LICENSING GUIDE

No.	Title	Rev
LG-1045	Guidance for Licensing Submissions involving Computer Software and Evaluation Models for Safety Calculations	0

APPROVED: DATE: 25/04/02

CHIEF EXECUTIVE OFFICER

Contents

List of Figures

1 Introduction

This guidance document supplements RD-0016 “Requirements for Licensing Submissions Involving Computer Codes and Evaluation Models for Safety Calculations” /1/ and it provides more detail concerning the V&V process based on international guidance and best practise.

The V&V recommendations in this document serve as guidance only and other approaches are permissible. However, major departures from this guidance should be justified in the relevant V&V documentation made available to NNR.

Wherever this document defines recommendations which are similarly addressed as requirements in /1/, the requirements of /1/ overrule the guidance collated in this document.

In order to establish the Safety Case for the PBMR, its behaviour has to be analysed for different operational steady states and for transients which are initiated by a multitude of postulated events. Usually the analyses to be performed reach a degree of complexity which forces the application of computer software programs.

2 Definitions and Abbreviations

ALARA As Low As Reasonably Achievable

Calculation Model An analytical representation or quantification of a real system used to predict or assess the behaviour of the real system under specified conditions. Where a Software Product is used, the Calculation Model is the combination of the System Model and the Software Program.

Category A events Category A events (or combinations of events) according to LG-1037 /12/ are those which lead to exposure and which occur with a frequency not less than one in one hundred years ($\geq 10^{-2}$ y^{-1}). Such events are treated as part of normal operations.

Category B events Category B events (or combinations of events) according to LG-1037 are those which lead to exposure and which occur with a frequency of between one in one hundred years (10^{-2} y^{-1}) and one in one million years (10^{-6} y^{-1}).

Category C events Category C events (or combinations of events) according to LG-1037 are all possible events which could lead to exposure. As such Category C events will include Category A and B events as well as events with an annual frequency of less 10^{-6}.

COTS Software Commercial Off-the-shelf software.

DiD Defense in Depth, defined as a hierarchical deployment of different levels of equipment and procedures in order to maintain the effectiveness of physical barriers placed between a radiation source or radioactive materials and workers, members of the public or the environment, in operational states and, for some barriers, in accident conditions.

EM Evaluation Model.

Evaluation Model A calculational framework consisting of one or more Calculation Models and specific input data used to model specific system behaviour under certain conditions.

GUI Graphical User Interface.

IE Initiating Event, defined as an event identified during design which can potentially challenge the design of the plant.

LBE Licensing Basis Events, defined as those events, enveloping all events, from categories A, B, C that shall be analysed and the results thereof shall demonstrate compliance with LG-1037 (doses, risk,

ALARA, DiD).

Legacy Software Software that has been developed in the past for specific applications in the development of predecessors of the PBMR. Due to its history Legacy Software may be poorly documented and need substantial V&V efforts.

New Software Software that is under development and can therefore be submitted to a full scale V&V process during all stages of development.

ODE ordinary differential equation.

PDE partial differential equation.

PIE Postulated Initiating Event, defined as an identified initiating event that leads to anticipated operational occurrences or accident conditions. PIE are the enveloping IE (covering one or several IE) and/or combinations of IE but excluding mitigators. Based on justified frequencies, taking uncertainties into account, the PIE will be allocated to the categories A, B, and C (as defined in LG-1037). The comprehensive set of PIE forms the basis for both deterministic and probabilistic safety analyses.

physical process inseparable combination of physical or chemical phenomena which describes a particular sector of the totality of processes or a particular time range in the behaviour of the PBMR.

PRA Probabilistic Risk Assessment

Program Software program.

Safety analysis Is the means to demonstrate the attainment of the safety objectives at a nuclear site. In the context of this guidance it is all calculations or other analysis pertaining to application of a Calculation Model or Evaluation model linked directly or indirectly to the justification of the PBMR Safety Case.

