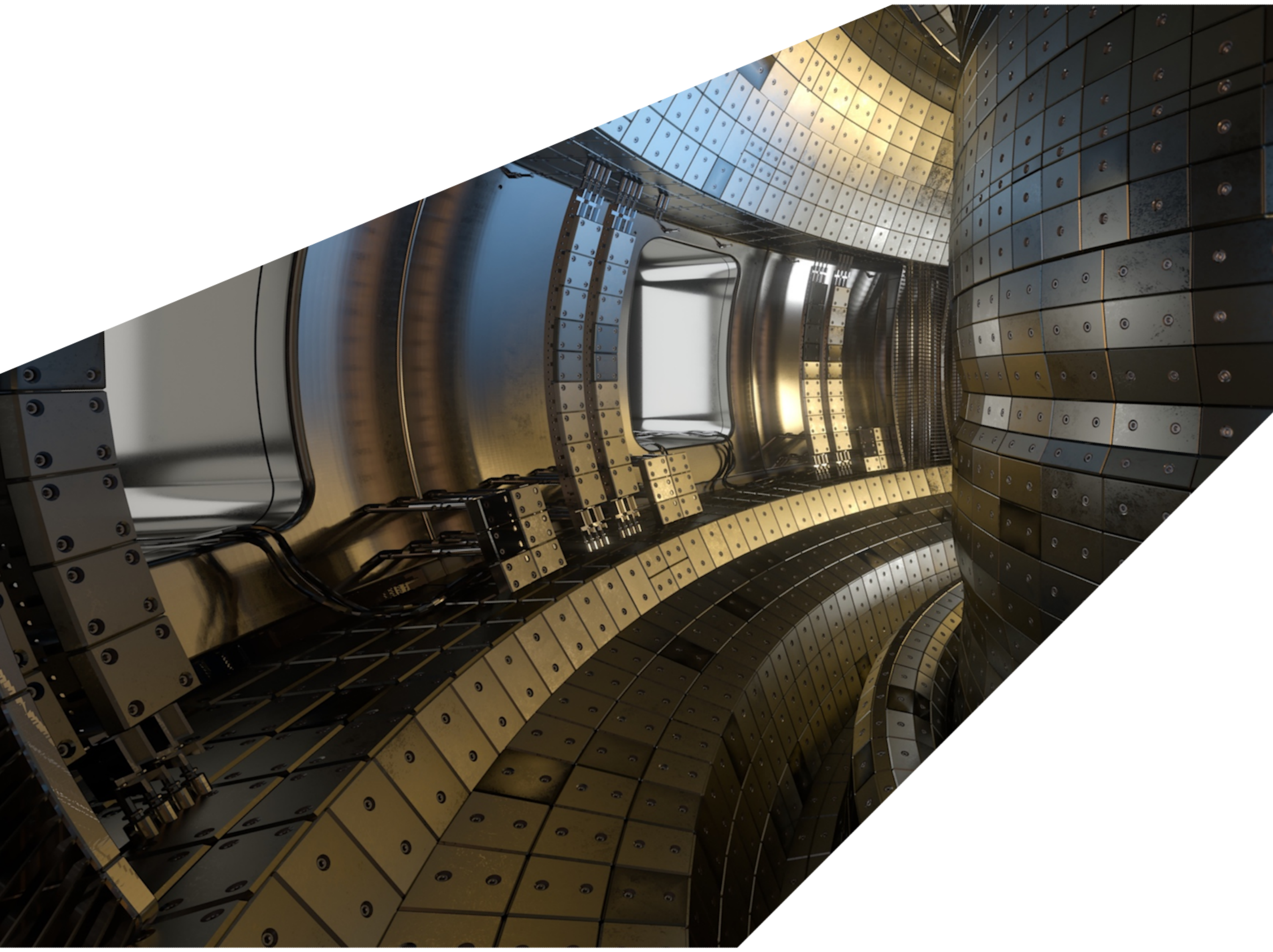
UK Atomic
Energy
Authority

ExCALIBUR

Development Plan

D3.4

Abstract
The report describes work for ExCALIBUR project NEPTUNE at Milestone M3.4. This report surveys the textbook and standards literature regarding the documents that should be produced as part of a major software development. The focus is on documents that should be produced before development starts, and on living documents that may require amendment as the development proceeds. The documents serve to define a software development process that is provisional in that it awaits confirmation from grantee reports. The key infrastructure for the next step of NEPTUNE is expected to be web-based, and an investigation into the suitability of the PANDOC and the LaTeX2HTML tools for generating web-based material is reported, with the latter to be preferred.



UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0033
	Issue:	1.10
	Date:	25 October 2021
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Ed Threlfall Wayne Arter Joseph Parker Will Saunders BD	N/A N/A N/A N/A	25 October 2021 25 October 2021 25 October 2021 25 October 2021
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		25 October 2021
Approved By:	Rob Akers Advanced Computing Dept. Manager		25 October 2021

1 Introduction

Drawing on the lessons learnt from earlier NEPTUNE work, especially ref [1], not only will the software will be opensource, but in the interests of encouraging as wide a possible range of developers, NEPTUNE will be treated as a community project with maximal transparency regarded decision-making, resource allocation etc. Successful examples of such projects set out principles that community members are expected to adhere to, and guidelines for them to follow. This amounts to significantly more detail both concerning the management of the development and the production of the software, than appears in the NEPTUNE Science Plan and Charter [2, 3] and the subsequent initial reports [4, 5].

