**CODE REVIEW CHECKLIST: MODULES (p. 1 of 1)**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **ITEM** | **PRESENT?**  **Y, N, N/A** |
| **Module header** | The module must have a module header block containing: |  |
|  | **File name:** |  |
|  | **Original creator:** |  |
|  | **Date created:** |  |
|  | **Person who last changed code** (if different from creator) |  |
|  | **Code revision number and change history** (with dates).  NOTE: The name of each changed unit should be listed  NOTE: If a change is made to correct a defect, the number or ID of the defect corrected should be entered as well. |  |
|  | **High level description:** (explain the module's purpose, and the name/purpose of key data structures, variables, sub-functions used, etc.) |  |
|  | **Failure modes and effects analysis:** List types of failures which could occur in this module and result in a hazard to the patient. List the types of mitigation actions the software takes to prevent hazards from occurring. If these risks are documented in a separate document, reference it. |  |
|  |  |  |
| **Module definitions and declarations** | **Grouping:** Definitions and declarations should be separated into distinct groups, each with a comment header.  For example, #defines, #includes, constant definitions, local function prototypes, etc. would all be grouped separately.  If required for greater logical clarity, however, related definitions and declarations may be mixed |  |
|  | **Commenting:** Each definition or declaration should have an associated descriptive comment unless the declaration is really obvious. |  |

**CODE REVIEW CHECKLIST: Units (p. 1 of 2)**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **ITEM** | **PRESENT?**  **Y, N, N/A** |
| **Function header block** | Every function (Unit) must have a comment header block containing: |  |
|  | **Function name** |  |
|  | **Change history:**  List of each change to the unit, with the date of the change and the name of the person making the change. Reference defect numbers or ID if the change was to correct a defect. |  |
|  | **Purpose:**  A short description of the unit's purpose. The description should be written such that the unit's purpose in fulfilling the original software requirements in the SRS can be understood. |  |
|  | **I/O description:** A description of the inputs and outputs expected, specifying their acceptable ranges. |  |
|  | **Return value:** A description of the return value |  |
|  | **External variables:** A description of any external variables used, specifying acceptable ranges. |  |
|  | **Unit design/algorithm:**  A more detailed description of the unit's processing. Should be detailed enough that reviewers can determine whether the code meets its design, but not so detailed that the description is just pseudo-code. |  |
|  | **Failure modes analysis:** A list of possible failure modes resulting in hazards or error conditions, and any mitigation actions this unit is required to take (for example, range checking a data value before use.) |  |
| **Lint results** | As noted in the Stage 1 Preparation instructions, Lint should have been run on the module or unit(s). The final Lint output should be recorded as part of the formal review meeting.  Each warning or message produced by Lint should have been inspected and any issues corrected. The items listed below must be checked for.  'Global wrap-up' output can be discarded and ignored for code review. |  |
|  | Loop index not modified within the loop |  |
|  | No extraneous code exists |  |
|  | All data references defined, computed, or obtained from external source. |  |
|  | All defined and referenced calling sequence parameters agree. |  |

**CODE REVIEW CHECKLIST: Units (p. 2 of 2)**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **ITEM** | **PRESENT?**  **Y, N, N/A** |
| **Code Checks** |  |  |
|  | Descriptive comments are accurate and informative. |  |
|  | Return values (in particular error returns) are not ignored. |  |
|  | Constants and literals are not hard coded. |  |
|  | All variables used have obvious or descriptive names, and correct scope. |  |
|  | Local functions and non-automatic variables are declared static. |  |
|  | System global functions have the module name as a prefix to the unit name. |  |
|  | All functions have prototypes (compiler checks this). |  |
|  | Data structure fields are described and commented clearly. |  |
|  | Code is logically correct (Code performs intended functions, operates correctly) |  |
|  | Numerical methods are sufficient |  |
|  | Accuracy of control outputs to external devices are within tolerance |  |
|  | System I/O mechanisms are consistently used. |  |
|  | Standard module communication techniques are used (e.g. use of message system) |  |
|  | Errors are detected and handled, and processing continued |  |
|  | Error handling conventions are followed (standard use of error handling task, etc.) |  |
|  | Input values (or other data used) are checked for reasonableness before use |  |
|  | Where necessary, critical output parameters or data are checked for reasonableness during processing |  |
|  | Code pays attention to recovery from potential hardware faults (e.g. arithmetic faults, power failure, clock). |  |
|  | Code pays attention to recovery from device errors. |  |
|  | There is no redundant code. |  |
|  | The structure is clean and indentations correct. |  |
|  | Over complication is avoided. |  |
| **SDS Check** | SDS (Software Design Specification) info for this unit is accurate |  |