Software Engineering Principles

Design for Testability

Implemented on (date)	20/05/2024
Approved by (name & role)	Head of Software Practices (Fahad Anwar) In consultation with TAG (Technical Advisory Group).
Last review on (date)	16/05/2024
Reviewed by	TAG (Technical Advisory Group)
Next review due on (date)	16/05/2025
Principle owner (name)	Head of Software Practices In consultation with Technical Advisory Group (TAG) representing Software Engineers, Cloud Division, TISS and Security Division.
Principle owner (division)	Digital Services and Technology (DST)
Main point of contact (name)	Fahad Anwar Software Engineering Head of Practice
Status	Approved

Principle Review Record

This Review Record is to be completed on each time a review is conducted. Its purpose is to maintain a record of reviews, recording who conducted the review, the date of the review and the outcome of the review (fit for purpose, amendment required, principle no longer required, etc).

This principle is to be reviewed annually.

Review No	Review Conducted By	Review Date	Review Outcome
01			

Amendment Details

Date	Amendment Summary	Amended by	Version

RASCI (For detail please – RASCI Information document)

Responsible	G6 Program Managers through Technical Leads, G7 and SEO's
Accountable	Head of Software Practices
Supportive	Head of Cloud Functions (Amazon, GCP, Azure) SAIM SIRA Software Engineering Community of Practice (SE-CoP)

Consulted	Technical Advisory Group (TAG) representing Software Engineers, Cloud Division, TISS and Security Division.
Informed	Senior Leadership Team Software Engineering Community SAIM SIRA Design Authority Chair

Design for Testability

Solutions should be designed - and code structured - in a way that makes execution of its tests happen more easily and quickly.

Rationale

Solutions that are designed with testing concerns in mind facilitate faster feedback and are ultimately able to release more frequently and safely. Designing for testability naturally leads to improved understandability and evolvability.

Implications

- The internal state of a component or system should be understandable through deliberately designed external interfaces or outputs, without invasive techniques such as attaching a debugger or filling the code with debug-level logging statements.
- Structure your code to allow components to be executed in isolation in order to observe its behaviour during checking and testing.
- Ensure that the components of a system are separated into well-defined responsibilities.
- It should be easy to understand the components of a system - the code should be written in a way that is self-explaining and documented through its tests or, if necessary, through separate documentation.
- Software should be designed and structured to support automation of tests wherever possible. These tests should run quickly and throughout the development cycle to support Continuous Delivery. Use exploratory testing as a supporting testing style for non-deterministic testing.
- Carefully consider whether a diversity of technologies in use by a system introduces challenges in the testing methods and tools required and adapt the approach accordingly.
- Release frequently through to Production. This helps build confidence in testing and reduces risk in the release process, and any issues identified soon after the cause are easier to fix whilst still fresh in the engineer's mind.

Questions to be considered.

- Does the solution have areas of high risk or complexity? These require more understanding and testing. Highlight integration interfaces, dependencies on third party interfaces and complex algorithms as potential areas to focus testing effort.
- Can the solution architecture be decoupled so individual components can be tested in isolation?

- Consider requirements for logging, monitoring and debugging? Having clearly defined logs and mechanisms to debug the code helps testing when issues are found?
- Understand the areas of testing required and prioritise which are the highest priority to verify the core functionality. Low priority areas of test that take a long time to automate should be lower down the list of priorities for resourcing.
- Consider both functional and non-functional requirements of the solution. Functional requirements are crucial to design effective tests to verify the solution performs as designed. Non-functional requirements will highlight performance, security and scalability of the solution which will have an impact on how the solution is tested.

Motivation for the principles

- https://www.gov.uk/guidance/the-technology-code-of-practice
- https://engineering-principles.jlp.engineering/
- https://github.com/otto-de/tech_manifest
- https://www.gov.uk/government/publications/security-policy-framework/hmg-security-policy-framework