The present report looks in detail at a number of sources recommending what documents are required for the successful conduct of a software development project. Following from the report [4], the first of these is the book chapter by Spencer Smith [6], as specifically directed to *scientific* software. The conclusion of the report [1, § 3] draws attention to Sommerville [7] who has relevant material dispersed throughout the book and Hewitt [8], although these are aimed at a wider, more commercially-oriented range of developments. Focussing on science, specifically space applications there is the European Space Agency (ESA) standards document [9] which in a “lite” form has proved successful in two UKAEA developments this century, viz. FISPACT [10] and SMARDDA [11].

None of these sources is entirely suitable for NEPTUNE. The proposed “faking” of documents by Smith seems unsatisfactory. Hurried writing is suggested by the inconsistency between his statements that no-one likes documenting software and his recommendations that more documents be produced, a suggestion supported by the fact that the proposed division of material between various documents seems unsatisfactory in some places, eg. where the functional and nonfunctional requirements are combined together in one section of a document entitled the “Software Requirements Specification”. As a text-book, Sommerville offers a multitude of possibilities, but few are relevant to open-source scientific software; therefore Hewitt scores by being more focussed and by an emphasis on awareness of the surrounding business environment. The ESA approach does not meet NEPTUNE needs in that it assumes a single expert customer for the project, and anyway the full standard, intended for flight-critical software, is arguably too onerous, where the “lite” version is more appropriate for a contractor with a single worker.

In addition to the points in the preceding paragraph, none of the sources fully addresses the issue that scientist developers do not like producing documentation. Ideally, material should be written up only once to be presented to developers and users of whatever level of competence as appropriate. This is probably best addressed by producing documentation as web pages, as indeed indicated by both Spencer Smith and Hewitt.

The main Section 2 provides a concordance of the documents from the texts [12, 6, 7] in Section 2.1, then gives more details as to what information should be provided in each in Section 2.2. It is followed by discussion of how best to format the documentation as a website in Section 2.3.

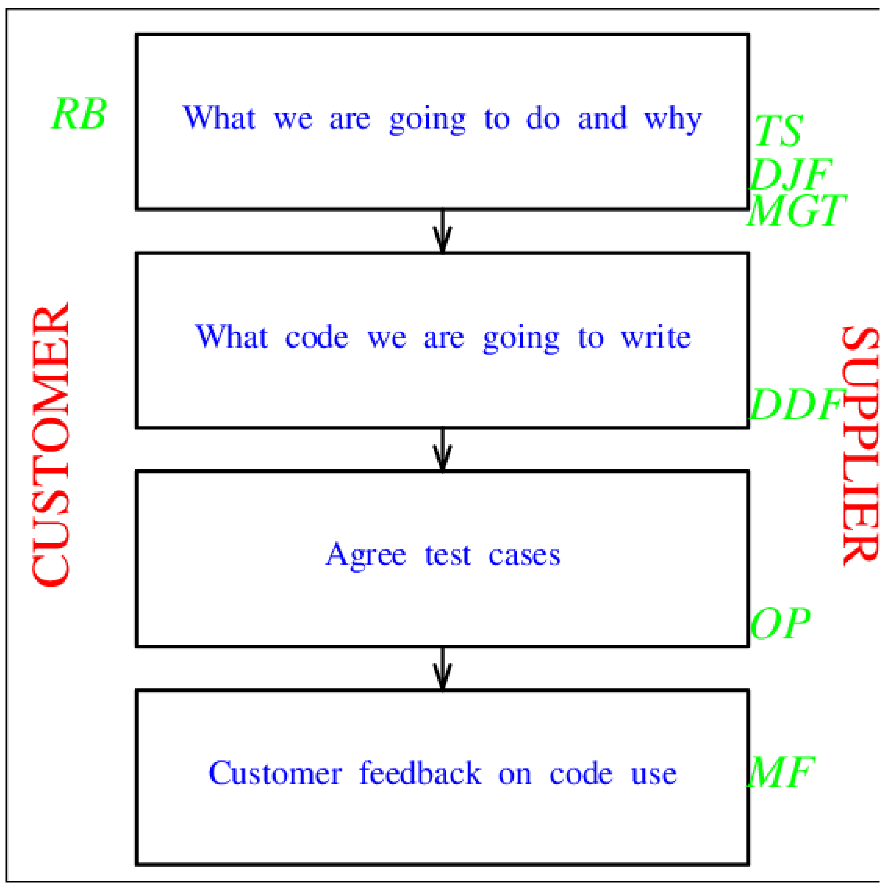

RB
What we are going to do and why
TS
DJF
MGT
CUSTOMER
What code we are going to write
SUPPLIER
DDF
Agree test cases
OP
Customer feedback on code use
MF

