Checklists for Chapter 3 & 4 : Provided by Steve McConnell

The following checklists are compiled from Chapters 3 and 4. Do not feel the need to memorize all of this information, but this may be helpful to just show-off what some questions may look like when you are developing these items!

CHECKLIST: REQUIREMENTS

The requirements checklist contains a list of questions to ask yourself about your project's requirements. This book doesn't tell you how to do good requirements development, and the list won't tell you how to do one either. Use the list as a sanity check at construction time to determine how solid the ground that you're standing on is—where you are on the requirements Richter scale.

Not all of the checklist questions will apply to your project. If you're working on an informal project, you'll find some that you don't even need to think about. You'll find others that you need to think about but don't need to answer formally. If you're working on a large, formal project, however, you may need to consider every one.

SPECIFIC FUNCTIONAL REQUIREMENTS

❑ Are all the inputs to the system specified, including their source, accuracy, range of values, and frequency?

❑ Are all the outputs from the system specified, including their destination, accuracy, range of values, frequency, and format?

❑ Are all output formats specified for Web pages, reports, and so on?

❑ Are all the external hardware and software interfaces specified?

❑ Are all the external communication interfaces specified, including handshaking, error-checking, and communication protocols?

❑ Are all the tasks the user wants to perform specified?

❑ Is the data used in each task and the data resulting from each task specified?

SPECIFIC NONFUNCTIONAL (QUALITY) REQUIREMENTS

❑ Is the expected response time, from the user's point of view, specified for all necessary operations?

❑ Are other timing considerations specified, such as processing time, datatransfer rate, and system throughput?

❑ Is the level of security specified?

❑ Is the reliability specified, including the consequences of software failure, the vital information that needs to be protected from failure, and the strategy for error detection and recovery?

❑ Are minimum machine memory and free disk space specified?

❑ Is the maintainability of the system specified, including its ability to adapt to changes in specific functionality, changes in the operating environment, and changes in its interfaces with other software?

❑ Is the definition of success included? Of failure?

REQUIREMENTS QUALITY

❑ Are the requirements written in the user's language? Do the users think so?

❑ Does each requirement avoid conflicts with other requirements?

❑ Are acceptable tradeoffs between competing attributes specified—for example, between robustness and correctness?

❑ Do the requirements avoid specifying the design?

❑ Are the requirements at a fairly consistent level of detail? Should any requirement be specified in more detail? Should any requirement be specified in less detail?

❑ Are the requirements clear enough to be turned over to an independent group for construction and still be understood? Do the developers think so?

❑ Is each item relevant to the problem and its solution? Can each item be traced to its origin in the problem environment?

❑ Is each requirement testable? Will it be possible for independent testing to determine whether each requirement has been satisfied?

❑ Are all possible changes to the requirements specified, including the likelihood of each change?

REQUIREMENTS COMPLETENESS

❑ Where information isn't available before development begins, are the areas of incompleteness specified?

❑ Are the requirements complete in the sense that if the product satisfies every requirement, it will be acceptable?

❑ Are you comfortable with all the requirements? Have you eliminated requirements that are impossible to implement and included just to appease your customer or your boss?

CHECKLIST : ARCHITECTURE

Here's a list of issues that a good architecture should address. The list isn't intended to be a comprehensive guide to architecture but to be a pragmatic way of evaluating the nutritional content of what you get at the programmer's end of the software food chain. Use this checklist as a starting point for your own checklist. As with the requirements checklist, if you're working on an informal project, you'll find some items that you don't even need to think about. If you're working on a larger project, most of the items will be useful.

SPECIFIC ARCHITECTURAL TOPICS

❑ Is the overall organization of the program clear, including a good architectural overview and justification?

❑ Are major building blocks well defined, including their areas of responsibility and their interfaces to other building blocks?

❑ Are all the functions listed in the requirements covered sensibly, by neither too many nor too few building blocks?

❑ Are the most critical classes described and justified?

❑ Is the data design described and justified?

❑ Is the database organization and content specified?

❑ Are all key business rules identified and their impact on the system described?

❑ Is a strategy for the user interface design described?

❑ Is the user interface modularized so that changes in it won't affect the rest of the program?

❑ Is a strategy for handling I/O described and justified?

❑ Are resource-use estimates and a strategy for resource management described and justified for scarce resources like threads, database connections, handles, network bandwidth, and so on?

❑ Are the architecture's security requirements described?

❑ Does the architecture set space and speed budgets for each class, subsystem, or functionality area?

❑ Does the architecture describe how scalability will be achieved?

❑ Does the architecture address interoperability?

❑ Is a strategy for internationalization/localization described?

❑ Is a coherent error-handling strategy provided?

❑ Is the approach to fault tolerance defined (if any is needed)?

3.6 Amount of Time to Spend on Upstream Prerequisites 55

❑ Has technical feasibility of all parts of the system been established?

❑ Is an approach to overengineering specified?

❑ Are necessary buy-vs.-build decisions included?

❑ Does the architecture describe how reused code will be made to conform to other architectural objectives?

❑ Is the architecture designed to accommodate likely changes?

GENERAL ARCHITECTURAL QUALITY

❑ Does the architecture account for all the requirements?

❑ Is any part overarchitected or underarchitected? Are expectations in this

area set out explicitly?

❑ Does the whole architecture hang together conceptually?

❑ Is the top-level design independent of the machine and language that will be used to implement it?

❑ Are the motivations for all major decisions provided?

❑ Are you, as a programmer who will implement the system, comfortable with the architecture?

CHECKLIST : UPSTREAM PREREQUISITES

❑ Have you identified the kind of software project you're working on and tailored your approach appropriately?

❑ Are the requirements sufficiently well defined and stable enough to begin construction? (See the requirements checklist for details.)

❑ Is the architecture sufficiently well defined to begin construction? (See the architecture checklist for details.)

❑ Have other risks unique to your particular project been addressed, such that construction is not exposed to more risk than necessary?

CHECKLIST : MAJOR CONSTRUCTION PRACTICES

CODING

❑ Have you defined how much design will be done up front and how much will be done at the keyboard, while the code is being written?

❑ Have you defined coding conventions for names, comments, and layout?

❑ Have you defined specific coding practices that are implied by the architecture, such as how error conditions will be handled, how security will be addressed, what conventions will be used for class interfaces, what standards will apply to reused code, how much to consider performance while coding, and so on?

❑ Have you identified your location on the technology wave and adjusted your approach to match? If necessary, have you identified how you will program into the language rather than being limited by programming in it?

TEAMWORK

❑ Have you defined an integration procedure—that is, have you defined the specific steps a programmer must go through before checking code into the master sources?

❑ Will programmers program in pairs, or individually, or some combination of the two?

QUALITY ASSURANCE

❑ Will programmers write test cases for their code before writing the code itself?

❑ Will programmers write unit tests for their code regardless of whether they write them first or last?

❑ Will programmers step through their code in the debugger before they check it in?

❑ Will programmers integration-test their code before they check it in?

❑ Will programmers review or inspect each other's code?

TOOLS

❑ Have you selected a revision control tool?

❑ Have you selected a language and language version or compiler version?

❑ Have you selected a framework such as J2EE or Microsoft .NET or explicitly decided not to use a framework?

❑ Have you decided whether to allow use of nonstandard language features?

❑ Have you identified and acquired other tools you'll be using—editor, refactoring tool, debugger, test framework, syntax checker, and so on?