CSC216: Project 1

Project 1, Part 1: Scrum Backlog

Project 1, Part 1: Scrum Backlog ▶

Project 1, Part 1: Scrum Backlog

Version with no table of contents (for printing).

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

Deliverables and Deadlines

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

Part 1 Due Date: Friday, February 15, 2019 at 11:45 PM

Late Deadline (Design): Saturday, February 16, 2018 at 9:00 AM

Late Deadline (BBTP): Sunday, February 17, 2019 at 11:45 PM

Reminder

You will *not* be working with a partner for Part 1 of *any* project. All work must be strictly your own.

Problem overview

The CSC Senior Design Center (SDC) provides the opportunity for students to work on small teams to create a software solution for an industry-sponsored project. Students must keep track of their requirements (what the system should do) and tasks (what they should do). The SDC staff have requested that you design and develop a system to maintain a backlog of tasks to support a software development process called *Scrum*.

Scrum is an agile and iterative software development process that is commonly used in industry. One of the important concepts in Scrum is the **backlog** of tasks that need to be completed. These tasks may be 1) features that need to be implemented, 2) bugs that need to be fixed, 3) technical work like refactoring, or 4) knowledge acquisition to complete other tasks. As tasks are created, they are added to a backlog. Team members then claim the tasks that they will work on. Only one task is worked on at a time and the task may require verification by another team member before the task is completed. This simplified version of the Scrum backlog process can be modeled using a Finite State Machine.

For this project, you must implement a Scrum task backlog that uses the FSM shown Figure 1. (The state names are too long to include in the diagram. Additionally, the transitions are too detailed to fully label in the diagram. Instead each transition is given a name corresponding to the state where it starts. Descriptions of the transitions follow below.)

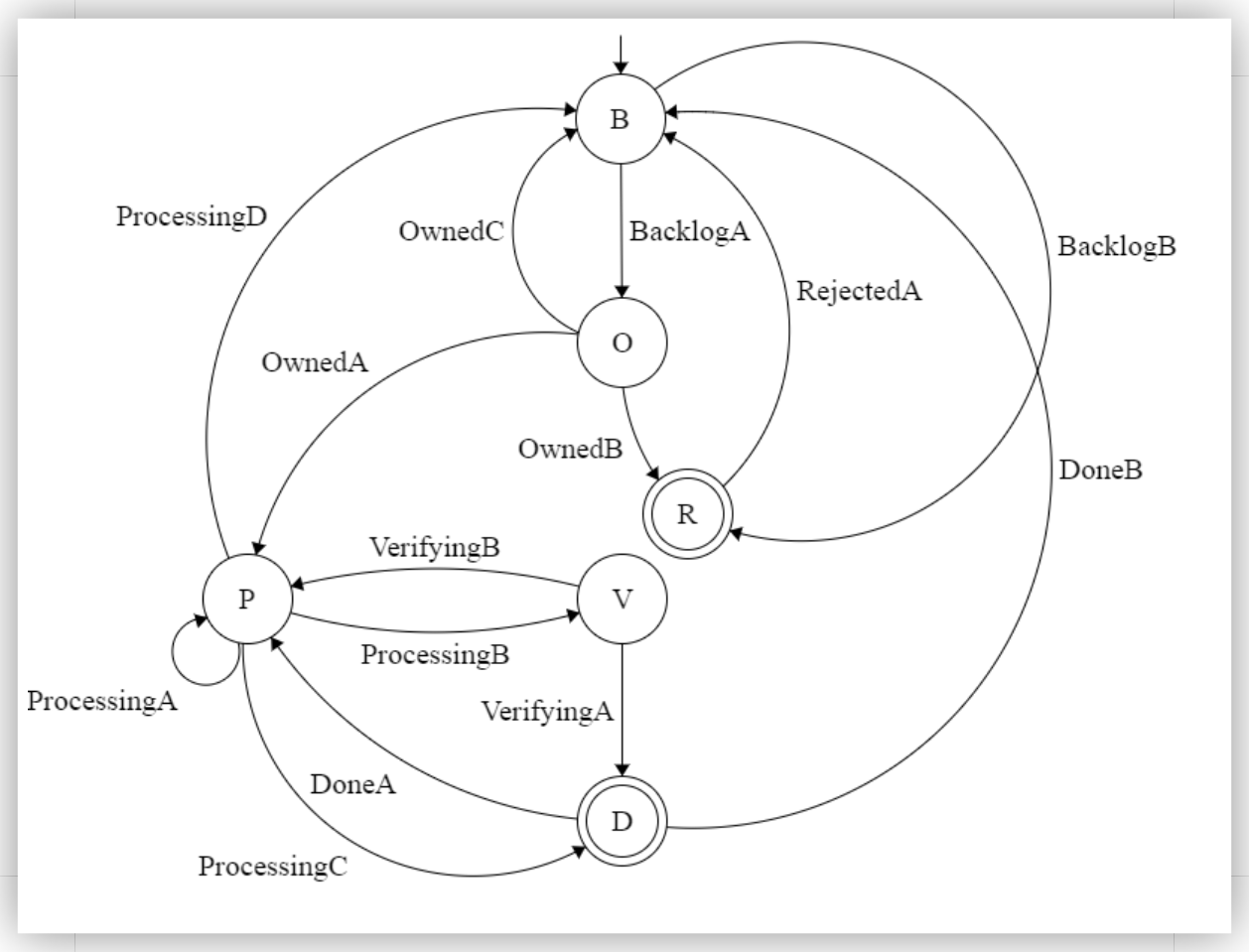

Figure 1: Scrum Backlog FSM.

A task in the backlog system can be in one of the following states: **Backlog**, **Owned**, **Processing, Verifying**, **Done, Rejected**. The meanings of these states and their corresponding transitions are described as follows.

Backlog. Tasks enter the backlog system into the Backlog state. Each Task has an id, title, type, and creator. Details of the task are recorded as the first note in a list of notes. Tasks in Backlog have not yet been claimed, so they have no owners. Transition out of the Backlog state occurs when the task is claimed by an owner who will complete the task or when the team rejects the task as out of scope.

BacklogA. Transition from Backlog to Owned. The owner's user id is associated with the task and identifies the person responsible for completing the task. The task retains that same owner unless the task is returned to the Backlog to be claimed by a new owner.

BacklogB. Transition from Backlog to Rejected. The team has rejected the task as out of scope.

Owned. Tasks in Owned have owners. In this state, the owner starts working on the task (which changes the state to Processing), rejects the task (which changes the state to Rejected), or returns the task to the Backlog.

OwnedA. Transition from Owned to Processing. The owner is working on the task.

OwnedB. Transition from Owned to Rejected. The owner has rejected the task as not appropriate for the project. The owner is removed.

OwnedC. Transition from Owned to Backlog. The owner returns the task to the backlog. The owner is removed.

Processing. The owner is currently working on the task. There are four transitions:

ProcessingA. From Processing to Processing. The owner adds a note to the task indicating some progress or additional information about the task.

