# Methodology Statement – Synputer Development Project

The Synputer project requires an integrated hardware–software life-cycle approach to deliver a commercially viable UK-built personal computer within a strict thirteen-month window and a £500 000 budget. The dual dependency on physical components and rapidly evolving firmware makes a purely linear Waterfall model too rigid, while a pure Agile model would lack the governance demanded by fixed hardware lead-times.

To balance these constraints, the team adopted a **Hybrid Agile Systems Development Life Cycle (SDLC)** that combines **Scrum** iterations with formal **stage-gates** for design and integration.

## Structure of the Hybrid Model

1. **Requirements & Feasibility Gate** – Stakeholder needs are derived from the case-study transcript and validated against Appendix 1 (Hardware BOM) and Appendix 2 (Software BOM).

2. **Design & Architecture Gate** – System and software architectures are peer-reviewed before component procurement.

3. **Incremental Development Sprints** – Two-week Scrum sprints deliver firmware and UI modules, each ending with a sprint review and a retrospective.

4. **Integration & Test Gate** – Hardware prototypes and software builds are combined and verified using Behaviour-Driven Development (BDD) tests.

5. **User Acceptance & Handover Gate** – Final validation, documentation, and release to production.

## Incremental Development Sprints

Each two-week sprint ends with two events:

- **Sprint Review (60 min)** – A stakeholder demonstration of the working increment (hardware prototype, software build, or emulator integration).

- **Sprint Retrospective (90 min)** – An internal process evaluation to identify what worked, what failed, and what should change.

### Justification for Bi-Weekly Retrospectives

Fixed-budget (£500 000) and fixed-deadline (December 1983) projects leave no tolerance for waste. **Durham and Michel (2021)** emphasise that Lean feedback loops accelerate learning and eliminate inefficiency; bi-weekly retros detect process errors, knowledge gaps, and coordination failures before they cascade. **Agrawal et al. (2024, pp. 15–17)** show that approximately 31 per cent of design-related errors originate from knowledge gaps; structured retros apply their taxonomy by asking whether an issue is cognitive, organisational, or process-based, and implementing corrective action within the next sprint.

Retrospectives are also vital for **hardware–software integration**: when pin-mapping or ULA–CPU design assumptions fail, a retro allows same-sprint mitigation instead of costly redesigns months later. For virtual collaboration, retros provide synchronisation points that surface miscommunication across asynchronous channels (WhatsApp, email).
Finally, each retro includes a budget check – current burn-rate versus remaining £500 k – ensuring early detection of variance and supporting transparent reporting to EDC.

## Justification of the Hybrid Agile Model

- **Lean Systems Thinking:** Durham and Michel (2021) stress iterative feedback loops to minimise waste and make progress visible. Embedding retros within each gate ensures measurable quality.

- **Knowledge-Based Error Prevention:** Agrawal et al. (2024) identify knowledge gaps as a major design-error source; continuous retros mitigate these through shared learning.

- **Schedule Realism:** Brooks (1975) warns that adding people to late projects only delays them further; our plan uses stable, cross-functional sprint teams and reserves buffer capacity for testing, not expansion.

- **Governance and Traceability:** *ISO/IEC/IEEE 16326 (2019)* requires defined gates for scope, risk, and quality artefacts – integrated here for auditability.

- **Stakeholder Engagement:** Scrum reviews allow EDC to inspect prototypes early, aligning evolving expectations without uncontrolled scope creep.

## Continuous Improvement *Example*

In Sprint 2, a retrospective revealed that the **HyperBasic** compiler failed on 512 KB systems because a developer assumed 1 MB memory. The root cause, classified as a *knowledge-based error* (Agrawal et al., 2024), prompted a Definition-of-Done update: every code change must pass 512 KB, 768 KB, and 1 MB tests. The fix consumed four engineering hours but prevented a week-long redesign, demonstrating tangible ROI from the retrospective process.

## Summary

This Hybrid Agile SDLC provides flexibility for software innovation and the rigour required for hardware manufacturing. Iterative sprints shorten feedback cycles, while gate reviews and retrospectives maintain accountability, knowledge capture, and cost control. The approach adheres to Lean and ISO/IEEE principles and directly supports the Synputer project's objective: a compliant, high-quality system delivered within fixed financial and temporal constraints.

---

## References

Agrawal, T., Walia, G.S. and Anu, V.K. (2024) 'Development of a software design error taxonomy: a systematic literature review', *SN Computer Science*, 5(467). doi: 10.1007/s42979-024-02797-2.

Brooks, F.P. (1975) *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley. Available via Open Library.

Durham, D. and Michel, C. (2021) *Lean Software Systems Engineering for Developers: Managing Requirements, Complexity, Teams, and Change Like a Champ.* 1st edn. Berkeley, CA: Apress. ISBN 978-1-4842-6932-9. Available at: SpringerLink.

ISO/IEC/IEEE (2019) *16326: Systems and Software Engineering — Life-Cycle Processes — Project Management.* Geneva: International Organization for Standardization. Available at: ISO Catalogue.