Software One or more software programs

Software program Compiled or interpreted computer executable or script

Software product(s) One or more of the set of software, software documentation, data, procedures and other materials, information and components pertaining to the use of software.

SSC Systems, Structures and Components.

SVVP Software Validation and Verification Plan.

System Model The inputs to a Software Program representing the physical properties of a real system. A System Model comprises the spatial and/or temporal model of the system to be analysed and the associated sets of input data.

Test Calculations:

'blind' Calculation carried out after an experiment or test being done without knowledge of the experiment or test results. Starting and boundary conditions of the actual experiment or test are employed.

'pre-test' Calculation carried out prior to an experiment or test. Starting and boundary conditions should be assumed appropriately.

V&V Verification and Validation.

Verification is the process of ensuring that the controlling physical equations have been correctly translated into the software.

Validation is defined as the evidence that demonstrates that the software is fit for its purpose. When calculating physical processes it may mean showing that the calculation is bounding with a suitable degree of confidence rather than a best estimate.

3 Scope

Considering the potentially high risks that might be associated with the PBMR, it is evident that computer based safety analyses require a thorough verification and validation (V&V) process. This comprises all aspects of the calculation not merely the software program. Thus the scope of the description of V&V in this document relates to the entire Evaluation Model.

For the purposes of this document all calculations and analysis being performed using relevant Evaluation Models will be termed ‘safety analysis’.

As stated in /1/ the procedures described in this guidance are also applicable to hand calculations.

This guidance is not directly applicable for software used directly for plant operational control and protection. International guidance for this type of software is given for example in /6/ and /11/.

The document continues with recommendations concerning Quality Assurance and Configuration Management.

A comprehensive V&V process for the computer based safety analyses of the PBMR starts with an identification and evaluation of the physical and chemical phenomena and/or risk and probability attributes governing the behaviour of the PBMR. Respective recommendations are given in the subsequent section of this document.

In the next step appropriate mathematical formulations have to be established for the identified physical/chemical phenomena and for combinations of them which might occur simultaneously.

The selection of appropriate software programs and their classification with regard to their importance for the Safety Case and to their V&V status forms the following step as part of the creation of the overall Evaluation Model.

Once a comprehensive list of software to be used in the safety analysis is established, the individual software applications have to undergo their individual V&V procedures. These can be divided into:

- V&V of software programs themselves
- Determination of the level of conservatism in the software and analyses
- V&V of the system models used in the calculations
- V&V of the input data used in the calculations
- Verification of the qualification/experience of the software users. (This point is discussed in sufficient detail in RD-0016 and therefore no extra guidance is presented in this document).

The NNR will not provide a general approval for specific computer software irrespective of its intended use, but will only state its acceptance for specific applications in the safety analysis under

specific conditions. For specific applications also an independent assessment involving separate calculation models and software programs may be required.

4 Quality Assurance and Configuration Management

4.1 Quality Assurance

In general Quality Assurance has to follow the requirements established in /10/.

As required in /1/ the V&V activities detailed below must be carried out by competent staff who are sufficiently independent of the software developers.

Independent staff may be an independent team / department of the users organisation or an external organisation. In this guidance both of them will be subsumed under the term 'V&V team'.

Involvement of the software developers themselves in the V&V activities is permissible but it is recommended that :

- The V&V plan and results of V&V efforts are reviewed independently.
- Checking of calculation/analyses should be performed by competent independent staff. (It is recommended that the bulk of V&V analysis is performed by independent staff since this also re-enforces the desire for comprehensive software documentation).
- The Software configuration management is independently checked.
- The V&V work management is not the responsibility of any of the developers.

Where a multi-level software classification scheme is applied the amount and type of V&V should be justified with respect to the importance classification of the program. For QA-purposes the division of responsibilities between software developers/users and the V&V-team should be specified in the V&V Plan.