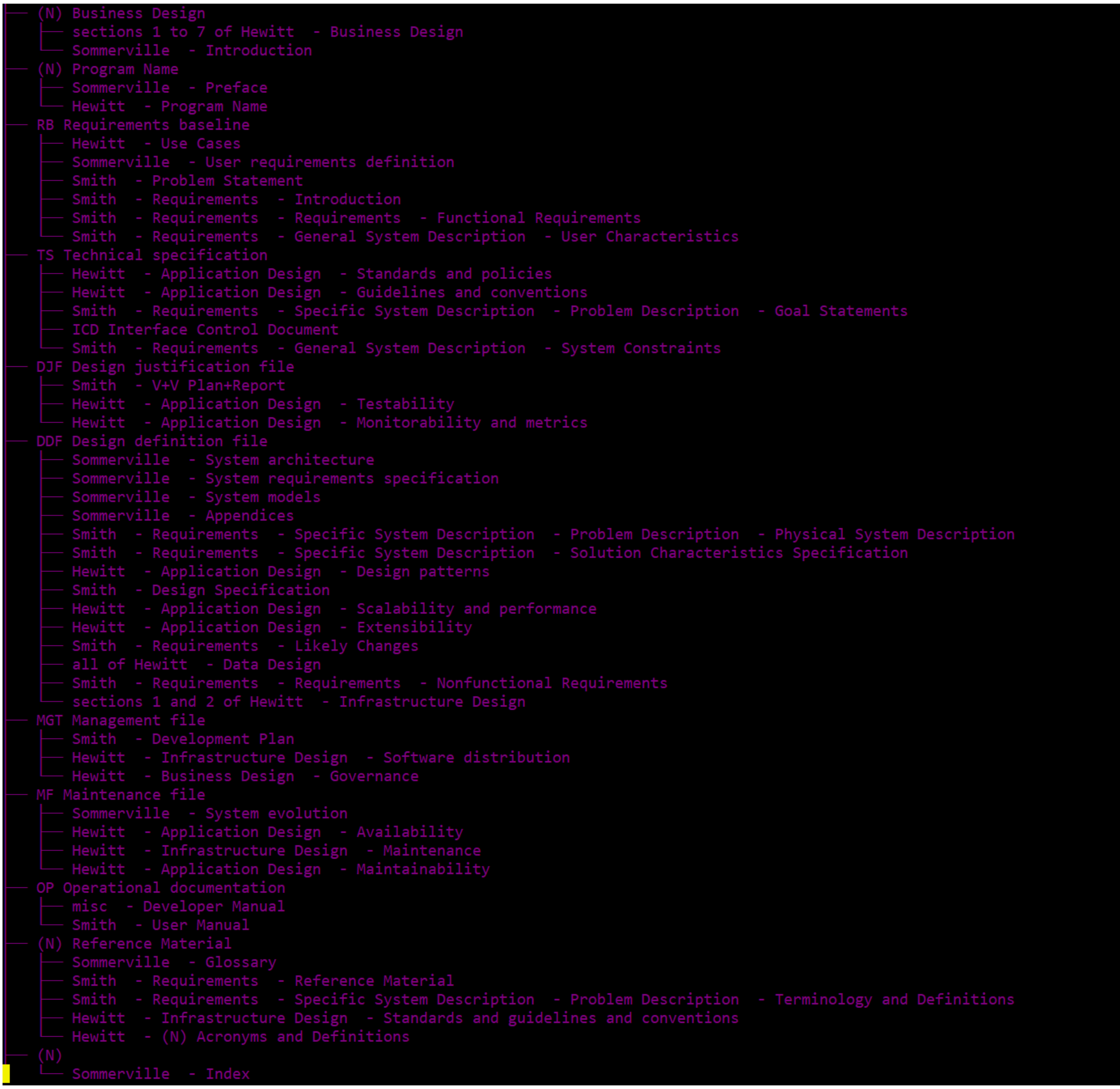
Figure 1: Documents according to ECSS-E-ST-40C [9], which is based on a customer-supplier relationship. The two- and three-letter acronyms indicate document titles as explained in Section 2.2.

2 Task Work

2.1 Concordance

The ECSS-E-ST-40C document or ESA standard for short is shown schematically in Figure 1. As the report's first author has developed software using a 'lite' version of this standard [13], it is taken as the reference point. The documents and sections of documents from the other texts [8, 6, 7] are are thus grouped according to the ESA documents, as shown in Figure 2. The other major reference consulted for design of scientific software, namely Rouson et al [14], mainly promotes use of software patterns, albeit illustrated by UML diagrams as advocated by refs [8, 7].

