## Design Review Checklist

1. Are the following attributes well-defined for each design entity?
    a. **Identification** (unique name)
    b. **Type** (describing what kind of design entity it is)
    c. **Purpose** (describing why it was introduced, in terms of the requirements)
    d. **Function** (summarizing what the component does)
    e. **Dependencies** (possibly `none'; describing the *requires* or *uses* relationship)
    f. **Interface** (*provided* by the design entity)
    g. **Processing** (including autonomous activities)
    h. **Data** (information `hidden' inside)
2. Is the relationship to the **requirements** clearly motivated? Is it clear why the proposed architecture realizes the requirements?
3. Is the software architecture as **simple** as possible (but no simpler)?
    - No more than 7 loosely-coupled coherent high-level components.
    - Lower-level components possibly clustered into high-level components (hierarchy).
    - Using standard(ized) components.
    - Is deviation from intuitively obvious solution motivated?
4. Is the architecture **complete**?
    - Are all requirements covered?
    - Trace some critical requirements through the architecture (e.g. via use cases).
5. Are the component descriptions sufficiently **precise**?
    - Do they allow independent construction?
    - Are **interfaces** and **external functionality** of the high-level components described in sufficient detail?
    - Interface details:
        - Routine kind, name, parameters and their types, return type, pre- and post-condition, usage protocol with respect to other routines.
        - File name, format, permissions.
        - Socket number and protocol.
        - Shared variables, synchronization primitives (locks).
    - Have features of the target programming language been used where appropriate?
    - Have implementation details been avoided? (*No* details of internal classes.)
6. Are the **relationships** between the components explicitly documented?
    - Preferably use a diagram
7. Is the proposed solution **achievable**?
    - Can the components be implemented or bought, and then integrated together.
    - Possibly introduce a second layer of decomposition to get a better grip on achievability.
8. Are all *relevant* **architectural views** documented?

    - **Logical** (Structural) view (class diagram per component expresses functionality).
    - **Process** view (how control threads are set up, interact, evolve, and die).
    - **Physical** view (deployment diagram relates components to equipment).
    - **Development** view (how code is organized in files).
9. Are **cross-cutting issues** clearly and generally resolved?
    - Exception handling.
    - Initialization and reset.
    - Memory management.
    - Security.
    - Internationalization.
    - Built-in help.
    - Built-in test facilities.
10. Is all **formalized material** and **diagrammatic material** accompanied by sufficient explanatory text in natural language?
11. Are **design decisions** documented explicitly and motivated?
    - Restrictions on developer freedom with respect to the requirements.
12. Has an evaluation of the software architecture been documented?
    - Have alternative architectures been considered?
    - Have non-functional requirements also been considered?
    - *Negative indicators*:
        - High **complexity**: a component has a complex interface or functionality.
        - Low **cohesion**: a component contains unrelated functionality.
        - High **coupling**: two or more components have many (mutual) connections.
        - High **fan-in**: a component is needed by many other components.
        - High **fan-out**: a component depends on many other components.
13. Is the **flexibility** of the architecture demonstrated?
    - How can it cope with likely changes in the requirements?
    - Have the most relevant change scenarios been documented?