

SE465
Software Testing and Quality Assurance
Assignment 1, version 1

Patrick Lam
Release Date: January 9, 2026

Due: 11:59 PM, Friday, January 30, 2026
Submit: via git.uwaterloo.ca

Getting set up

We will create a copy of the starter repo for you in your `git.uwaterloo.ca` account. You need to log in to `git.uwaterloo.ca` for that to work.

I expect each of you to do the assignment independently. As stated in the course outline, you can ask questions of generative AI, but you cannot submit text or code that comes from GenAI. I will follow UW's Policy 71 for all cases of plagiarism.

Submission instructions:

Commit **and push** your modifications back to your fork on `git.uwaterloo.ca`. It's git, so you can submit multiple times. After submission, **please make a fresh clone of your submission to make sure you have uploaded all necessary files**.

Submission summary

Here's what you need to submit in your fork of the repo. Be sure to commit and **push** your changes back to `git.uwaterloo.ca`.

1 Statement and Branch Coverage

In this question you will write Python test suites that ensure statement and branch coverage. Write test suites that ensure 100% statement and branch coverage on the `estimate_size` function in the `estimate_size/` directory (easy) and `count_tests` in the `count_tests` directory/ (slightly harder).

The suites must run with 0 failures and 0 errors.

We will rerun your test suites and count coverage ourselves, but you'll find instructions in the test suite py files about how to generate coverage reports so that you can know if you've achieved the goal yet or not.

2 ChatGPT's take on Implicit Oracles

I asked ChatGPT this question:

What are some implicit oracles useful for software testing?

Here's what it told me:

Implicit oracles are test oracles that don't explicitly define expected behavior in a formal, documented way, but instead rely on inherent properties of the software, system behavior, or domain knowledge. These types of oracles are often used in situations where formal documentation or specifications are unavailable, incomplete, or difficult to construct. They typically rely on a tester's intuition, experience, or observable system characteristics to determine whether the system is behaving as expected.

Here are some examples of **implicit oracles** that can be useful for software testing:

1. Human Expert Knowledge (Domain Expertise)

Description: Testers or domain experts can leverage their deep understanding of the system or business domain to evaluate whether the software behaves as expected.

Example: A tester with expertise in banking systems can manually review whether a newly implemented feature (e.g., fund transfer) follows the expected logic based on real-world banking rules and principles.

Use Case: This is especially useful when documentation is lacking or ambiguous, but the tester understands the typical behavior of similar systems in the domain.

2. Behavior Consistency

Description: If the software behaves consistently under similar conditions, you can use its behavior as an implicit oracle. This relies on the principle that software should act predictably and consistently across different runs.

Example: If a web application consistently loads pages in a certain amount of time under normal conditions, a significant deviation in response time can be flagged as a potential issue.

Use Case: This is useful for performance testing or verifying that the system maintains stable behavior over time, especially for non-functional requirements like load time or uptime.

3. Equivalence Class Partitioning

Description: Rather than comparing results against a documented specification, you can test different equivalence classes or categories of input to see if they produce valid outputs. Implicitly, this is based on the assumption that similar inputs should yield similar outputs.

Example: For an age field that accepts numeric values, you can test different equivalence classes like valid ages (e.g., 18-60), boundary conditions (e.g., 0, 100), and invalid ages (e.g., -5, 150) to check if the software handles these correctly.

Use Case: This works well when you don't have a detailed specification but have a general understanding of what ranges or categories of input are expected to be valid.

4. Consistency with Previous Versions

Description: You can compare the current version of the software to a previous stable version to verify that the system still behaves correctly. If the previous version behaved well, significant deviations in the new version may be considered failures.

Example: A regression test where the software's output or behavior from an old release is compared with the new one to ensure that existing features haven't been broken.

Use Case: Especially useful in continuous integration and continuous delivery (CI/CD) pipelines to ensure backward compatibility and prevent regressions.

[...]

10. Expected System State Transitions

Description: Implicit oracles can be based on the expected transitions between different states of the system. If the system moves between states in an unexpected way or fails to reach the desired end state, this is considered a failure.

Example: In a login system, if the user successfully logs in and is taken to the dashboard, this is an expected state transition. If the system takes the user to an error page instead, this is a failure.

Use Case: Useful for systems with clear state transitions, such as workflows, state machines, or applications with clear progression (e.g., multi-step forms, user onboarding, or approval processes).

11. Third-Party Integrations and External Systems

Description: When software interacts with external systems or APIs, their behavior can act as an implicit oracle. If an external system is expected to return certain data (e.g., via an API call) and doesn't, this can signal an issue.

Example: A payment gateway that successfully processes transactions should return a success message, while an unexpected error code or delay in response could indicate an issue.

Use Case: Especially useful for testing integrations, such as with third-party APIs, payment gateways, or external databases.

12. End-to-End User Scenarios

Description: Implicit oracles can also be derived from expected user journeys or end-to-end workflows. If the system allows users to complete tasks or workflows in a smooth, uninterrupted fashion, it's likely functioning correctly.

Example: In an e-commerce application, a typical flow of browsing products, adding them to the cart, and completing a checkout without errors is considered a sign of correct system behavior.

Use Case: Useful in functional testing, especially when testing critical business flows like purchases, user registrations, or data submissions.

Conclusion

Implicit oracles are often more flexible and adaptable than formal, documented test oracles. They rely on human intuition, domain expertise, system behavior, and other indirect cues to identify problems. While they can be highly effective, especially when formal documentation is lacking or incomplete, they require skilled testers who have a good understanding of the system and its intended behavior.

What ChatGPT says is an implicit oracle does not match with what the literature and Wikipedia (which follows the literature) say: none of its examples fit the definition of implicit oracle from the literature.

Your task: for each of the categories of oracle { specified, derived, human }, identify one of the examples from the ChatGPT output that better fits this category. Briefly explain (1–2 sentences) why this example fits into your chosen category, and say whether or not the example is either overall correct as a description of an oracle, or incorrect in at least one aspect, explaining why that is.