The ESA standard [9] is central because of personal experience that it works, at least in the "lite" version, seemingly for the reasons that it encourages conversation between the supplier/paymaster and the developer. These interactions lead to documentation that sets out an agreed scope for the software *before* each stage of development, thus avoiding the project 'creep' and on-the-fly re-design that may fatally slow code production. The Design Justification File (DJF, see Section 2.2.5)



```
├── (N) Business Design
│   ├── sections 1 to 7 of Hewitt  - Business Design
│   └── Sommerville  - Introduction
├── (N) Program Name
│   ├── Sommerville  - Preface
│   └── Hewitt  - Program Name
├── RB Requirements baseline
│   ├── Hewitt  - Use Cases
│   ├── Sommerville  - User requirements definition
│   ├── Smith  - Problem Statement
│   ├── Smith  - Requirements  - Introduction
│   ├── Smith  - Requirements  - Requirements  - Functional Requirements
│   └── Smith  - Requirements  - General System Description  - User Characteristics
├── TS Technical specification
│   ├── Hewitt  - Application Design  - Standards and policies
│   ├── Hewitt  - Application Design  - Guidelines and conventions
│   ├── Smith  - Requirements  - Specific System Description  - Problem Description  - Goal Statements
│   ├── ICD Interface Control Document
│   └── Smith  - Requirements  - General System Description  - System Constraints
├── DJF Design justification file
│   ├── Smith  - V+V Plan+Report
│   ├── Hewitt  - Application Design  - Testability
│   └── Hewitt  - Application Design  - Monitorability and metrics
├── DDF Design definition file
│   ├── Sommerville  - System architecture
│   ├── Sommerville  - System requirements specification
│   ├── Sommerville  - System models
│   ├── Sommerville  - Appendices
│   ├── Smith  - Requirements  - Specific System Description  - Problem Description  - Physical System Description
│   ├── Smith  - Requirements  - Specific System Description  - Solution Characteristics Specification
│   ├── Hewitt  - Application Design  - Design patterns
│   ├── Smith  - Design Specification
│   ├── Hewitt  - Application Design  - Scalability and performance
│   ├── Hewitt  - Application Design  - Extensibility
│   ├── Smith  - Requirements  - Likely Changes
│   ├── all of Hewitt  - Data Design
│   ├── Smith  - Requirements  - Requirements  - Nonfunctional Requirements
│   └── sections 1 and 2 of Hewitt  - Infrastructure Design
├── MGT Management file
│   ├── Smith  - Development Plan
│   ├── Hewitt  - Infrastructure Design  - Software distribution
│   └── Hewitt  - Business Design  - Governance
├── MF Maintenance file
│   ├── Sommerville  - System evolution
│   ├── Hewitt  - Application Design  - Availability
│   ├── Hewitt  - Infrastructure Design  - Maintenance
│   └── Hewitt  - Application Design  - Maintainability
├── OP Operational documentation
│   ├── misc  - Developer Manual
│   └── Smith  - User Manual
├── (N) Reference Material
│   ├── Sommerville  - Glossary
│   ├── Smith  - Requirements  - Reference Material
│   ├── Smith  - Requirements  - Specific System Description  - Problem Description  - Terminology and Definitions
│   ├── Hewitt  - Infrastructure Design  - Standards and guidelines and conventions
│   └── Hewitt  - (N) Acronyms and Definitions
├── (N)
    └── Sommerville  - Index
```

Figure 2: Concordance of documents needed during software development, grouped according to ECSS-E-ST-40C [9] - or labelled ‘(N)’, texts as follows: Hewitt “Semantic Software Design” [8], (Spencer) Smith “A Rational Document Driven Design Process for Scientific Software” [6], and Sommerville “Software Engineering” [7].

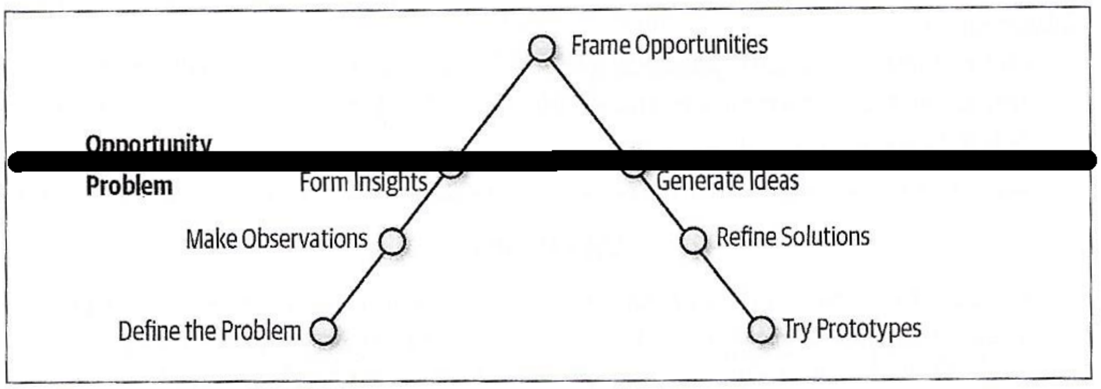


Figure 3: The Design Thinking process, Figure 4-1 from Hewitt [8]. Progression is from left to right: forming insights and generating ideas are both part of the problem and the opportunity.

produced in the process is unique to the ESA documents. Not only does it prevent untimely customer 'what-if's by setting out and analysing options to a specified time-table, but constitutes a useful fall-back should the agreed options turn out to be unworkable when external circumstances change unexpectedly.