ProcessingB. From Processing to Verifying. The owner requests that a team member verifies the change associated with the task. This step is needed only for tasks that are features, bugs, or technical work.

ProcessingC. From Processing to Done. The owner completes a knowledge acquisition task. If the task is a feature, bug, or technical work, the task must be verified before it can move to Done.

ProcessingD. From Processing to Backlog. The owner moves the task back to the backlog to be claimed by another developer.

Verifying. An owner has requested that a team member verifies the change for a feature, bug, or technical work task.

VerifyingA. From Verifying to Done. The team member approves the change for a feature, bug, or technical work task and the task is completed.

VerifyingB. From Verifying to Processing. The team member does not approve the

change for a feature, bug, or technical work. The task requires more work by the owner.

Done. A final state. From here the task can be returned to the owner for additional work or can be returned to the backlog for another developer to work on.

DoneA. From Done to Processing. The task is returned to the owner for additional work.

DoneB. From Done to Backlog. The task is returned to the backlog for another developer to claim for additional work.

Rejected. A final state representing a task that has been rejected by the team as out of scope for a project. If that changes, the task may be returned to the backlog for a developer to work on.

RejectedA. From Rejected to Backlog. The task is returned to the backlog for a developer to work on.

Requirements

The remaining requirements for the Scrum Backlog will be modeled with use cases that will utilize the backlog state pattern described above. Use cases divide system behavior into related scenarios around a core piece of functionality. These scenarios tie directly to user experience with the GUI. The different paths that a user can take when interacting with the system translate into black box tests for the program.

Use Case 1: Task XML File Interactions

Preconditions: A user has started the Scrum Backlog application and the GUI (shown in Figure 2) has opened.

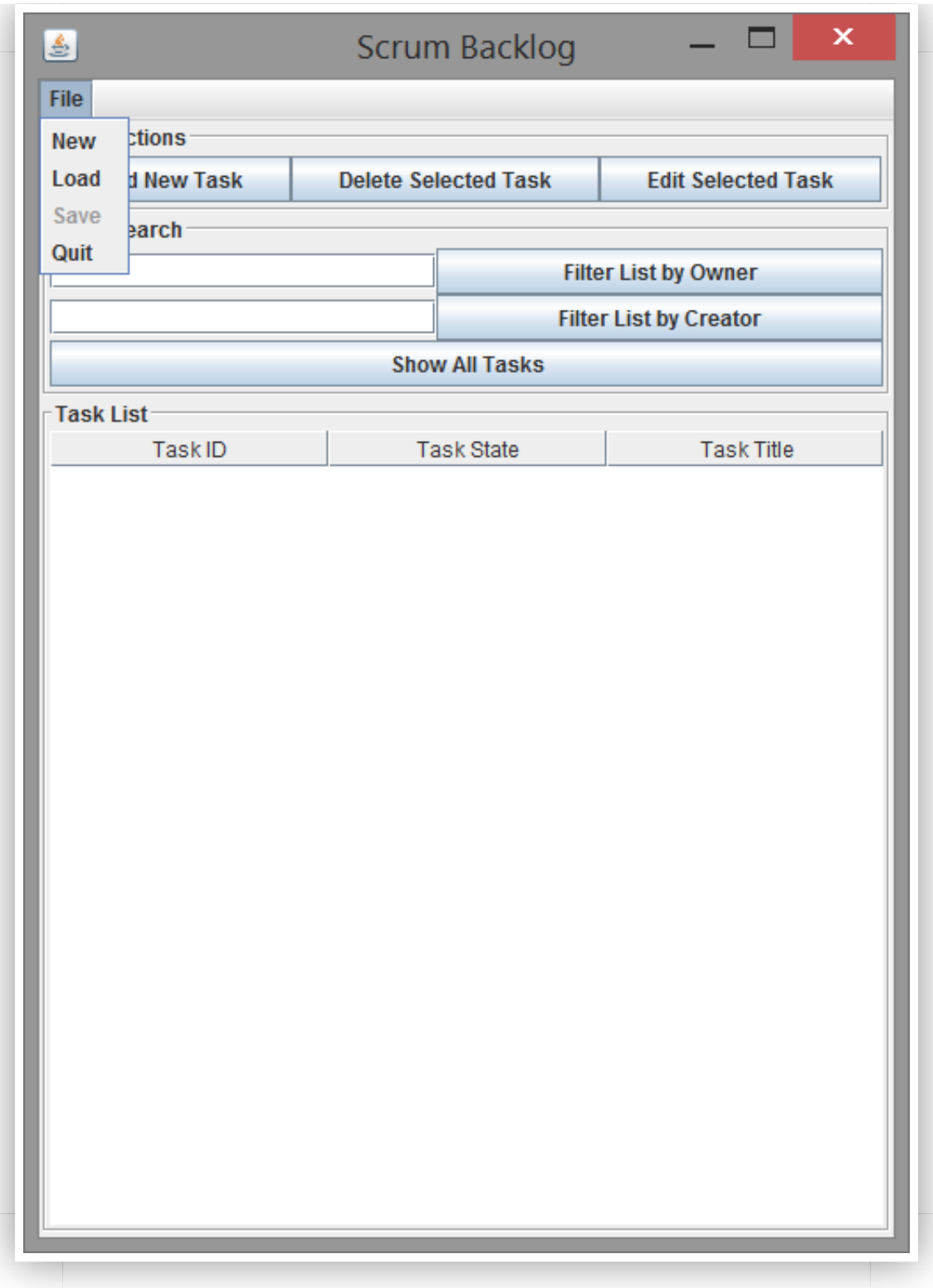

Figure 2: Scrum Backlog with File menu for interacting with Task XML files.

Main Flow: A user selects an option from the file menu: create a new task file [S1], load a task file [S2], save a task file [S3], or quit the application [S4].

Subflows:

[S1] The user selects **New** from the file menu. A new, empty list of tasks 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 TaskXML 3rd party library that you will be provided as part of the project. The library contains a TaskReader class that processes task XML files. The TaskReader class will throw an exception if any of the following standards are violated when processing a task XML file [E1]:

The file matches the task.xsd schema.
The task's id must be greater than or equal to 0.
The task's state cannot be null.
The possible states are "Backlog", "Owned", "Processing", "Verifying", "Done", and "Rejected".
The task's title cannot be null.
The task's type must be "F" for feature, "B" for bug, "TW" for technical work, or "KA" for knowledge acquisition.
The task's creator cannot be null or empty.
The task cannot be in the Owned, Processing, Verifying, or Done state without an owner.
The task with an owner cannot be in the Backlog or Rejected states.
A feature, bug, or technical work task cannot be in the Done state without verification of the change.
The task cannot be in any state without at least one note.

[S3] The user selects **Save**. This option is available only if there is an active task list. 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 TaskWriter class from the TaskXML library writes the XML file. The TaskWriter class 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 TaskWriter class in the TaskXML library writes the XML file. The TaskWriter class 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 task file." The user clicks OK and is returned to the ScrumBacklog application [UC1]. None of the tasks in the file are loaded into the system.
[E2] If the file cannot be saved, a dialog opens with the message "Unable to save task file." The user clicks OK and is returned to the ScrumBacklog application [UC2].

