Feature oriented documentation

This is a documentation guild line derived from the process model for the implementation of a feature into a program. The basic assumption behind these guild lines is that the purpose for documentation is to maintain the FEATURES of an application or peace of code, **not** the code itself.

The goals of documentation include
• convey the concepts and ideas which went into the implementation of a feature
• identify the scope of the implementation on the rest of the code
• collect programming "wisdom" on topics of interest to developers
• collect warnings and gotchas
• provide the records needed to improve the development process model on an individual basis
• explain the non-obvious

The following is a list of documentation which should be generated for each non-trivial task performed of feature implemented.

OFF-LINE Documentation:
1) Operational Goals
2) Fundamental, "key", or cornerstone architectural requirements
3) Model of the implementation fulfilling these key requirements
4) Impact/scope of the implementation on the existing body of code
5) Coding notes (gotchas, warnings, process thoughts, items to revisited later...)
6) Testing notes(bug types, what made a bug hard to fix, what could have been done to catch it sooner....)
7) Process notes (what process did you follow, could it be improved)

IN-LINE Documentation:
1) Identify the cornerstone items and reference them back to the model (off line documentation item number 3)
2) Warnings, gotchas, bug work arounds, suggestions for further enhancements
3) Normal in-line documentation explaining mystery variables, control structures, and process flows.
4) Identify the purpose for the existence of minor peace's of code, (what do the support?)

This standard is intended be used in a 2 step process for the delivery of a product. The fist step is to generate and use this documentation in support of the development efforts, as such generate all of the items listed. The second step is to use the documentation to generated deliverable product documentation, as such the off-line items 5,6,7 need not be included. However for deliverable product documentation effort must be made to verify the documentation with the actual implementation.

Process Model for programming task execution

1) Verify purpose of doing the task

2) Determine the operational requirements/goals

3) Determine the "key" architectural requirements implied by the goals

3.5) Identify the key abstractions involved with these architectural requierments (when applicable)

4) Develop models of possible designs which may meet the "key" requirements.

4.5) Follow the Booch OOD steps (when applicable)

5) Evaluate the designs and select a good one.

6) Focus on the chosen model and enhance it and sketch out its implementation

7) Jump into the coding of the architectural corner stones as enhancements continue.

8) continue development to fill in the details via a "follow your nose" process.

8.1) When most of (or all) the "follow your nose" code has been written/partially tested at Level 1, and the remaining features to be added (that's the stuff which isn't easy to do without thought) , initiate a review of what model is being implemented (or what path have you started down).

8.2) Convince yourself that the path your on will result in the successful meeting of the operational goals. If you don't think so then START OVER!!!!

8.3) If path still looks good, fill out documentation on model presently implemented, and implement Level 1 testing.

8.4) Start addressing the non-trivial components of the model being implemented as if they are new tasks (which they are).

9) Level 1 testing (compile, link, warnings, read test, comments)

10) Level 2 testing (integration, operational, side effect)

11) Task verification and completion of integration.

12) Documentation clean-up, verification, and writing.

13) Process evaluation, what could be improved upon?

Off-Line Documentation template:

1) Operational Goals
task spec

2) Fundamental, "key", or cornerstone architectural requirements (POSTMORTEM)
structured analysis from a supper high level

2.5) Key abstractions involved with the implementation of these key requirements (when applicable)
provide the language for the problem domain

3) Language independent Model of the implementation fulfilling these key requirements (POSTMORTEM)
explain the design

3.5) Explain roles of the Key abstractions USED or IMPLEMENTED in the model (when applicable)
explain the OOD

4) Impact/scope of the implementation on the existing body of code (POSTMORTEM)

5) Coding notes (gotchas, warnings, process thoughts, items to revisited later...)

6) Testing notes(bug types, what made a bug hard to fix, what could have been done to catch it sooner....)

7) Process notes (what process did you follow, could it be improved)

From a maintence point of view it seems that there are more points of view than I originaly thought. The goal is to maintane the features, the question is what kinds of information is needed to do it. One set of information is a mapping from features to implementation with a communiction of what the key requiements the feature represented to the developer and a model of how the developer meet these requierments in implementation. Another set of information is the key abstractions involved and created in the implementation of the key requierments.

So the maintence process requiers a mapping from features to key requeriments and abstractions to code, and the reviers mappings