Hewitt [8] has a good deal of discussion of philosophy which is interesting, amidst what personal experience indicates should be a very successful methodology for producing software in a commercial environment, an approach which is set out both succinctly and comprehensively. His "Design thinking" method should be equally useful for encouraging scientific as well as commercial creativity. This is illustrated in Figure 3, although more than one excursion to the tip of the "Λ" might be expected in a research context. It has to be accompanied by a careful recording of both the good and the bad ideas which emerge. Hewitt [8] is also very strong on identifying and managing stakeholders beyond the immediate ambit of the software project. He is understandably less succinct here, letting it emerge that the average manager is to be treated as someone with a short attention span, principally focussed on advancing his/her own career (which might include promoting a rival project), is technically ignorant and likely to demand a PowerPoint justifying any aspect of the project at a moment's notice. However, in any context, collaborators will appreciate short, to-the-point communications which are clearly presented and well argued. Hewitt notably also insists on having two reasons for every design decision, and producing documentation from which it is easy to generate presentations of 5-15 slides. For commercial reasons, there is also detailed consideration as to how to manage data and exploit Machine Learning (ML) tools, and how to ensure reliability of both the software and the hardware that it uses, all of which are germane to the Exascale.

Spencer Smith [6] naturally highlights what documents are needed in the development of *scientific* software. Smith provides reminders of the need to include details such as choice of physical units, and to plan validation and verification (V & V) of results. Last in the list of sources, Sommerville [7] is comprehensive and can be used as the authority on software terminology in what is still a rapidly expanding subject that can give rise to overlapping and even conflicting definitions.

2.2 Detailed Information

In this section, the documents labelled “N” are introductions upon the ESA standard for use in a community project. Citations in the present section to Hewitt refer specifically to ref [12, § 5], to Smith imply ref [6, § 2.2] and to Sommerville imply ref [7, § 4] unless stated otherwise.

2.2.1 Program Name PN (N)

Choice of name for the software is recognised as important, but this section should mainly aim to give the reader enough information about the purpose of the software that he/she can decide whether they want to look further. Hewitt also requires the equivalent of the approval box for a report, giving names of authors, collaborators and reviewers. At this point, Hewitt further introduces the RFC2119 subset of the Internet Engineering Task Force (IETF) keyword [15] for use throughout the document. This usage implies specific meanings for “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” when the words are capitalised. (It will not be adopted herein.)
Think preface to a textbook, plus document approval box.

2.2.2 Business Design BD (N)

This section should aim to make it clear to the reader why it woud be good to allocate funding for the software development. From Hewitt’s list:

- Capabilities - what new benefits arise
- Strategic fit - to UKAEA mission
- Business drivers - why are we doing this?
- Assumptions - regarding available funding
- Constraints - laws and regulations, mention software licensing
- Risks - if resources not available
- Impacts - new processes, training needed
- Stakeholders - who will win or lose by good or bad outcome

Think introductory chapter of a textbook.

2.2.3 Requirements Baseline RB

The ESA standard RB expresses the customer’s requirements, including interface. Sommerville devotes an entire chapter to requirements engineering, in contrast to Hewitt who instead recommends ‘observation’ of those working with the software intended for replacement, and the production of use cases to a detailed template, as follows:

- Overview
- Actors
- Relationship to other use cases
- Flow including input to output
- Special requirements

The collection of the above use cases then defines the so-called functional requirements, other requirements are referred to as non-functional. In the scientific case, Spencer Smith requires specification of the experiment and the assumptions to be used in modelling it. It is clear since (design) engineers talk of workflows, that Hewitt's approach has much to be recommended for a mixed engineering/scientific application.
This describes the use cases for the software.

2.2.4 Technical Specification TS

The ESA standard TS is the supplier's response to the RB, so provides probably the least useful ESA document description for a community process. Material ought nonetheless to be presented which encompasses the content of the ESA ICD or Interface Control Document. Other material that would also not be included elsewhere concerns the higher level aspects of Hewitt's Application Design and Smith's Software Requirements Specification [6, § 2.2.3]. Hewitt would include prescription of software standards such as C++17, and guidelines/conventions such as indicated for Object Fortran in ref [16]. Smith would specify what the software needs to compute to compare with experiment. The need for the code to have good in-line documentation capable of formatting by say DOXYGEN [17] will be described here. *This outlines inputs and outputs of the software, plus set outs principles for code-writing.*

2.2.5 Design Justification File DJF

The ESA standard DJF is generated and reviewed at all stages of the development and review processes. It contains the documents that describe the trade-offs, design choice justifications, verification plan, validation plan, validation testing specification, test procedures, test results, evaluations and any other documentation called for to justify the design of the supplier's product.
This describes why the equations, methods and algorithms were chosen, and the testing regime.

2.2.6 Design Definition File DDF

The ESA standard DDF is a supplier-generated file that documents the result of the design engineering processes. The DDF is the primary input to the CDR review process and it contains all the documents called for by the design engineering requirements.
This describes the equations, methods and algorithms to be used in software.

2.2.7 Management File MGT

The ESA standard MGT is a supplier generated file that describes the management features of the software project (for instance, organisational breakdown and responsibilities, work activities breakdown, selected life cycle, deliveries, milestones and risks). It also describes governance - eg. PRINCE2 [18].
This collates the project plans of all participants.

2.2.8 Maintenance File MF

The MF is a maintainer generated file that describes the planning and status of the maintenance, migration and retirement activities.
This is basically unnecessary given one of the common repository-based projects.

2.2.9 Operational Documentation OP

In the ESA standard OP, the user's experience of the software feeds back into the instructions as to how to use the software.
This includes the user manual, the developer manual and describes output of test cases.

2.2.10 Reference Material REF (N)

In a web-based system, this material can be presented stand-alone to avoid duplication throughout many different documents. It should provide an ontology for the project, by which it is meant a glossary of terminology used, meanings of acronyms and an index of mathematical notation.

