Conducting and Reviewing the Software Design Model

The design model resides at the core of the software engineering process. It is the place where quality is built into the software (and the place where quality is assessed. For this checklist, the more questions that elicit a negative response, the higher the risk that the analysis model will adequately serve its purpose. . For this checklist, the more questions that elicit a negative response, the higher the risk that the design model will not adequately serve its purpose.

General issues:

- Does the overall design implement all explicit requirements? Has a traceability table been developed?
- Does the overall design achieve all implicit requirements?
- Is the design represented in a form that is easily understood by outsiders?
- Is design notation standardized? Consistent?
- Does the overall design provide sufficient information for test case design?
- Is the design created using recognizable architectural and procedural patterns?
- Does the design strive to incorporate reusable components?
- Is the design modular?
- Has the design defined both procedural and data abstractions that can be reused?
- Has the design been defined and represented in a stepwise fashion?
- Has the resultant software architecture been partitioned for ease of implementation? Maintenance?
- Have the concepts of information hiding and functional independence been followed throughout the design?
- Has a *Design Specification* been developed for the software?

For data design:

- Have data objected defined in the analysis model been properly translated into required data structured?
- Do the data structures contain all attributes defined in the analysis model?
- Have any new data structures and/or attributes been defined at design time?
- How do any new data structures and/or attributes related to the analysis model and to overall user requirements?
- Have the simplest data structures required to do the job been chosen?
- Can the data structures be implemented directly in the programming language of choice?
- How are data communicated between software components?
- Do explicit data components (e.g., a database) exist? If so, what is their role?

For architectural design:

- Has a library of architectural styles been considered prior to the definition of the resultant software architecture?
- Has architectural tradeoff analysis been performed?
- Is the resultant software architecture a recognizable architectural style?
- Has the architecture been exercised against existing usage scenarios?
- Has an appropriate mapping been used to translate the analysis model into the architectural model?
- Can quality characteristics associated with the resultant architecture (e.g., a factored call-and-return architecture) be readily identified from information provided in the design model?

For user interface design:

- Have the results of task analysis been documented?
- Have goals for each user task been identified?
- Has an action sequence been defined for each user task?
- Have various states of the interface been documented?
- Have objects and actions that appear within the context of the interface been defined?
- Have the three "golden rules" (SEPA, 5/e, p. 402) been maintained throughout the GUI design?
- Has flexible interaction been defined as a design criterion throughout the interface?
- Have expert and novice modes of interaction been defined?

- Have technical internals been hidden from the causal user?
- Is the on-screen metaphor (if any) consistent with the overall applications?
- Are icons clear and understandable?
- Is interaction intuitive?
- Is system response time consistent across all tasks?
- Has an integrated help facility been implemented?
- Are all error message displayed by the interface easy to understand? Do they help the user resolve the problem quickly?
- Is color being used effectively?
- Has a prototype for the interface been developed?
- Have user's impressions of the prototype been collected in an organized manner?

For component-level design:

- Have proof of correctness techniques (SEPA, 5/e, Chapter 26) been applied to all algorithms?
- Has each algorithm been "desk-tested" to uncover errors? Is each algorithm correct?
- Is the design of the algorithm consistent with the data structured that the component manipulates?
- Have algorithmic design alternatives been considered? If yes, why was this design chosen?
- Has the complexity of each algorithm been computed?
- Have structured programming constructs been used throughout?