Use Case 2: Task List

Preconditions: A user has selected a task list to display [UC1]. The task listing is shown in

Figure 3.

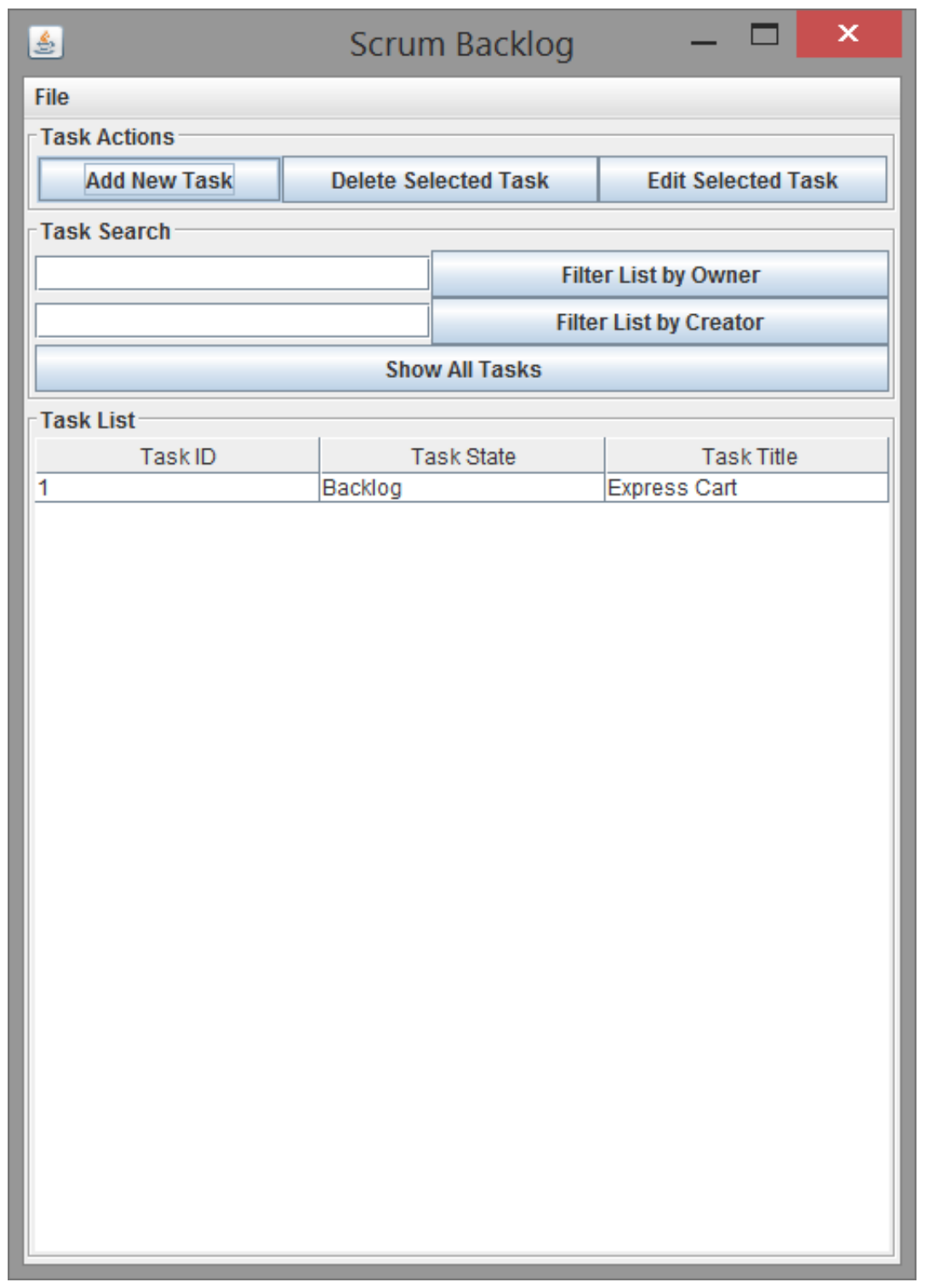

Figure 3: ScrumBacklog GUI with a listing of tasks read in from an XML file.

Main Flow: A user can add a new task [S1], delete the selected task [S2], edit the selected task [S3], filter the task list by a given owner [S4], filter the task list by a given creator [S5], and show all tasks in the list [S6].

Subflows:

[S1] To enter a new task in the system, the user clicks **Add New Task.** A window opens for the user to enter the task title; task type from options of feature, bug, technical work, and knowledge acquisition; the creator's id; and a note. See Figure 4 for an illustration. The user enters the information and clicks **Add Task to Backlog** to complete the add activity [E1]. When the task is added to the system, it is assigned a unique id which is 1 + the greatest id already used. If the task is the first in the list, the id is set to 1. The new task enters the Backlog state. If the user clicks **Cancel** instead, no task is added. In either case, the user is returned to the task list [UC2].

