Faculty of Computers and Artificial Intelligence - Cairo University (credit hours system)

Assignment 2

Software Reuse

Supervised by:

Dr. Desoky Abd El-qawy

Done by:

Mark Rofaeel

Software Construction

Academic Year 2024 – 2025

Part 1

1. Description of the proposed feature's algorithm:

To put a hexagon figure into `Magelan`, I made a subclass of the existing `Figure` class called `HexagonFigure`. The hexagon is drawn by overriding the method `draw(Graphics g)` thanks to the logic of a six-point polygon. I then shaped the menu system through the implementation of a `MoveHexagonAction` and `ChangeHexagonColorAction` class, each with a base `MoveFigureAction` and `ChangeFigureColorAction` for reuse. Menu items were tied dynamically to these actions so subsequent numbers can be added in the same was without having to modify core logic.

2. Relevant code modifications:

Classes understood/modified:

1. figures.Figure (base class for figures)
2. figures.HexagonFigure (new class created)
3. ui.Editor (to integrate new figure and menu items)
4. actions.MoveFigureAction (created base class for move actions)
5. actions.ChangeFigureColorAction (created base class for color change actions)
6. menus.FigureMenu (modified to add new menu entries)

New classes created:

1. figures.HexagonFigure
2. actions.MoveHexagonAction
3. actions.ChangeHexagonColorAction

3. Write-up:

Reuse task took more than 4 hours with Eclipse IDE 2024-03 (4.32).

Early problems with setup included Eclipse project import failure and missing library dependencies requiring manual .classpath adjustments. Subsequently, the most significant issue was poor internal documentation in the Magelan source. Figure construction was deeply nested in non-extensible root classes, especially Figure and Editor, which violated modularity principles.

Picking up the drawing logic was relatively straightforward. Glancing at Figure, I understood that new figures must override draw(Graphics g) and manage their coordinates themselves. Drawing a HexagonFigure extending java.awt.Polygon was straightforward after checking coordinate math by hand. Incorporating movement and color change involved even more abstraction: I created generic base classes MoveFigureAction and ChangeFigureColorAction to enable extensibility beyond the hexagon.

The most excruciating task was the menu system change. Magelan's UI code was extremely brittle: menu actions were hardcoded per-figure in Editor, which caused duplication for each new figure action unless significant refactoring was undertaken (not permissible under time pressure). Eclipse's outline view and quick refactor assisted partially, but a lack of tests and uncertain separation of model/view/controller concepts impeded quick validation.

Opinion on Pre-Planned Reuse Approach

Advantages:

Pre-planned reuse shows strong effectiveness when frameworks define clear extension points and have low internal coupling. Reuse frameworks that are well-designed allow for the rapid addition of features without requiring changes to core code. In a well-designed framework, my hexagon and actions might have been added by configuration rather than through manual changes to the Editor.

Challenges:

In Magelan, hard-coded tight coupling severely undermined reusability. New figures entailed manual changes to Editor and explicit wiring to menus, as opposed to modularity goals. Example: Editor explicitly binds every figure type instead of using dynamic discovery (violating Open-Closed Principle). In addition, absence of a command pattern for figure operations led to duplicated boilerplates. Therefore, despite experience, recycling Magelan without a large technical debt was cumbersome and risky.

Prospective Preference

I would apply pre-planned reuse only if:

- The framework possesses well-documented, clean APIs.
- Extension follows Open-Closed Principle strictly.
- Plugins and dependency-injection allow other functionalities to be bound late.

Otherwise, clean implementation rather than reuse to prevent hardcoding fragile designs would be my choice.

Concrete Examples

Menu integration: Had to change `menus/FigureMenu` and `ui.Editor` directly.

Action extensibility: Manual subclassing of move and color change per figure because of no generic action dispatching.

Recommended Framework Enhancements

Implement plugin architecture: Register new figures/actions dynamically at runtime. Use dependency injection frameworks (e.g., Guice) for loose coupling. Implement a Command Pattern for operations on figures to avoid action duplication per figure. Provide developer documentation with extension point documentation.

GitHub Link

https://github.com/mark-rofaeel/Software-Reuse

Part 2

1. *Reinhartz-Berger, I. (2024). Challenges in software model reuse: cross application domain vs. cross modeling paradigm. Empirical Software Engineering, 29(1), 16.*

In Reinhartz-Berger's work, challenges in reusing software models across different application domains and modeling paradigms are investigated. In the study, controlled experiments between 64 participants were conducted in comparing model reuse correctness across object-oriented (UML) and data-oriented paradigms (ER and DFD).

Strengths:

- Experimental design used in this work provides concrete evidence about cross-paradigm and cross-domain model reuse issues.
- It distinguishes between reuse types and points out that cross-domain reuse is more effective than cross-paradigm reuse, particularly in the case of addition operations.

Limitations:

- The study only looks at specific modeling paradigms (UML, ER, DFD), which may limit the extensibility of findings to other paradigms or modeling styles.
- The research does not investigate how tool support or automation may mitigate the barriers to model reuse identified.

2. *Davis, J. C., Jajal, P., Jiang, W., Schorlemmer, T. R., Synovic, N., & Thiruvathukal, G. K. (2024). Reusing Deep Learning Models: Challenges and Directions in Software Engineering. arXiv preprint arXiv:2404.16688.*

Software engineering challenges of reusing deep neural networks (DNNs) are being addressed here with challenges like a lack of technical skills and engineering practices.

Overcoming Reinhartz-Berger's Limitations:

- Better tools for DNN reuse are highlighted by research, indicating the gap Reinhartz-Berger had identified in terms of tool support.
- Since the focus is placed on DNNs, the article generalizes model reuse discourse from traditional paradigms to provide insight into issues in modern times.

Limitations:

- The focus on DNNs may limit the applicability of findings to other paradigms of modeling.

---

3. *Taraghi, M., Dorcelus, G., Foundjem, A., Tambon, F., & Khomh, F. (2024). Deep Learning Model Reuse in the HuggingFace Community: Challenges, Benefit and Trends. arXiv preprint arXiv:2401.13177.*

The study explores pre-trained model reuse within the HuggingFace community, with challenges ranging from the lack of novices' guidance to model comprehension issues.

Breaking Away from Reinhartz-Berger's Shortcomings:

- The paper sheds light on real-world practices and challenges in model reuse, complementing Reinhartz-Berger's experimental findings.
- It stresses the importance of proper documentation and user guidance, factors not properly addressed in Reinhartz-Berger's research.

Limitations:

- Platform Specificity: The focus on the HuggingFace platform limits the external validity of the outcomes to other contexts.

Recommended Alternative Approach:

Integrated Reuse Framework:

In order to meet the limitations given in Reinhartz-Berger's work and that of others, we recommend an integrated reuse framework with the following features:

1. Meta-Modeling Support: Create a meta-modeling layer abstracting different modeling paradigms for cross-paradigm reuse.
2. Context-Aware Tooling: Offer tools inspecting the context of the source and target domains to determine suitability and suggest necessary adjustments.
3. Component Encapsulation: Encapsulate reusable components with well-defined interfaces and documentation to facilitate integration in new domains.
4. Empirical Validation: Perform intensive empirical tests to confirm the efficiency of the framework in a multitude of domains and paradigms.

Benefits:

- Improved Reusability: Abstraction of modeling paradigms and context-aware tools make reuse of models more likely.
- Low Adaptation Cost: Well-documented interfaces and documentation reduce adaptation effort.

Potential Challenges:

- Development Overhead: Building and maintaining the meta-modeling layer and tools can be labor-intensive.
- Learning Curve: Users may need to be trained in order to effectively use the framework and its tools.