4.2 Configuration Management

As required in /1/ all activities relating to software products must be performed under a defined configuration management. Main elements of a software configuration management are:

- It should be ensured that a software system and its components, specifications, verification evidence and descriptive documentation are all mutually identified and at a known status (defined by issue/revision/version identifiers) and are contained in a suitable database.
- A software configuration control system should be established with a datum defined as a configuration baseline from which all amendments and/or changes should be controlled.
- The physical security of source codes should be assured in order to prevent unauthorised changes.

- Software configuration should be checked against hardware configuration to ensure mutual compatibility.

- Controls should be established to record changes of the configuration status.

- The configuration system should only permit sanctioned program versions to be used.

- Provisions should be made for informing all personnel of the latest changes to the software and/or documentation.

- A master configuration list should be established containing all configured items together with their current configuration status and identification of all associated documentation. In order to limit the scope for inadvertent changes that list should be prepared automatically from the database.

Further guidance is also provided in /17/.

5 Identification of Governing Physical and Chemical Phenomena

The steady states as well as the transient behaviour of the PBMR are governed by a variety of physical and chemical phenomena. Examples for such phenomena are listed in Appendix 1.

In many cases the behaviour of the PBMR is characterised by an interaction of several phenomena which cannot be easily separated. In the following the term 'physical process' is defined as any inseparable combination of physical or chemical phenomena which describes a particular sector of the totality of processes or a particular time range in the behaviour of a nuclear facility.

Given a basic design for the PBMR, the establishment of the Safety Case requires a comprehensive evaluation of its planned operational behaviour and of its behaviour following any Postulated Initiating Event (PIE). It is therefore very important that the physical processes which govern this behaviour are structured and analysed properly. The methods to be applied for such analyses should be submitted to a thorough verification and validation process. The results of these analyses could potentially impact the design of the PBMR and govern an iteration process in the design.

As a first step those physical processes should be identified, which are relevant for the establishment of the Safety Case. A comprehensive list of these physical processes should be produced and reviewed against international practise and experiences. This is the basis for the selection of the methods of analysis (mathematical formulations) and the subsequent selection and classification of software.

6 Evaluation Model, Mathematical Formulations

6.1 Selection of Appropriate Mathematical Formulations

For each of the listed physical processes an appropriate mathematical formulation will need to be selected. This selection should regard the required dimensionality, any time dependency, the degree of interaction between individual phenomena and the availability and sufficiency of analytical solutions. Mathematical formulations in this guidance also include those used in correlations whether based on physical models or derived from test and experimental data.

Note: In addition, mathematical formulations will be selected for those elements of safety analysis which are not based on the modelling of physical/chemical phenomena, for example methods to perform probabilistic risk analyses or fault tree analyses.

The limits of application of the selected mathematical formulation should be clarified.

The physical phenomena, which the mathematical formulation shall model, should be defined and any changes in the physical phenomena, which render the formulation to be inapplicable, should be identified and documented.

Modelling a physical situation will involve the development of mathematical equations to describe the processes that are believed to occur. In general, idealisations and simplifications are made to enable a tractable mathematical formulation to be established.

In order to allow a validation of the selected method of analysis, the derivation of the equations should be clearly stated along with the justification for any simplifying assumptions. Guidance for the validation is given in Section 8.6. The principles stated there for validation of software products apply analogously to mathematical formulations used in hand calculations and in correlations based on physical models or empirical data.

The design of the PBMR is in line with several predecessors for which physical processes have already been identified and mathematical formulations have been defined and verified in the past. These mathematical formulations should typically be preferred. They have, however to be compared with the state of the art of comparable applications. Where this comparison leads to more than one available option, the options should be discussed and the selected option justified in the relevant documentation.

6.2 Level of Conservatism

Computer based calculations will typically be performed as conservative (biased) calculations or as best estimate calculations. However, some safety analysis calculations may use a mixture of these approaches for different parameters depending upon the particular requirements of the analysis problem involved.

Evaluation Model V&V sensitivity analysis can be used to provide information as part of an Uncertainty Analysis to determine best estimate plus uncertainty results to the required confidence level.