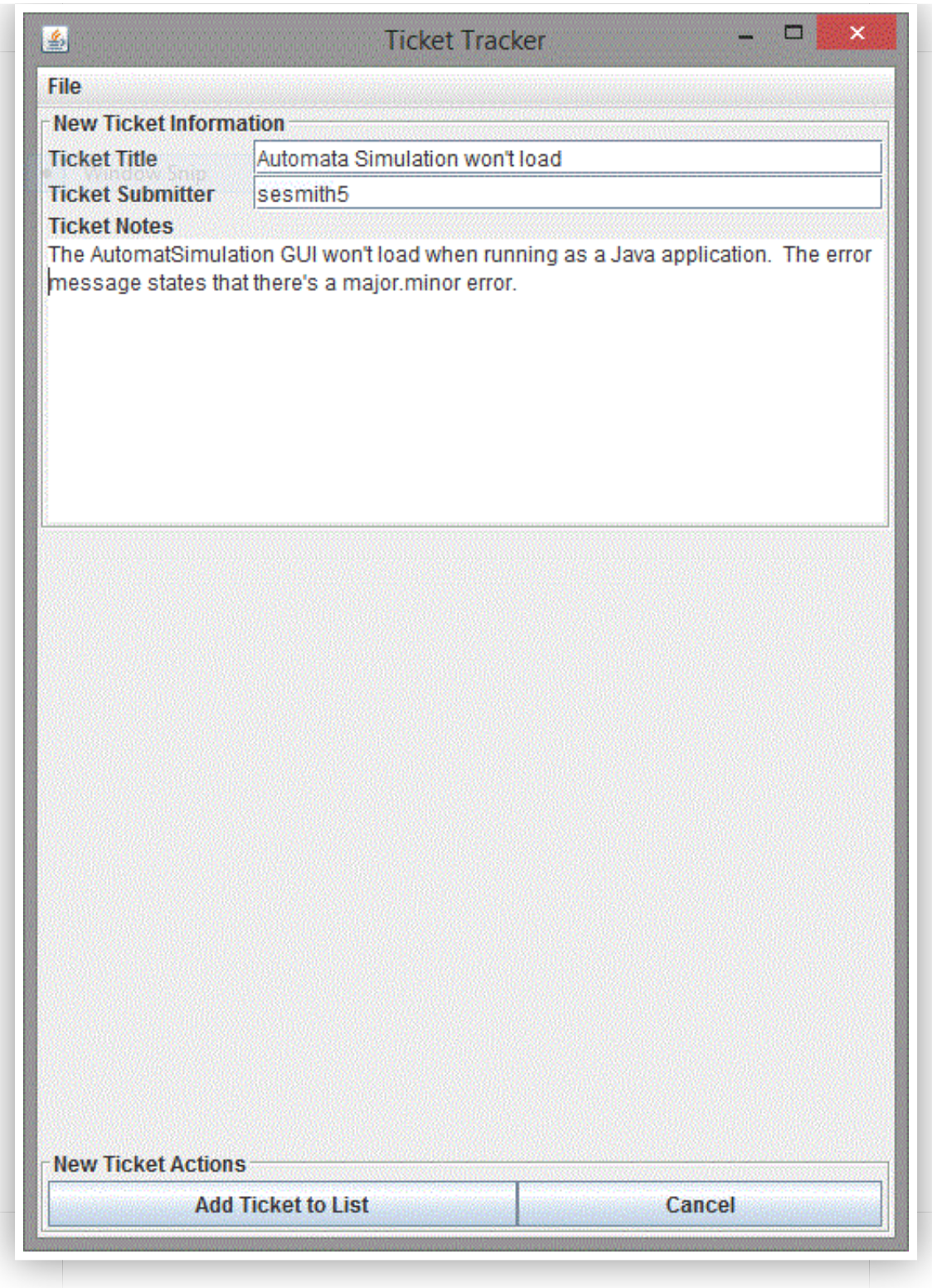

Figure 4: ScrumBacklog GUI showing functionality for adding a new task.

[S2] To remove an existing task, the user selects the task from the list and clicks **Delete Selected Task** [E2]. The task is removed from the list

[S3] To change an existing task, the user selects the task from the list and clicks **Edit Selected Task** [E2]. The task's information is displayed (id, state, title, creator, owner, type, and the notes list) but cannot be directly edited. Each of the task's notes are shown in a

table. Different buttons are displayed depending on the state of the task:

Backlog [UC3]
Owned [UC4]
Processing [UC5]
Verifying [UC6]
Done [UC7]
Rejected [UC8]

[S4] The user enters an owner id in the text field and clicks **Filter List by Owner.** The list display is refreshed to show only tasks owned by the the person with that user id. If the user id does not belong to any user who owns tasks, the list display is empty.
[S5] The user enters a creator id in the text field and clicks **Filter List by Creator.** The list display is refreshed to show only tasks created by the person with that user id. If the user id does not belong to any user who created tasks, the list display is empty.
[S6] The user clicks **Show All Tasks** to refresh the task list display to show all tasks.

Alternative Flows:

[E1] If the task title, type, creator id, or note are null, an empty string, or otherwise invalid, a dialog opens with the message, “Invalid task information.” The user clicks OK and is returns to the add task window.
[E2] If a task is not selected, a dialog opens with the message “No task selected.” The user clicks OK and is returned to the list with no changes.

Use Case 3: Backlog State

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

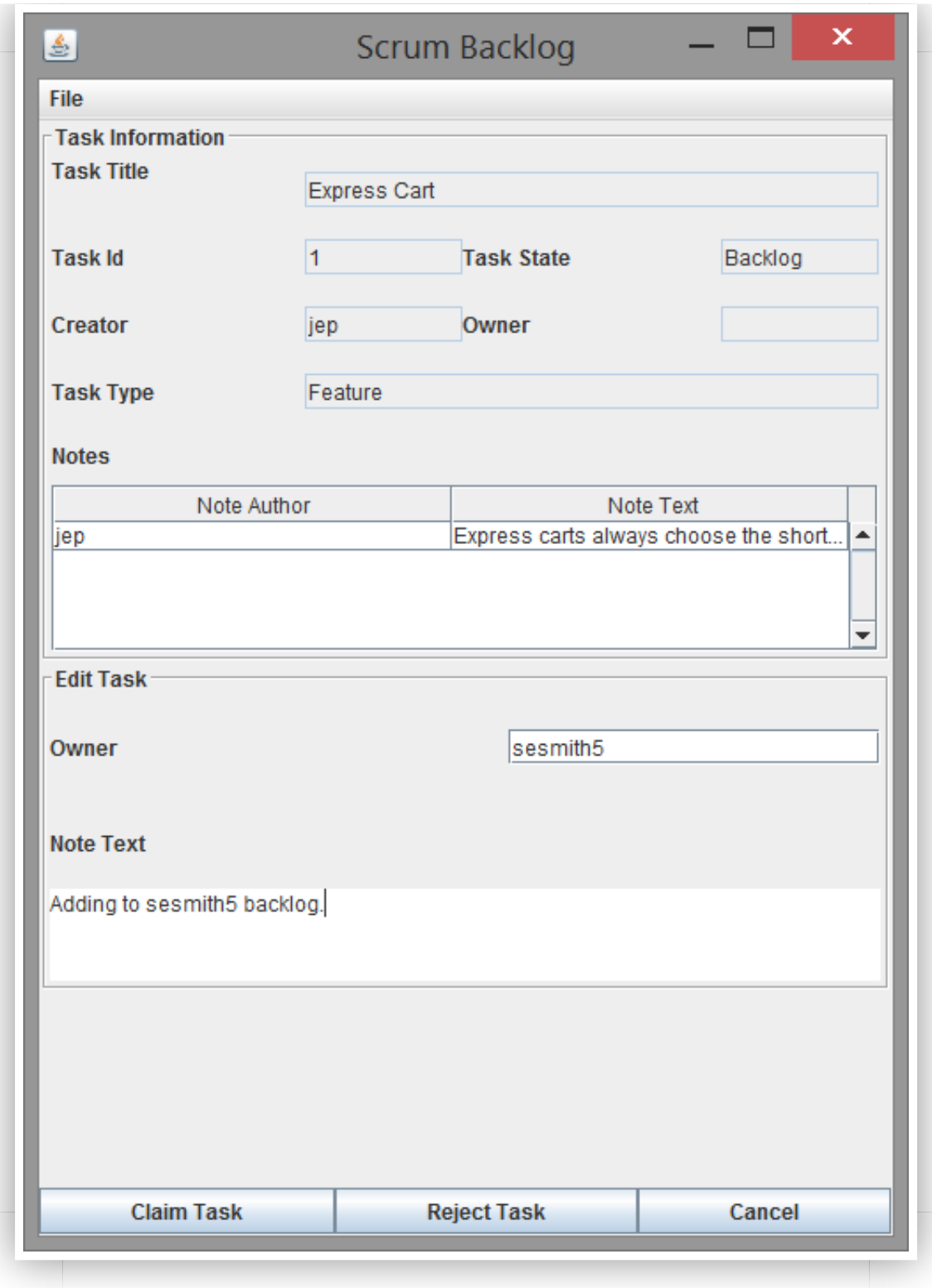

Figure 5: ScrumBacklog GUI showing the user interface for interacting with a task in the Backlog state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: A user can claim the task as an owner [S1], reject the task [S2], or cancel the edit action [S3]. The user must provide a note. The owner is the note's author.

