Software Engineering Principles

Design for Emergent Reuse

Implemented on (date)	
Approved by (name & role)	Head of Software Practices (Fahad Anwar) In consultation with TAG (Technical Advisory Group).
Last review on (date)	13/03/2024
Reviewed by	TAG (Technical Advisory Group)
Next review due on (date)	
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	Final Draft
Published version link	

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 Emergent Reuse

Design for well-defined use cases and adaptability. Address reuse as an optimisation opportunity rather than a goal.

Rationale

This principle is a version of YAGNI applied specifically to reusability rather than functional features. We believe it deserves extra attention in this context as it can conflict with our natural instincts about what good looks like.
Designing for reuse from the outset, in the absence of well-understood use cases, requires us to make assumptions about what will be valuable at some point in the future. These assumptions usually turn out to be wrong. As well as wasting effort and creating a cost of delay, we have seen this approach to reuse significantly inhibit change, by creating unnecessary complexity, dependencies and bottlenecks.
Code that is designed for reuse is generally harder to build, maintain and use than code designed for a single purpose, so we should first establish that there is concrete value in reuse outweighing the costs, and accept duplication until then.
However, where there is a clear justification of building a reusable code then we should follow avoid duplicating effort and unnecessary costs by collaborating across government and sharing and reusing technology, data, and services.

Implications

- Identify reuse as it emerges through evolution or modelling the business domain; prefer duplication over premature abstraction until you have evidence new dependencies will not constrain the required pace of change.
- Be sceptical of components that look similar but aren't the same once you consider business use cases and what might trigger them to change and diverge.
- This principle can only be effective if reuse can be enabled in the future at a similar cost to today. We must therefore focus on building easy-to-change Evolutionary Systems.
- If reused code becomes a bottleneck, remove the coupling by forking or duplicating the code.
- The risks of reuse are usually lower for generic, domain-independent features such as monitoring, logging, service discovery, configuration, etc. The opportunities and risks of patterns such as Service Templates and Service Chassis should be evaluated in this context.

Questions to be considered.

- Are you implementing actual or anticipated requirements?

- Did you design using a BDUF (Big Design Up Front) approach or MVP? as using the BDUF should provide more context if usability is required, whereas, in MVP model the requirements will evolve

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
- https://hbr.org/2023/06/how-to-find-the-time-to-connect-with-colleagues-when-youre-very-very-busy

Principle Enforcement

G6 Program Managers through Technical Leads or Sr. member of respective software development team is responsible for monitoring and enforcing principle compliance.

Exception process

It is acknowledged that situations arise in which the Principal Detail as above may not be able to be met. Where this is the case, a documented Principal exception must be sought from the Principal Owner (specified in the table on the title page).