Conservatism of a particular calculation may result from conservatism built into the mathematical formulation of the software or into the calculation model, or the input data may have been biased. In every case the degree of conservatism shall be clearly stated and possibly quantified and it should be recalled that conservatism of a particular aspect in a particular event sequence may lead to optimistic results in other event sequences.

Example: For licensing of the PBMR the risk for the public and the workers has to be analysed (PRA). If confidence on input data is limited, the level of conservatism of the analyses has to be increased and justified even though the grade of V&V of the respective software may be sufficient.

7 Evaluation Model, choice and classification of software programs

7.1 Selection of Appropriate Computer Codes

For many cases no analytical solutions exist for the selected mathematical formulations, which can be treated by manual calculations, especially if the steady state or event sequence to be analysed is governed by several interacting phenomena. Therefore appropriate programs need to be selected which are able to properly analyse the relevant interacting phenomena. The selection of the software programs should be clearly stated and justified.

Relevant criteria for the selection are:

- The mathematical formulation of the software is suitable for analysis of the envisaged physical process.
- The software is well documented and has a complete development QA trail.
- A documented V&V-process has been performed on the software.
- The software is well established for applications equal or similar to its intended application.
- Qualified/experienced users are available or can be trained to a satisfactory level of expertise on the required timescale.

New designs usually refer to previous designs developed and partly operated in the past. For software used for previous designs and intended to be used also for a new design it is probable that they have been validated to a certain extent. Nevertheless they should be approved concerning the following aspects:

- Development and documentation (e.g. V&V plan)
- V&V measures and results
- State of the art concerning current requirements
- User requirements regarding ‘up to date’ input/output processing

7.1.1 Computer Codes for Separate Analysis of Physical Phenomena

It may turn out that there is no appropriate software available for the analysis of certain identified physical processes. This may happen especially in stage 1 of the licensing process. The reason that no software exists at all for a particular physical process may be because it is unique to the envisaged design of the reactor, that the V&V status of existing software is too poor to inspire confidence in the calculational results and cannot be improved with a reasonable amount of effort and time, or that development of new software is not possible for some reason.

In this situation the physical/chemical phenomena constituting the identified physical process may need to be analysed separately with software appropriate for these tasks. The separate analysis of actually interacting phenomena should, however, be accompanied by a thorough discussion of the interfaces and the higher safety margins or conservatisms needing to be applied in the safety analysis performed with the software. Thus part of the effort saved in the V&V of a complex program may be lost to covering of the interface problem. See also Figure 5.

7.2 Classification and Collation of Selected Software Programs

The selected software programs should ideally be classified with regard to their importance for the Safety Case as well as with regard to the degree to which they are already validated.

Such a classification would allow for an identification of those programs which reveal a deficiency between their importance for the Safety Case and their given documented and approved V&V status. However this will necessarily be a coarse grading process. Overall it is more important to identify areas where the analysis capabilities are deficient at the relevant level of detail to allow these deficiencies to be rectified. Note that such deficiencies should be addressed promptly so that any interim analysis results are based upon the best available models at any given time.

An example classification scheme is discussed in Appendix 5.

7.3 Collation of Information for Selected Software Programs

Once the software to be applied for safety analyses has been selected, a list should be produced summarising the complete set of selected software with references to individual code documents. As a minimum the following software attributes should be included in this list:

- Function of the program
- Version
- Language
- Operating system
- Applied hardware configurations

- Supplier/origin
- Availability of source code
- Number of qualified/experienced users
- V&V status
- Method of configuration control
- Identification/qualification of program controller(s) responsible for configuration control
- Classification of the program (if a classification scheme is in use).

8 Evaluation Model, Software program V&V

8.1 Introduction

The scope of the measures for V&V depends on the level of confidence already established for the respective software. It may be based on the following items:

- Status of program documentation
- Current level of documented V&V status
- Past experience
- Comparability of application
- Difference between the importance and established V&V status

Detailed guidance for V&V of software is established in various publications, rules and standards (see section 10).