Subflows:

[S1] The user enters the id of the owner who will complete the task, provides details in a note, and clicks **Claim Task** [E1][E2]. The task's state is updated to Owned. The note is saved with the task and the owner is the note author. The user is returned to the task list [UC2] and the task listing reflects the updated state for the task.
[S2] The user enters the id of the developer rejecting the task, provides details in a note, and clicks **Reject Task** [E1][E2]. The task's state is updated to Rejected. The note is saved with the task and the listed developer is the note author. The user is returned to the task list [UC2] and the task listing reflects the updated state for the task.
[S3] The user clicks **Cancel.**The user is returned to the task list [UC2] and the task listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the owner's id is left empty, a dialog opens with the message "Invalid owner id." The user clicks OK and is returned to the Backlog state user interface to enter the owner id.
[E2] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Backlog state user interface to enter the note text.

Use Case 4: Owned State

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

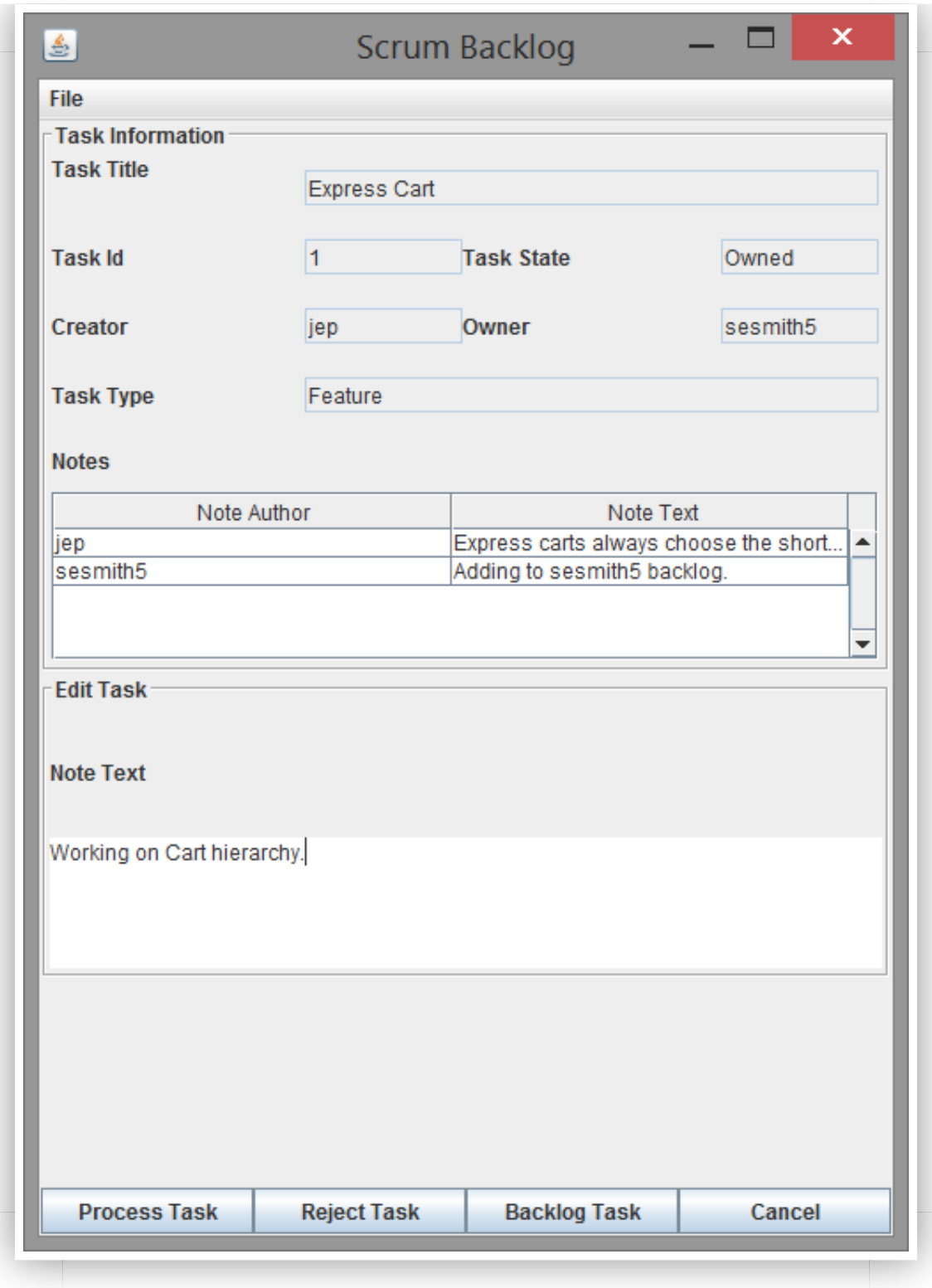

Figure 6: ScrumBacklog GUI showing the user interface for interacting with a task in the Owned state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: An owner can start processing the task [S1], reject the task as out of scope [S2], backlog the task for another developer to work on [S3], or cancel the edit action [S4]. The user must provide a note. The note's author is the owner.

Subflows:

[S1] The user provides details in a note and clicks **Process Task** [E1]. The task moves to the Processing state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S2] The user provides details in a note and clicks **Reject Task** [E1]. The task moves to the Rejected state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S3] The user provides details in a note and clicks **Backlog Task** [E1]. The task moves to the Backlog state. The note, with the owner as author, is saved with the task. The owner is then removed. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S4] The user clicks **Cancel.** The user is returned to the task list [UC2] and the task's listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Owned state user interface to enter the note text.

