requirement is added. In either case, the user is returned to the requirement list, which is shown in the main application window [UC1].

[S2] To remove an existing requirement, the user selects the requirement from the list and clicks **Delete Selected requirement** [E1]. The requirement is removed from the list.

[S3] To change an existing requirement, the user selects the requirement from the list and clicks **Edit Selected requirement** [E1]. The requirement's information is displayed (id, state, acceptance test id, developer, and so on) but cannot be directly edited. Different buttons are displayed depending on the state of the requirement:

Submitted [[UC3](#)]

Accepted [[UC4](#)]

Working [[UC5](#)]

Completed [[UC6](#)]

Verified [[UC7](#)]

Rejected [[UC8](#)]

Alternative Flows:

[E1] If a requirement is not selected, a dialog opens with the message "No requirement selected." The user clicks **OK** and is returned to the list, which remains unchanged.

Use Case 3 (UC3): Submitted State

Precondition: A user has selected a requirement to edit that is in the Submitted state [UC2, S3]. The user interface for editing a requirement in the Submitted state is shown in Figure 4.

ReqKeeper

File

Requirement Id Requirement State

Requirement Summary

ies an estimate of the time to implement the requirement and clicks the Accent button

Acceptance Test ID

Priority Estimate

Developer Rejection Reason

Priority

Estimate

Rejection Reason

Accept Reject Cancel

Figure 4: REQKEEPER GUI showing the user interface for interacting with a requirement in the Submitted state.

The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can accept the requirement [S1] which requires filling in a priority and an estimate for the requirement's implementation, or a user can reject the requirement [S2], or cancel the edit action [S3].

Subflows:

[S1] The user enters a value between 1 and 3, inclusive, for the priority and an estimate of the time it will take to complete the implementation of the requirement. The user clicks **Accept** [E1][E2]. The requirement's state is updated to Accepted. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user chooses a rejection reason then clicks **Reject**. The requirement's state is updated to Rejected. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S3] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing is not changed.

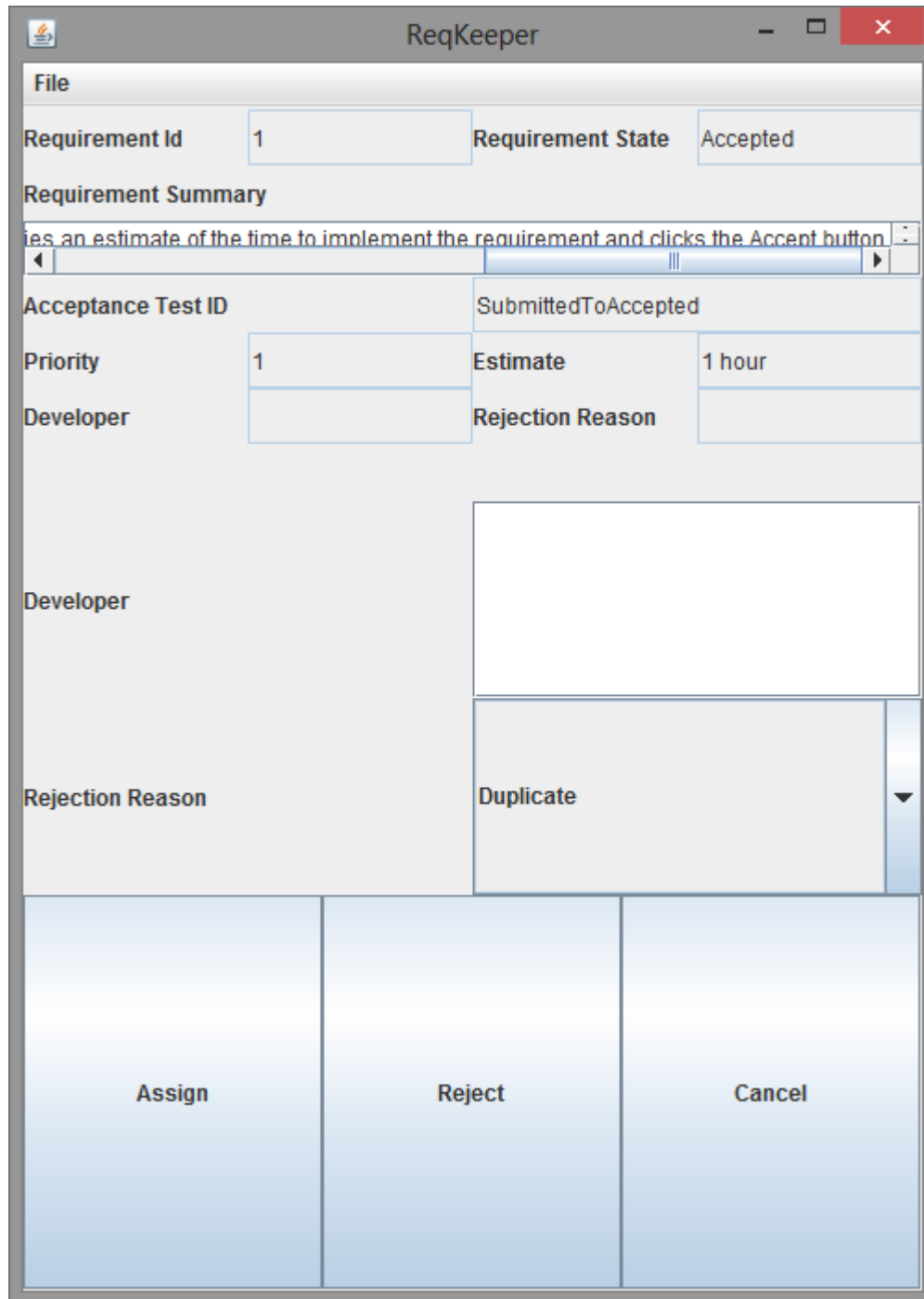
Alternative Flows:

[E1] If the field for the priority is left empty or is not a number, a dialog opens with the message "Invalid priority". The user clicks **OK** and is returned to the requirements list.

[E2] If the field for the priority is less than 1 or greater than 3 or if the estimate is left empty, a dialog opens with the message "Invalid command". The user clicks **OK** and is returned to the requirements list.

Use Case 4 (UC4): Accepted State

Precondition: A user has selected a requirement to edit that is in the Accepted state [UC2, S3]. The user interface for editing a requirement in the Accepted state is shown in Figure 5.



The ReqKeeper application window displays a form for managing requirements. The form includes fields for Requirement Id, Requirement State, Requirement Summary, Acceptance Test ID, SubmittedToAccepted, Priority, Estimate, Developer, and Rejection Reason. The Requirement Id is set to 1, and the Requirement State is Accepted. The Requirement Summary field contains the text "ies an estimate of the time to implement the requirement and clicks the Accent button". The Acceptance Test ID is SubmittedToAccepted. The Priority is 1, and the Estimate is 1 hour. The Developer field is empty. The Rejection Reason field is set to Duplicate. At the bottom of the window, there are three buttons: Assign, Reject, and Cancel.

File	
Requirement Id	1
Requirement State	Accepted
Requirement Summary	
ies an estimate of the time to implement the requirement and clicks the Accent button	
Acceptance Test ID	SubmittedToAccepted
Priority	1
Estimate	1 hour
Developer	
Rejection Reason	Duplicate
Assign Reject Cancel	

Figure 5: REQKEEPER GUI showing the user interface for interacting with a requirement in the Accepted state.
The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can assign the requirement to a developer [S1], reject the requirement [S2], or cancel the edit action [S2].

Subflows:

[S1] The user enters the name of the developer who will implement the requirement and clicks **Assign** [E1]. The requirement's state is updated to Working. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user chooses a rejection reason then clicks **Reject**. The requirement's state is updated to Rejected. The user is returned to the requirement list [UC2] and the requirement's listing is updated. The rejection reason is saved.

[S3] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing is not changed.

Alternative Flows:

[E1] If the field for the developer id is left empty, a dialog opens with the message "Invalid command" The user clicks **OK** and is returned to the Accepted state user interface to enter the developer id.

Use Case 5 (UC5): Working State

Precondition: A user has selected a requirement to edit that is in the Working state [UC2, S3]. The user interface for editing a requirement in the Working state is shown in Figure 6.

ReqKeeper

File

Requirement Id

2

Requirement State

Working

Requirement Summary

ies an estimate of the time to implement the requirement and clicks the Accent button

Acceptance Test ID

SubmittedToAccepted

Priority

1

Estimate

1 hour

Developer

excellent 216 student!

Rejection Reason

Rejection Reason

Duplicate

Complete

Reject

Cancel

Figure 6: REQKEEPER GUI showing the user interface for interacting with a requirement in the Working state.
The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can complete requirement [S1], reject the requirement [S2], or cancel the edit action [S3].

Subflows:

[S1] The user clicks **Complete**. The requirement's state is updated to Completed. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user chooses a rejection reason then clicks **Reject**. The requirement's state is updated to Rejected. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state, and the reason for rejection is stored.

[S3] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing is not changed. The rejection reason is not saved.

Use Case 6 (UC6): Completed State

Precondition: A user has selected a requirement to edit that is in the Completed state [UC2, S3]. The user interface for editing a requirement in the Completed state is shown in Figure 7.