The level and type of V&V activities depend on the type of software to be validated and where applicable its classification (see also section 7.2). It is important to note that the V&V activities should cover all of the EM safety analysis calculation applications.

The type of the software may vary from programs which are still to be developed, through existing software programs of various ages with previous application in specialised areas, to widespread, internationally used commercially developed software for general applications. These types of software shall be accordingly called New Software, Legacy PBMR Software, and Commercial Off The Shelf (COTS) Software in this guidance.

General guidance on software V&V is given in the initial sub-sections below. Additional guidance specifically for the development of New Software, Legacy Software and for COTS Software is given in sections 8.7 to 8.9 respectively.

8.2 Clarification of V&V Status

For existing software the first step involves an identification and assessment of the available documents linked to software development such as:

- A Program Documentation based on the guidance listed in Appendix 3
- Functional Requirement Specifications
- Design Specification
- Test Plan and Test Results

If formal documents on development and V&V are not available, they should be reconstructed on the basis of available information.

Based on the assessment results and the requirements of current application, complementary measures should be defined to bring the V&V up to the appropriate standards.

8.3 V&V Plan and Documentation

According to /1/ an adequate documentation of the V&V efforts is required. V&V Plans, interim and final V&V Reports are identified as elements of the appropriate comprehensive documentation.

Having clarified the status of the software to be validated, a Software V&V Plan (SVVP) should be produced addressing the following items:

- Identification and classification of the software to undergo V&V
- Reasons why retrospective V&V is to be performed
- Scope and objectives for the level of V&V selected
- Existing documents on software development to be used for review
- Availability and use of user experience
- Actions to be taken to supplement missing or unavailable development products
    - Activities to be performed
    - Tools and techniques to be used

  - Documentation to be produced
- Time schedule of V&V activities and milestones
  - Receipt of development products
  - Performance of V&V activities
  - Delivery of V&V products
- V&V project organisation
  - Staffing levels for each activity
  - Responsibilities
- V&V project management
  - Supervision and coordination of V&V activities and deliverables
  - Project controls
  - Security measures

For documentation of the results the following information should be included in V&V Reports:

- Summary of V&V performed
  - Development materials evaluated
  - Tools and techniques used
  - V&V documents generated
  - Security and control procedures followed
  - Deviations from the V&V plan
- V&V results
  - Summary of positive and negative findings
  - Summary of discrepancies and deficiencies detected
  - Resolution of discrepancies and deficiencies
  - Conclusions concerning the materials evaluated

- Recommendations and Requirements for additional efforts

Any recommendations and requirements for additional efforts should immediately be fed back into the SVVP by issuing a revised SVVP version.

For QA-purposes the results of the V&V process should be reviewed by the V&V-team and the review should be documented in a Final V&V Report. The review documentation should include the following issues:

- Completeness of the V&V Plan, especially of the identified or reconstructed program requirements
- Verification efforts: Test coverage and evaluation of test results
- Validation efforts (comparison to analytical solutions, benchmarks, other software, experiments, or plant data) and evaluation of validation results

As required in /1/ the V&V-Plans (prior to execution and if necessary, also revised versions) and the Final V&V-Reports must be submitted to the NNR. The review process is likely to be assisted by making these available as they become available. For clarification of details NNR can request additional documents for background information.

8.4 Scope of V&V Measures

This sub-section provides guidance for the scope of V&V measures that should be considered in the V&V plans and in the V&V reports accordingly. It should be noted that for most of the aspects listed below requirements or recommendations are already defined in /1/. They are detailed here in order to provide a complete picture of the scope of V&V measures. Recommendations provided here are overruled by the applicable requirements defined in /1/.