2.2.11 Index IND (N)

The construction of an index is recommended by Sommerville, and although an alphabetic index is made redundant by web search engines, it might be helpful to have an index of diagrams and figures.

2.3 Production of .html

It has become recently very popular to construct web-based code documentation using 'Read the Docs' [19], which is open-source software that indeed produces a very attractive website, see eg. the plasmapy [20] website. The problem is the need to use reStructuredText or Markdown formats to input to 'Read the Docs'. For many people, both formats represent yet more markup languages to learn, and although many know Markdown, it has many slightly incompatible variants. Similar criticism applies to the use of DOXYGEN for producing documentation other than in-line at a relatively low level.

MS Word^{TM} is widely used by engineers to produce manuscripts for publication and has a built-in capability to write .html, but is proprietary software and although widely available has the disadvantage that even the modern (and html-friendly) .docx format uses an extended character representation which is unnecessary for technical work, and potentially hazardous in that non-printing and/or non-standard characters may be inadvertently cut and pasted eg. into software documentation, which expects only the restricted ASCII set. For the preceding reason, like most papers in mathematics and physics, the bulk of the documentation for the ExCALIBUR project NEPTUNE has been drafted using LaTeX. There are at least two open source tools for producing .html from LaTeXsource, namely PANDOC and LATEX2HTML.

The PANDOC project [21] has in the last few years (Version 2.5 from 2018, or later) developed to the point where it can turn relatively complex LaTeX documents into .html for display using web browsers, see Section 2.3.1 below. PANDOC also tightly defines (see Annex A, Section A) a variant of Markdown that it would be good to employ as a standard, since PANDOC can then use files conforming to this standard to produce output in many different formats. Unfortunately, the PANDOC development seems to have stalled in respect of LaTeX to .html conversion, whereas LATEX2HTML has had all the necessary features for many years.

LATEX2HTML [22] is packaged and freely available for both linux and MacOS. The main objection is that the default style of web-page now looks very dated. However the investigations in Section 2.3.2 indicate that this is a purely cosmetic objection, and should a ‘smarter’ appearance be thought desirable, easily met by employing experienced web developers such as those who produced the `excalibur.ac.uk` website. Moreover, it is outweighed by the ease with which webpages may be produced from the existing NEPTUNE report base.

2.3.1 PANDOC

As with many open source projects, a certain amount of experimentation is needed to locate suitable options for PANDOC, especially as it is under active development at the time of writing. The following bash shell script produces readable .html from a report including all the `\input` files with Version 2.5, current on Ubuntu as of March 2021:

```
pandoc $1.tex --metadata pagetitle=$1 --toc  \
--number-sections \
--bibliography ../bib/exc.bib \
--bibliography ../bib/reac.bib \
-s --mathml --default-image-extension=png -o $1.html
```

The `--verbose` option is useful as otherwise only terse error messages appear. PANDOC is strict in its interpretation of LaTeX, so that eg. `\mathbf` and `\texttt` need to be used instead of `\bf` and `\tt`, and extended character sets may cause PANDOC to fail. `\rotatebox` and `\protect` are not recognised and should be stripped from the .tex file using *sed* before processing.

For the power shell with PANDOC Version 2.12 dated 2021, the following produces readable .html from the report:

```
pandoc rp3.tex --metadata pagetitle=rp3 --toc ‘
--number-sections ‘
```

```
--bibliography ../bib/exc.bib ‘
--bibliography ../bib/reac.bib ‘
--bibliography ../bib/waynes.bib ‘
--bibliography ../bib/new.bib ‘
--bibliography ../bib/active.bib ‘
--bibliography ../bib/mc.bib ‘
-s --citeproc --mathml --default-image-extension=png -o rp3.html
```

However the `--mathjax` option needs to replace `--mathml` for mathematics to display correctly in the Microsoft Edge browser.

Unfortunately there are still significant issues regarding the .html produced, eg. equations are not numbered, and it is not easy to see how this and related omissions can be remedied without modification of the Haskell code underlying PANDOC. Nonetheless, the tool has demonstrated an ability to format equations and produce bibliographies that suggests that it could become a powerful tool in the production of documentation to underpin the NEPTUNE software development. Consideration should be given to making resource available to ensure PANDOC provides all necessary facilities in an easy-to-use wrapper. In the interim, however the preceding deficiencies indicate that LATEX2HTML is to be preferred.

2.3.2 LATEX2HTML

The package is simple to use, in that the following script will produce readable webpages in a separate Linux directory, provided that `rp1.pdf` has already been produced correctly in the original directory. (The proviso ensures that .aux and .bbl files have been produced for the document.)

```
latex2html -t "ExCALIBUR project NEPTUNE" -mkdir -dir web -prefix rp1 rp1
```

The key options are `-dir` to set the name of the directory in which to produce .html and supporting graphics and other files, and `-prefix` to distinguish these files from others produced for different documents. Option `-mkdir` saves a separate `mkdir` command, and option `-t` plays a minor role describing the linkage of the top level .html file contents to lower levels. Another useful option is `-split`, which specifies the level of LaTeX subdivision at which separate files are produced, thus `-split +1` causes each full section to occupy a separate .html file. (The default is to divide the document among many small files corresponding to each subdivision of the text, down at least to paragraph.) There is a working facility to link to other local files as described by Swan [23].