Use Case 5: Processing State

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

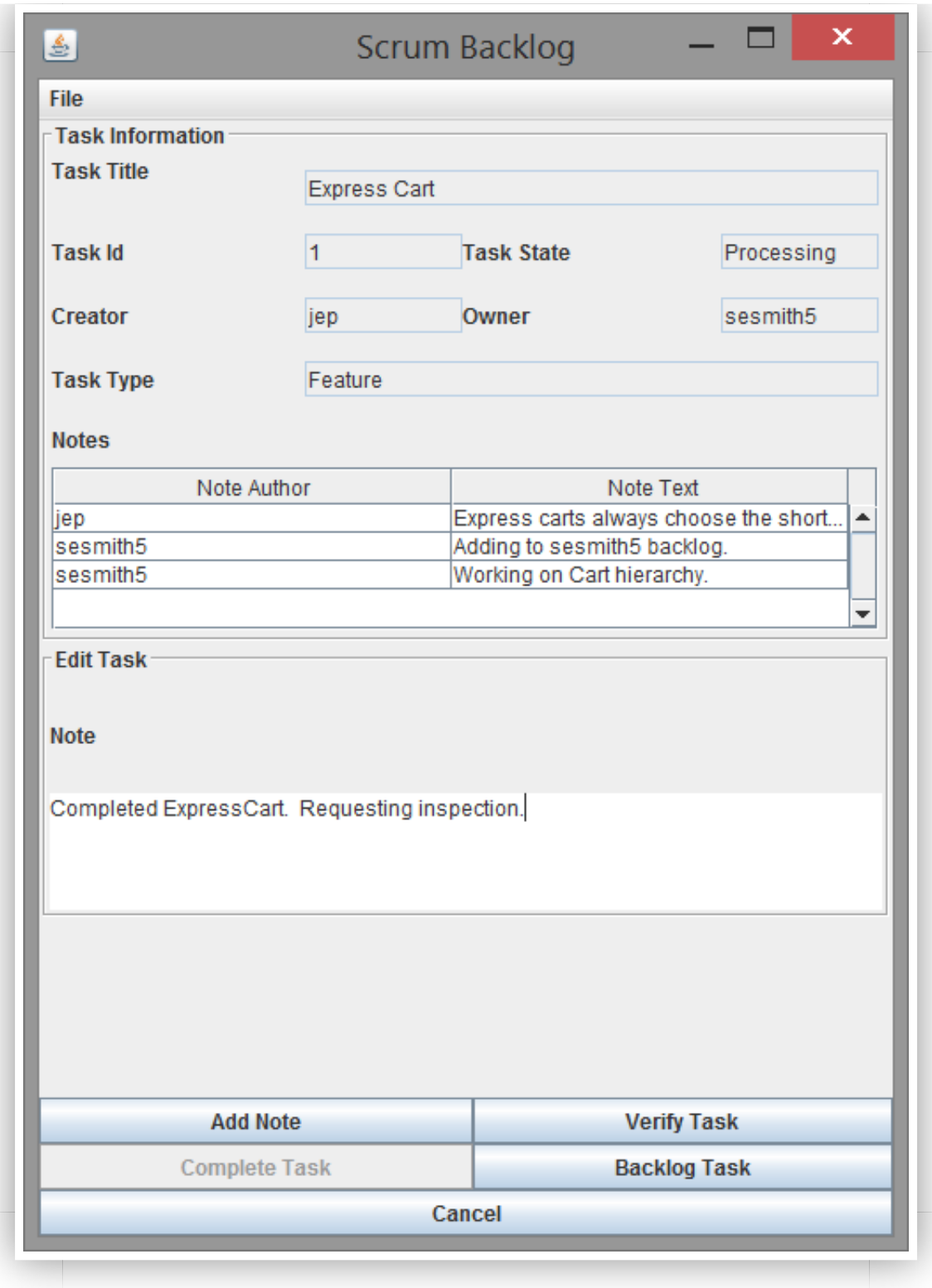

Figure 7: ScrumBacklog GUI showing the user interface for interacting with a task in the Processing state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: A user can add a note to the task with progress on completing the task [S1], request verification of a feature, bug, or technical work task [S2], complete a knowledge acquisition task [S3], return the task to the backlog [S4], or cancel the edit action [S5]. The

user must provide a note, which is authored by the owner.

Subflows:

[S1] The user provides details in a note and clicks **Add Note** [E1]. The task remains in the Processing state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S2] If the task is a feature, bug, or technical work task, the user may request that a completed task be verified by another team member and clicks **Verify Task** [E1]. The task moves to the Verifying state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S3] If the task is a knowledge acquisition task and the user has completed the task, the user clicks **Complete Task** [E1]. The task moves to the Done state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S4] The user can no longer work on the task and clicks **Backlog Task** [E1]. The task moves to the Backlog state and the owner is removed. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S5] The user clicks **Cancel.** The user is returned to the task list [UC2] and the tasks listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Processing state user interface to enter the not text.

