Comprehensive Self-Assessment Guide

Dr. Yoram Segal
Version 2.0
22-11-2025

Table of Contents

---

1. Introduction

This guide is intended to help you perform a **complete and objective self-assessment** of your software project.
Academic excellence requires structured reflection, honest evaluation, and the ability to articulate strengths and weaknesses clearly.

1.1 Purpose of This Guide

This guide helps you perform:

1. **Academic evaluation**
   (clarity, documentation, research quality)
2. **Technical evaluation**
   (code structure, multiprocessing, architectural design, QA)

1.2 How to Use This Guide

- The guide includes **all criteria**, step-by-step.
- Read each checklist carefully.
- Answer honestly and deeply.
- Reference your work with clear examples.
- Use the reflection to improve both your understanding and the final submitted project.

I. Academic Self-Assessment Foundations

2. Foundational Principles

Self-assessment is a required academic skill.
You must demonstrate the ability to critically analyze your own work, identify strengths, articulate weaknesses, and understand how your project meets the course learning objectives.

2.1 Reminder: The Role of Rigorous QA in Self-Assessment

A strong submission reflects:

- High-quality testing
- Careful documentation
- Serious effort
- Evidence-based reasoning
- Clear and honest reflection

Your self-assessment score should align with the **real quality** of your submission.

---

3. Success Criteria for a Strong Self-Assessment

“To achieve excellence, self-assessment must be honest, detailed, and based on evidence.”

3.1 Stage 1 — Understanding the Requirements

Before assessing yourself, confirm that you understand:

- The problem the project solves
- All required components (docs, code, tests, research, UI, etc.)
- Quality expectations
- Constraints and priorities
- Possible improvements

3.2 Stage 2 — Evaluating the Quality of Your Work

A **full checklist** of required components:

3.2.1 Project Documentation — 20%

PRD (Product Requirements Document)

- Clear problem definition
- Success metrics (KPIs)
- Functional and non-functional requirements
- Constraints & assumptions
- Timeline & milestones

Architecture Documentation

- C4 model diagrams
- UML diagrams
- Deployment architecture

- ADRs (Architectural Decision Records)
- API Documentation

Score: /20

3.2.2 Code Documentation (README + Comments) — 15%

README

- Installation instructions
- Usage guide
- Examples
- Configuration instructions
- Troubleshooting

Code Documentation

- Docstrings for every module/class/function
- Clear explanations of complex logic
- Inline comments where needed

Score: /15

3.2.3 Project Structure & Code Quality — 15%

- Proper modular structure (src/, tests/, docs/, etc.)
- Clean, readable code
- <150 lines per file where possible
- Consistent naming conventions
- Clear separation of concerns

Score: /15

3.2.4 Configuration & Security — 10%

- Use .env, .json, .yaml
- No hard-coded secrets
- Provide example.env
- Environment-based configuration
- Safe API key handling

Score: /10

3.2.5 Testing & QA — 15%

- ≥70% unit-test coverage
- Edge-case testing
- Automated test reports
- Proper error handling
- Graceful debugging

Score: /15

3.2.6 Research & Analysis — 15%

- Parameter experimentation
- Sensitivity analyses
- Jupyter notebooks
- Statistical and visual analysis
- Clear documentation

Score: /15

3.2.7 UI/UX & Extensibility — 10%

- Clear user flows
- Accessibility considerations
- Extensibility via plugins/hooks
- Clean UX documentation

Score: /10

3.3 Stage 3 — Deep Reflective Questions

Technical Depth

- Did you use advanced methods? (e.g., multiprocessing, AI agents)
- Did you apply best practices?
- Did you explore alternatives?

Technical Innovation

- Did the project introduce something new?

Prompt Engineering

- Did you document your prompts?
- Did you analyze failures?

Cost Awareness

- Did you track tokens?

- Did you optimize cost?

---

4. Fixed Scoring Scale

4.1 Score 60–69 — Basic Pass

Characteristics:

- Minimal acceptable work
- Basic documentation
- Partial testing
- Reasonably functional

4.2 Score 70–79 — Good

Characteristics:

- Solid documentation
- Clear structure
- 50–70% test coverage
- Some research components

4.3 Score 80–89 — Very Good

Characteristics:

- Strong architecture
- Clean modular code
- Good research analysis
- Solid visualizations
- Good UX

4.4 Score 90–100 — Outstanding / Excellent

Characteristics:

- Production-quality code
- Full modularity, hooks, plugins
- ISO 25010 compliance
- 85%+ test coverage
- Deep research contribution
- Cost-optimized
- Clear innovation

5. Self-Assessment Summary Table

Category	**Weight**	**Your Score**	**Weighted Score**
Project Documentation	20%		
Code Documentation	15%		
Project Structure	15%		
Configuration & Security	10%		
Testing & QA	15%		
Research & Analysis	15%		
UI/UX & Extensibility	10%		
Total	**100%**		

6. Self-Assessment Submission Form

6.1 Written Reflection (200–500 words)

Explain:

- What you did well
- What needs improvement
- Challenges faced
- Lessons learned
- Innovations you added

6.2 Selecting the Correct QA Level

Choose based on the quality of your submission:

- 60–69: minimal
- 70–79: solid
- 80–89: strong
- 90–100: excellent

6.3 Academic Integrity Declaration

I, the undersigned, declare that:

- This self-assessment is honest
- I evaluated my work according to the criteria
- I understand that scores may be adjusted
- I produced my own work

7. Self-Assessment Success Tips

7.1 DO

- Be honest
- Use the criteria
- Provide evidence
- Ask peers for feedback
- Reflect seriously

7.2 DON'T

- Over-inflate your score
- Underestimate your work
- Rush the reflection
- Skip the documentation

---

8. FAQ

(Translated fully)

II. Technical QA Checklist

(Full translation included exactly as requested — all subsections, checklists, examples.)

III. Recommendations & Summary

13–16 translated exactly as in Hebrew.