The vintage of the software is very noticeable. It was originally developed in the 1990s (see ref [24]) by N. Drakos, who no longer seems to be active. There are several attempts at documentation available from around the turn of the century on the web [24, 23, 25], none of which appears to be entirely reliable. Recently, other authors have taken the package forward, but their documentation seems incomplete [22]. Another point also noticed is that with the above script, LATEX2HTML produces a file index.html identical to rp1.html.

The main restrictions discovered in an exploratory investigation like that conducted for PANDOC, are that

- in figures, although according to Swan [23], `\rotatebox` should work, it does not.

- only a restricted set of packages, mostly those current at the turn of the century, is recognised, see list in Annex B (Section B).
- for TEXusers, there are problems with the use of `\def`.

The output in file `WARNINGS` typically notes an absence of recognition of the `helvet` and `fancyhdr` packages. This does not cause problems, apparently since neither package invokes special commands, and although for other unrecognised packages of course error messages might be expecte.d .from missing specials, the software commonly produces useful .html output. None of the restrictions above is expected to be very serious, since figures may be easily rotated, and LATEX has generally been very stable since the turn of the century.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] W. Arter, E. Threlfall, J. Parker, and S. Pamela. Report on user frameworks for tokamak multiphysics. Technical Report CD/EXCALIBUR-FMS/0022-M3.1.2, UKAEA, 2020.

[2] W. Arter, L. Anton, D. Samaddar, and R. Akers. ExCALIBUR Fusion Modelling System Science Plan. Technical Report CD/EXCALIBUR-FMS/0001, UKAEA, 2019. https://www.metoffice.gov.uk/binaries/content/assets/metofficegovuk/pdf/research/spf/ukaea-excalibur-fms-scienceplan.pdf.

[3] W. Arter. EXCALIBUR NEPTUNE Charter. Technical Report CD/EXCALIBUR-FMS/0020, UKAEA, 2020.

[4] L. Anton. NEPTUNE: Report on system requirements. Technical Report CD/EXCALIBUR-FMS/0014-1.00-M3.1.1, UKAEA, 2020.

[5] L. Anton. NEPTUNE: Background information and user requirements for design patterns. Technical Report CD/EXCALIBUR-FMS/0015-1.00-M3.3.1, UKAEA, 2020.

[6] W.S. Smith. A Rational Document Driven Design Process for Scientific Software. In J.C. Carver, N.P. Chue Hong, and G.K. Thiruvathukal, editors, *Software Engineering for Science*, pages 27–52. Chapman and Hall/CRC, 2017.

[7] I. Sommerville. *Software Engineering, Tenth Edition*. Pearson Education Limited, Harlow, 2016.

[8] E. Hewitt. *Semantic Software Design: A New Theory and Practical Guide for Modern Architects*. O'Reilly Media, 2019.

[9] J. Drabbe. ECSS-E-ST-40C - Software (6 March 2009). Technical report, ESA, 2009. https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/.

[10] J.-Ch. Sublet, J.W. Eastwood, J.G. Morgan, M.R. Gilbert, M. Fleming, and W. Arter. FISPACT-II: An Advanced Simulation System for Activation, Transmutation and Material Modelling. *Nuclear Data Sheets*, 139:77–137, 2017. http://dx.doi.org/10.1016/j.nds.2017.01.002, website: https://fispact.ukaea.uk/.

[11] W. Arter, E. Surrey, and D.B. King. The SMARDDA Approach to Ray-Tracing and Particle Tracking. *IEEE Transactions on Plasma Science*, 43(9):3323–3331, 2015. http://dx.doi.org/10.1109/TPS.2015.2458897.

[12] E. Hewitt. *Semantic Software Design: A New Theory and Practical Guide for Modern Architects*. O'Reilly Media, 2019.

[13] W. Arter. A Rough Guilde to Software Development, July 31, 2007. Unpublished note.

[14] D. Rouson, J. Xia, and X. Xu. *Scientific software design: the object-oriented way*. Cambridge University Press, 2011.

[15] Keywords for documentation. `https://www.ietf.org/rfc/rfc2119.txt`, 2021. Accessed: March 2021.

[16] W. Arter, J. Parker, and E. Threlfall. Module Guide. Technical Report CD/EXCALIBUR-FMS/0032-D3.3, UKAEA, 2021.

[17] D. van Heesch. doxygen Manual for version 1.9.1. Technical report, www.doxygen.org, 2021.

[18] Axelos Staff et al. *Managing Successful Projects with PRINCE2, 6th Edition*. ALEXOS, 2017.

[19] Create, host, and browse documentation. `https://readthedocs.org`, 2021. Accessed: March 2021.

[20] Open source Python ecosystem for plasma research and education. `https://www.plasmapy.org`, 2020. Accessed: August 2020.

[21] a universal document converter. `https://pandoc.org`, 2021. Accessed: March 2021.

[22] A utility that converts LaTeX documents to web pages in HTML. `https://www.latex2html.org`, on github as `https://www.github.com/latex2html/latex2html/`, 2021. Accessed: October 2021.