Use Case 6: Verifying State

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

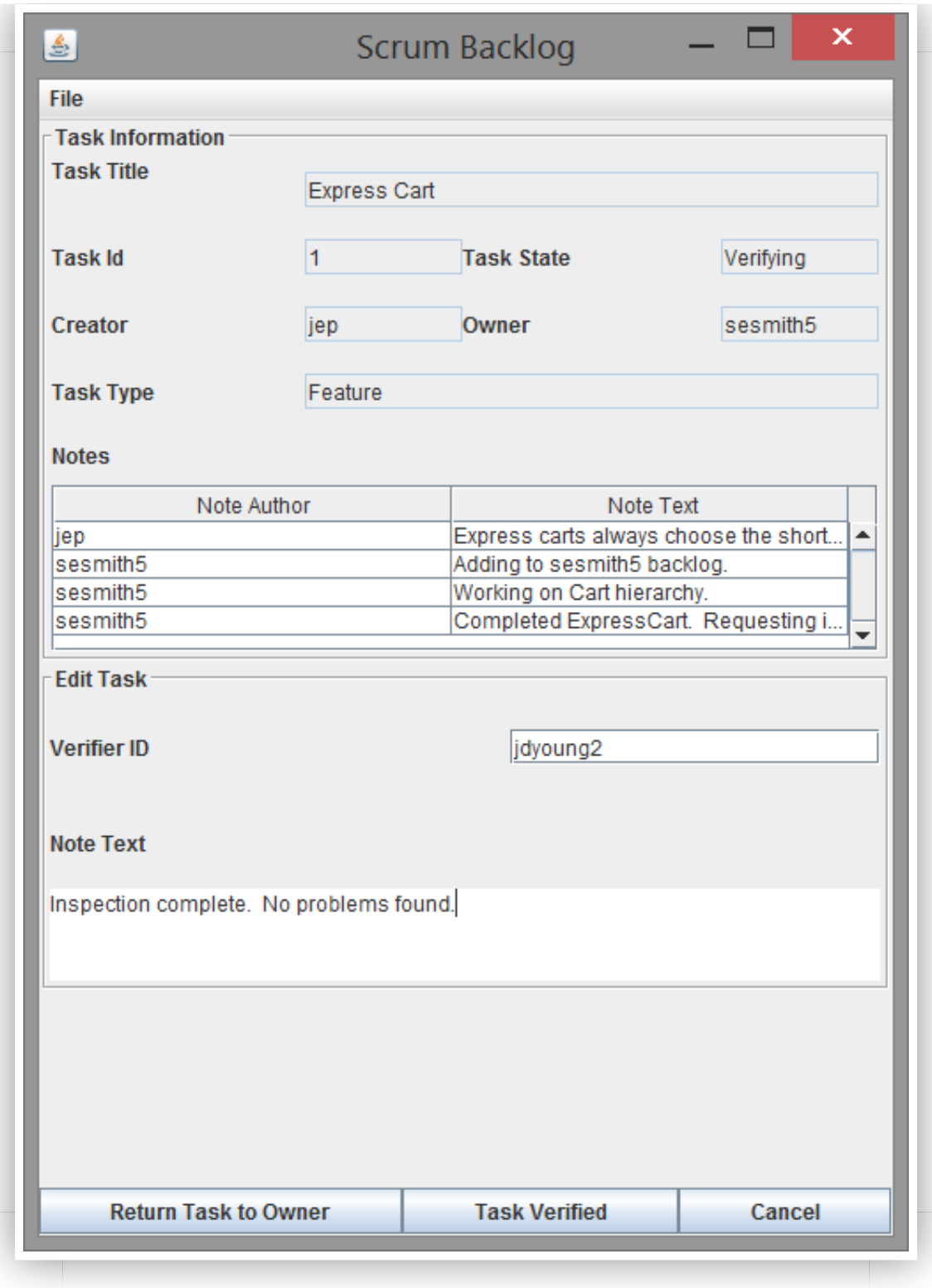

Figure 8: ScrumBacklog GUI showing the user interface for interacting with a task in the Verifying state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: A user can return the task to the owner as needing additional work [S1], verify the change for a feature, bug, or technical work task as correct [S2], or cancel the edit action [S3]. The user must provide a note. A note includes the note's author (as the verifying team

member) and text.

Subflows:

[S1] The user verifies the change and finds that more work needs to be completed. The user clicks **Return Task to Owner** [E1][E2]. The task moves to the Processing state. The note and verifier id are saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S2] The user verifies the change as correct and clicks **Task Verified** [E1][E2]. The task moves to the Done state. The note and verifier id are saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S3] The user clicks **Cancel.** The user is returned to the task list [UC2] and the task listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the Verifier ID is left empty or is the id of the owner, a dialog opens with the message "Invalid verifier id." The user clicks OK and is returned to the Verifying state user interface to enter the note author id.
[E2] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Verifying state user interface to enter the not text.

Use Case 7: Done State

Preconditions: A user has selected a task to edit that is in the Done state [UC2, S3]. The user interface for editing a task in the Done state is shown in Figure 9.

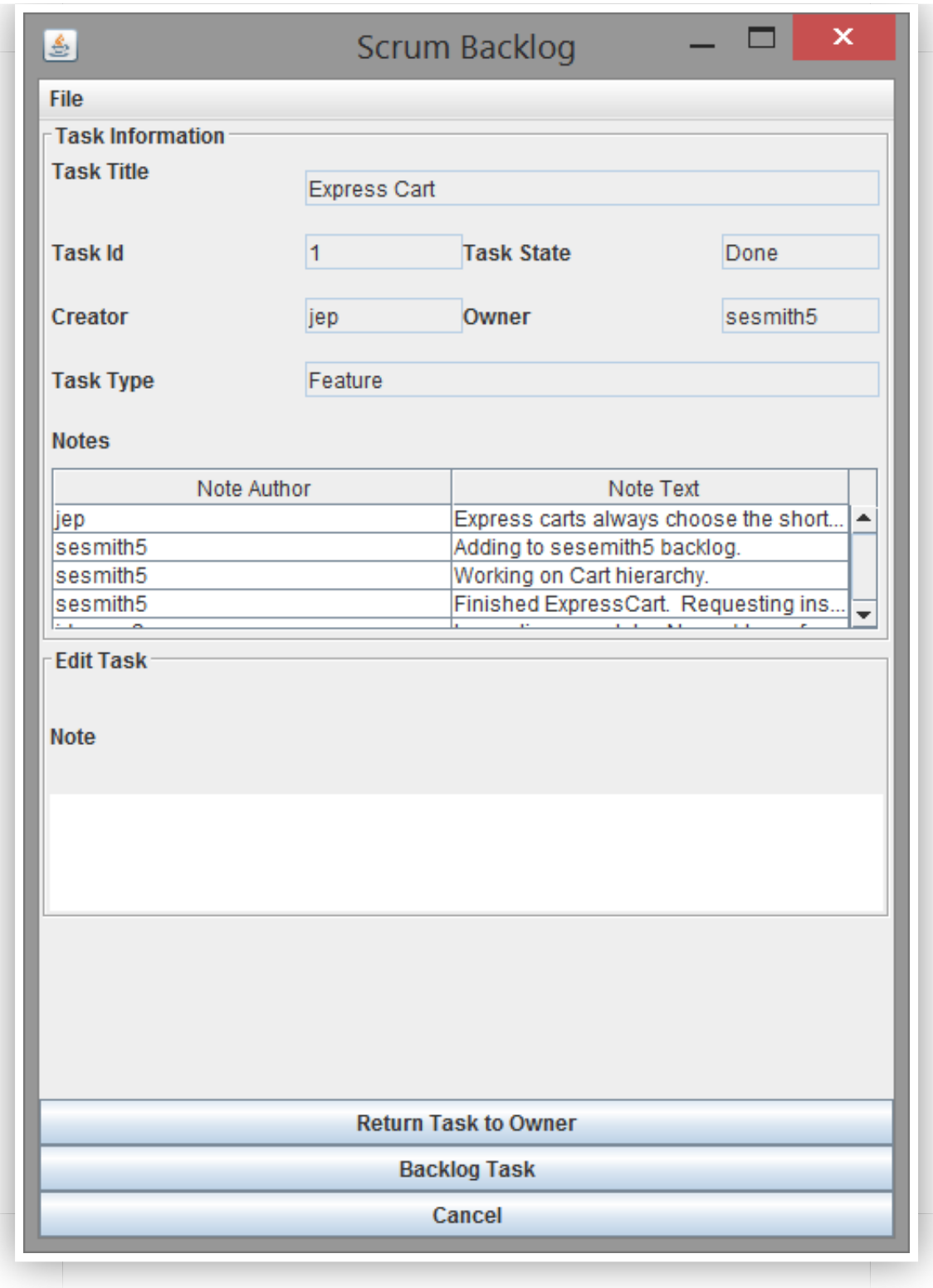

Figure 9: ScrumBacklog GUI showing the user interface for interacting with a task in the Done state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: A user can return the task to the owner for additional work [S1], return the task to the backlog for additional work by another developer [S2], or cancel the edit action [S3]. The user must provide a note, which is authored by the owner.

Subflows:

[S1] The user provides details in a note and clicks **Return Task to Owner** [E1]. The task moves to the Processing state. The note, with the owner as author, is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S2] The user provides details in a note and clicks **Backlog Task** [E1]. The task's state is updated to Backlog. The note, with the owner as author, is saved with the task. The owner is then removed. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S3] The user clicks **Cancel.** The user is returned to the task list [UC2] and the task listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Done state user interface to enter the not text.

Use Case 8: Rejected State

Preconditions: A user has selected a task to edit that is in the Rejected state [UC2, S3]. The user interface for editing a task in the Rejected state is shown in Figure 10.