ReqKeeper

File

Requirement Id Requirement State

Requirement Summary

Acceptance Test ID

Priority Estimate

Developer Rejection Reason

Developer

Rejection Reason

Pass Fail Assign Reject Cancel

Figure 7: REQKEEPER GUI showing the user interface for interacting with a requirement in the Completed state.
The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can verify the requirement passed the test [S1], indicate that the requirement failed the test [S2], assign the requirement to a new developer [S3], reject the requirement [S4], or cancel the edit action [S5].

Subflows:

[S1] The user has tested the requirement and clicks **Pass** to certify the requirement passes. The requirement's state is updated to Verified. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user has tested the requirement and the test has failed. The user clicks **Fail**. The requirement transitions to the Working state. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S3] The user enters a developer's id and clicks **Assign** [E1]. The requirement transitions to the Working state. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S4] The user chooses a rejection reason then clicks **Reject**. The requirement transitions to the Rejected state. The user is returned to the requirement list [UC2] and the requirement's listing is updated to the new state, and the reason for rejection is stored.

[S5] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing is not changed.

Alternative Flows:

[E1] If the field for the developer id is left empty, a dialog opens with the message "Invalid command" The user clicks **OK** and is returned to the Accepted state user interface to enter the developer id.

Use Case 7 (UC7): Verified State

Precondition: A user has selected a requirement to edit that is in the Verified state [UC2, S3]. The user interface for editing a requirement in the Verified state is shown in Figure 8.

ReqKeeper

File

Requirement Id: 32 Requirement State: Verified

Requirement Summary

ies an estimate of the time to implement the requirement and clicks the Accent button

Acceptance Test ID: SubmittedToAccepted

Priority: 1 Estimate: 1 hour

Developer: excellent 216 student! Rejection Reason:

Developer:

Rejection Reason: Duplicate

Assign Reject Cancel

Figure 8: REQKEEPER GUI showing the user interface for interacting with a requirement in the Verified state.

The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can assign a new developer to the requirement [S1], reject the requirement [S2], or cancel the edit action [S3].

Subflows:

[S1] The user enters a developer's id and clicks **Assign** [E1]. The requirement transitions to the Working state. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user chooses a rejection reason then clicks **Reject**. The requirement transitions to the Rejected state. The user is returned to the requirement list [UC2] and the requirement's listing is updated to the new state, and the reason for rejection is stored.

[S3] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing has not changed.

Alternative Flows:

[E1] If the field for the developer id is left empty, a dialog opens with the message "Invalid command" The user clicks **OK** and is returned to the Accepted state user interface to enter the developer id.

Use Case 8 (UC8): Rejected State

Precondition: A user has selected a requirement to edit that is in the Rejected state. The user interface for editing a requirement in the Rejected state is shown in Figure 10.

ReqKeeper

File

Requirement Id

23

Requirement State

Rejected

Requirement Summary

ies an estimate of the time to implement the requirement and clicks the Accent button

Acceptance Test ID

SubmittedToAccepted

Priority

0

Estimate

Developer

Rejection Reason

Duplicate

Summary

ie requirement and clicks the Accept button.

Acceptance Test ID

SubmittedToAccepted

Revise

Cancel

Figure 9: REQKEEPER GUI showing the user interface for interacting with a requirement in the Rejected state.
The top part of the GUI shows the requirement information, the lower part of the GUI shows how the user can interact with the requirement.

Main Flow: A user can indicate that the requirement should now be revised [S1] or cancel the edit action [S2].

Subflows:

[S1] The user edits the summary and acceptance test ID, if needed. The user clicks **Revise**, which transitions the requirement to the Submitted state. The user is returned to the requirement list [UC2] and the requirement's listing reflects the updated state.

[S2] The user clicks **Cancel**. The user is returned to the requirement list [UC2] and the requirement's listing has not changed.

Design

Trying to jump from requirements right into coding without doing some design first is bound to lead to trouble. You will eventually implement our design, but first you need to propose your own. To do this, we give you a scenario and insist on some restrictions for your design.

What you must do: Design Rationale and UML diagram

You must design an application that satisfies the requirements of the Requirements Management application. You will create a design proposal that includes classes suggested by the requirements. Your design must be described in document containing a design rationale and a UML class diagram.

Your design should:

- utilize the Model-View-Controller (MVC) design pattern (see the note about MVC, below).
- contain at least one interface *or* abstract class.
- contain at least one *"is-a"* or inheritance relationship.
- contain at least one *"has-a"* or composition relationship that can be modeled in your class diagram using an association (arrow), aggregation (open diamond), or composition (solid diamond) connector.