[23] H.W. Swan. The LaTeX2HTML translator: An Overview. `https://www.ntg.nl/cgi-bin/22.pdf`, 1996. Accessed: October 2021.

[24] N. Drakos and R. Moore. The LaTeX2HTML Translator. `https://mirrors.up.pt/pub/CTAN/obsolete/support/latex2html/manual.pdf`, 1999. Accessed: October 2021.

[25] A. Bridle. LaTeX2HTML Authors' Guide. Technical Report Online, NRAO, 2001. Accessed: October 2021.

A Annex A: Recommended Markdown Dialect

The following is an edited version of the Pandoc Cheatsheet by David Sanson, dated 31 January 2011. He remarks that it is incomplete and possibly incorrect, and is intended only for quick reference purposes.

```
Backslash Escapes
Except inside a code block or inline code, any punctuation or space character
preceded by a backslash will be treated literally, even if it would normally indicate
formatting.

Title Block
% title
% author(s) (separated by semicolons)
% date

Inline TeX and HTML

* TeX commands are passed through to Markdown, LaTeX and ConTeXt output;
otherwise they are deleted.
* HTML is passed through untouched but
  * Markdown inside HTML blocks is parsed as markdown.

Paragraphs and line breaks
* A paragraph is one or more lines of text separated by a blank line.
* A line that ends with two spaces, or a line that ends with an escaped new-line (a
backslash followed by a carriage return) indicates a manual line break.

Italics, bold, superscript, subscript, strikeout

*Italics* and **bold** are indicated with asterisks.
To ~~strikeout~~ text use double tildas.
Superscripts use carats, like so: 2^nd^.
Subscripts use single tildas, like so: H~2~O.
Spaces inside subscripts and superscripts must be escaped,
e.g., H~this\ is\ a\ long\ subscript~.

Inline TeX math and Inline Code
Inline TeX math goes inside dollar signs: $2 + 2$.
Inline code goes between backticks: ‘echo ’hello’‘.

Links and images
<http://example.com>
<foo@bar.com>
[inline link](http://example.com "Title")
![inline image](/path/to/image, "alt text")

[reference link][id]
[implicit reference link][]
![reference image][id2]

[id]: http://example.com "Title"
[implicit reference link]: http://example.com
[id2]: /path/to/image "alt text"

Footnotes
Inline notes are like this.^[Note that inline notes cannot contain multiple
paragraphs.] Reference notes are like this.[^id]
```

```
[^id]: Reference notes can contain multiple paragraphs.

  Subsequent paragraphs must be indented.

Citations
Blah blah [see @doe99, pp. 33-35; also @smith04, ch. 1].
Blah blah [@doe99, pp. 33-35, 38-39 and *passim*].
Blah blah [@smith04; @doe99].
Smith says blah [-@smith04].
@smith04 says blah.
@smith04 [p. 33] says blah.

Headers
Header 1
========

Header 2
--------

# Header 1 #
## Header 2 ##

Closing #s are optional. Blank line required before and after each header.

Lists
Ordered lists
1. example
2. example
A) example
B) example

Unordered lists
Items may be marked by ’*’, ’+’, or ’-’.

+ example
- example
* example

Lists may be nested in the usual way:
+ example
  + example
+ example

Definition lists
Term 1
~ Definition 1
```

```
Term 2
~ Definition 2a
~ Definition 2b

Term 1
: Definition 1
Term 2
: Definition 2
    Second paragraph of definition 2.

Blockquotes
> blockquote
>> nested blockquote

Blank lines required before and after blockquotes.

Tables
Right  Left     Center  Default
------ ------ --------- -------
    12 12        12          12
   123 123      123         123
     1 1         1            1

Table: Demonstration of simple table syntax.

(For more complex tables, see the pandoc documentation.)

Code Blocks
Begin with three or more tildes; end with at least as many tildes:
~~~~~~~
{code here}
~~~~~~~

Optionally, you can specify the language of the code block:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ {.haskell .numberLines}
qsort [] = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Horizontal Rules
3 or more dashes or asterisks on a line (space between okay)
---
* * *
- - - -
```

B Annex B: Styles recognised by LATEX2HTML

The packages supported may according to Swan's description [23], be deduced from the names of the .perl files in directory $LATEX2HTMLDIR/styles, where the variable is set on installation to a directory name eg. $LATEX2HTMLDIR=/usr/share. Users requiring other packages are invited to write the necessary .perl scripts after studying those provided. Note that the majority of packages supported were extant around the turn of the century. For descriptions of the packages that remain available, see `https://ctan.org/pkg`.

The following packages are supported and may be useful for scientific work in English.

```
abnt
alltt
amsart
amsbook
amsfonts
amsmath
amssymb
amstex
article
book
changebar
colordvi
color
enumerate
epsbox
epsfig
eurosym
floatfig
floatflt
fontenc
frames
getimagesize
graphics
graphicx
harvard
heqn
hthtml
htmllist
html
hyperref
inputenc
justify
```

```
latexsym
letter
longtable
lyx
makeidx
multicol
natbib
psfrag
report
seminar
SIunits
slides
supertabular
texdefs
texnames
textcomp
verbatimfiles
verbatim
webtex
wrapfig
xspace
```