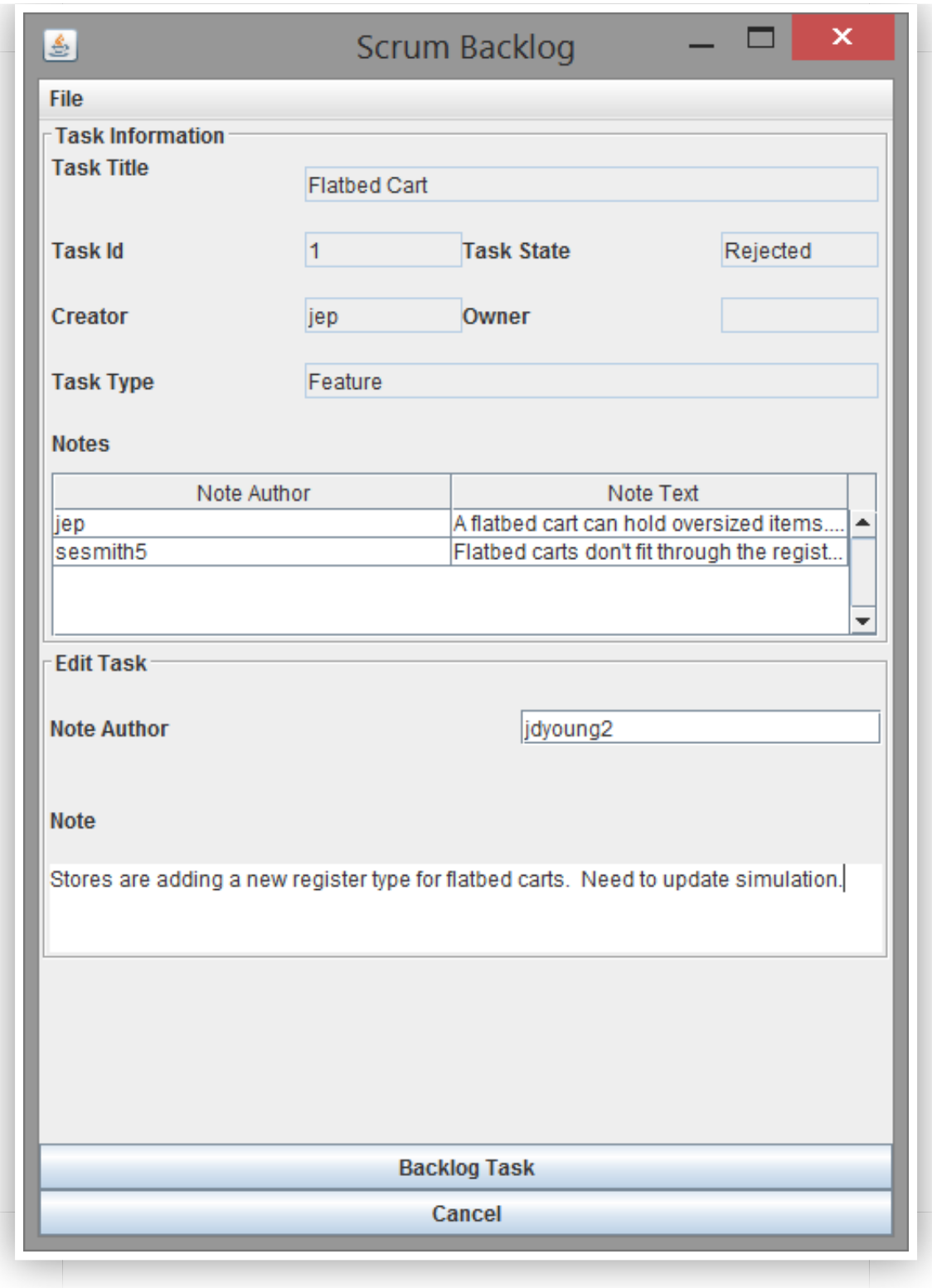

Figure 10: ScrumBacklog GUI showing the user interface for interacting with a task in the Rejected state. The top part of the GUI shows the task information, the lower part of the GUI shows how the user can interact with the task.

Main Flow: A user can return the task to the backlog for additional work by another developer [S1] or cancel the edit action [S2]. The user must provide a note. A note includes the note's author and text.

Subflows:

[S1] The user provides details in a note and clicks **Backlog Task** [E1][E2]. The task's state is updated to Backlog. The note is saved with the task. The user is returned to the task list [UC2] and the task listing reflects the updated state.
[S2] The user clicks **Cancel.** The user is returned to the task list [UC2] and the task listing is not changed. The note is not saved.

Alternative Flows:

[E1] If the field for the note's author is left empty, a dialog opens with the message "Invalid note author id." The user clicks OK and is returned to the Done state user interface to enter the note author id.
[E2] If the field for the note's text is left empty, a dialog opens with the message "Invalid note text." The user clicks OK and is returned to the Done state user interface to enter the not text.

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 teaching staff design (Project 1 Part 2), but first you need to propose your own. To do this, we give you a scenario and insist on some restrictions for your design.

Scenario

Now that you have the requirements you can propose a design to create an application that will help the Senior Design Center, and everyone, keep track of project tasks! Your manager has requested that you and your co-workers each propose a design for the static layout of the system by identifying the objects, their state and behavior, and the relationships between the objects in the system. Your design must include a State Pattern that models the finite-state machine of an incident's state. Your manager will then choose a design that best describes the requirements for implementation.

Assignment

You must design an application that satisfies the requirements of the Scrum Backlog application. You will create a design proposal that shows the objects, their states and behaviors, and the relationships among the objects needed to implement 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),

utilize the State Pattern (see the note about State Pattern, below),
contain at least one interface or abstract class,
contain at least one inheritance relationship (which may include implementing an interface),
contain at least one composition relationship,

To help your evaluate your design, you should 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 these relationships are important?
5. Have you identified any design patterns appropriate for implementing the system (i.e., the State Pattern)? What are they and why are they appropriate?
6. 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. Your design must 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 an incident 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 incident and delegates the transitions to the current state for the given incident. For more information on

the State Pattern, see the Wikipedia article and the example Horner's Rule State Pattern implementation.

Design Proposal and Rationale Submission Format

Submit your design proposal via Gradescope 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 more direction?

See the following example design proposal: Sample Design Proposal.

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. We will start you off with a scenario.

Scenario

Your manager has requested a black box test plan that describes **five test cases** that will determine if the finished program is ready to ship to the customer. Your manager wants you to write the tests before you begin development 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 Scrum Backlog 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? What are the command line arguments, if any? What do the input files contain?). Describe the instructions at a level where anyone using Eclipse could run your tests.

Task XML Test Files

Format

You must submit your black box test plan via your section's submission platform as a PDF. Use the provided template as a starting point for your black box test plan. Save your BBTP as a PDF. Submit the PDF to Gradescope under the assignment named **P1P1: BBTP**.

Need a little more direction?

See the following example black box test plan: Sample Black Box Test Plan.

Deployment

For this class, deployment means submitting your work for grading. For Part 1 of this programming assignment, you must submit two PDF documents:

1. Design document with incorporated UML class diagram.
2. Black box test plan document.

Make sure that your submissions satisfy the grading rubrics.

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

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 Gradescope failures). You are able to make multiple submissions of the same file. (Later submissions to a Gradescope assignment in 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 1: Scrum Backlog ▶