Answer the following technical questions in your design document as part of the rationale:

1. What objects are important to the system's implementation and how do you know they are important?
2. What data are required to implement the system and how do you know these data are needed?
3. Are the responsibilities assigned to an object appropriate for that object's abstraction and why?
4. What are the relationships between objects (such as inheritance and composition) and why are those relationships are important?

5. What are the limitations of your design? What are the constraints of your system?

MVC Note

Java Swing, the user interface (UI) libraries for Java, does not follow the strictly traditional definition of MVC. Instead, Java Swing utilizes what might be called a [separable model architecture](#). This means that the model is separate and distinct from the view/controller classes that make up the UI. For your design, you will focus on the Model. In your UML class diagram, you should represent the UI as a class with no state or behavior. Your diagram should also show which class(es) your UI will interact with through some type of composition/aggregation/association relationship. The relationship between your model and the UI must be justified in your design rationale.

When thinking about the relationships between your UI and the model, consider the following questions:

1. What are the data and behaviors of your model that will be shown through the UI?
2. How does your UI get those data to display; what methods of the model must be called?

State Pattern

The State Pattern is an object-oriented solution to a state-based application. The finite-state machine that models the state of a requirement should be modeled using the State Pattern. This means the bubbles, which represent states, becomes classes. The transitions become the behaviors of the classes. A context class encapsulates the state pattern for each requirement and delegates the transitions to the current state for the given requirement. For more information on the State Pattern, see the [Wikipedia article](#) and the example [Horner's Rule State Pattern implementation](#).

Submitting your work

You must submit your design proposal via your section's submission platform as a PDF. Use the [provided design proposal template](#) as a starting point for your design proposal. Use a UML diagramming tool, like [Dia](#), to create your UML diagram. You may also hand draw your UML diagram. In either case you submit your UML diagram (either scan it in or create an electronic image of it) along with your design proposal for credit. Make sure that your diagram is readable! Name your design proposal document **DesignProposal_P1P1.pdf**. If you have a separate file for your UML diagram, name the file **DesignUML_P1P1.***, where * is the extension of your diagram file (pdf/gif/jpg/png are accepted file formats).

Need a little more direction?

See the following example design proposal: [Sample Design Proposal](#).

Implementation

Explicit implementation details will be available for the second part of this project.

Testing

This project requires you to do white box and black box testing. You can defer white box testing until Part 2 of this project. But now, you need to prepare some black box test cases to clarify the inputs to and outputs from the system. Each test must demonstrate that the program satisfies a scenario from the requirements. A scenario is one major path of functionality as described by the requirements.

Assignment

You will write at least five (5) tests for the REQKEEPER project. Use the [provided black box test plan template](#) to describe your tests. Each test must be repeatable and specific; all input and expected results values must be concrete. All inputs required to complete the test must be specified. Additionally, you must provide instructions for how a tester would set up, start, and run the application for testing (what class from your design contains the main method that starts your program?). The instructions should be described at a level where anyone using Eclipse could run your tests.

Test Files

There are a [set of test files](#) that you may use in your tests. Note that the files are written in XML - you will be provided a library for reading and writing XML files in Part 2. You can read the contents of the files to understand what the elements of the listed requirements are.

Format

Submit your design proposal via Moodle under the assignment named **P1P1: Design**.

Use the [provided design proposal template](#) as a starting point for your design proposal. Use a UML diagramming tool (options are listed the [Software Development Practices notes](#)) to create your UML diagram. Incorporate your UML diagram into your written proposal (using an editing tool such as MS Word) and save the entire document as a PDF. Alternatively, create a PDF for the design proposal and another PDF for the UML diagram, then append the diagram to the proposal to make a *single* PDF document.

Need a little help?

See the following example black box test plan: [Sample Black Box Test Plan](#)

Deployment

For this class, deployment means submitting your work for grading. Before you submit your work, make sure your work satisfies the [gradesheet](#) for this assignment.

For Part 1 of this programming assignment, you must submit two (or three) documents:

1. Design document (pdf). Name your design proposal document **DesignProposal_P1P1.pdf**.
2. UML diagram (may be incorporated into the Design document or may be a separate file). If you have a separate file for your UML diagram, name the file **DesignUML_P1P1.***, where * is the extension of your diagram file. pdf/gif/jpg/png are accepted file formats.
3. Black box test plan document (pdf). Your black box test plan document must be named **BBTP_P1P1.pdf**.

You should submit the documents for Project 1 Part 1 to Moodle.

Submission Reminders

The electronic submission deadline is precise. Do not be late. You should count on last minute system failures (your failures, ISP failures, or Moodle failures). You are able to make multiple submissions of the same file. (Later submissions to a Moodle assignment overwrite the earlier ones.) To be on the safe side, start submitting your work as soon as you have completed a substantial portion.

[Project 1, Part 2: Scrum Backlog](#) ►