The scope of V&V measures should cover the following aspects in particular:

- The description of the considered physical/chemical phenomena should be reviewed and their range of applicability evaluated.
- The adequacy of the mathematical model for the class of problems to be analysed and any justification of idealisations and simplifications should be evaluated.
- Any claim to conservatism of the mathematical model should be justified and possibly be quantified in order to allow for a meaningful analysis of uncertainties of a given calculation. It has to be kept in mind that, in complex calculations, conservatism in the modelling of a single phenomenon might well lead to misleading sequences of events, unrealistic transient time-scales being predicted and some physical phenomena being missed.
- The adequacy of the solution algorithms and numerical models for the class of problems to be analysed should be evaluated with special attention to convergence and stability of solution.

- If empirical correlations are used, the methods used to prevent or notify use of the correlation outside its definition range should be described. For complex correlations flow charts should be provided showing their implementation in the calculation. Where the calculation may switch between different correlation ranges, discontinuities may be present. Any modifications to overcome such computational difficulties should be described.

- Sensitivity studies should be performed with the software in order to clarify important parameters and more generally to seek evidence about any cliff edge effects. Validation efforts should be intensified whenever the software predicts such effects for an analysed phenomenon in a range that might be covered by the accident behaviour of the PBMR. If identified cliff edges remain in the software being used in the safety analysis then justification for this approach should be presented in the relevant software documents made available to the NNR

  Sensitivity analysis can also be used to provide information as part of an Uncertainty Analysis to determine best estimate plus uncertainty results to the required confidence level.

- The source code should be programmed according to good programming practises such as listed in Appendix 2 in order to ease understanding of the code and to allow its portability to other operating systems and compilers as well as to other hardware environments. Appendix 2 also contains guidance for software using Graphic User Interfaces (GUI) and for Object Oriented Code development.

- The Program Documentation should be verified to ensure that it has been completed, conforms to established standards, and is a clear and correct description of the completed computer program. Respective conditions are given in Appendix 3.

- The software should be designed in a way to allow effective user control of input processing. It should be furnished with suitable measures to trap input data errors.

- The software should check basic conservation laws globally as well as locally and issue warnings should violations occur. Relevant output data should be processed to easily understandable formats, additional output should be available to facilitate review and plausibility checking. It is important that the program output should include run time/date and particularly code version information.

- It should be assured that the software is compatible with the hardware and the operating system (and version) of the computer installation on which it shall be run. The algorithms used should maintain the required numerical precision and should preferably be free of numerical instabilities. If any known numerical instabilities are present the approach taken should be justified.

8.5 Verification - general methodology

In order to verify whether the software is programmed in a way to satisfy the specified requirements, program tests should be performed.

The verification activities for software can be divided into two main items: into the verification of the selected numerical algorithm addressed by the first bullet points of section 8.4 and into the verification of the coding itself, i.e. proof that it is to a reasonably achievable degree free from programming errors.

In the following a wide spectrum of methods is addressed which can be (and should be) applied during the development of New Software and can also be considered for improvements in verification of Legacy Software or for further development of Legacy Software. More specific aspects linked to legacy or COTS software are addressed in the respective subsections below.

Software testing methods can primarily be divided in static analysis and dynamic testing methods where the former analyse the form, structure , and consistency of the program without executing it while the latter involve execution of the program.

8.5.1 Numerical Algorithm Verification

Types of Algorithm Testing (see /13/ for details and continuative references):

- Analytic Solutions

  These are closed-form solutions to special cases of the PDEs that are represented in the mathematical formulation of the software. They may be represented by infinite series, complex integrals, and asymptotic expansions which themselves need numerical methods to be evaluated. Nevertheless they are usable for comparison with the results of the software, since the accuracy of these solutions can usually be quantified much more rigorously. Their most significant shortcoming is, however, that they exist only for very simplified physics and geometries.

- Method of manufactured solutions (MMS)

  This is a method of designing verification test problems of wide applicability, where a specific form of the solution function is assumed to satisfy the PDE of interest. This function is inserted into the PDE, and all derivatives are analytically derived.

- ODE benchmark solutions

  These are semi-analytic solutions which are achieved by reduction to numerical integration of ODEs. They are thus accurate solutions to special cases of the general PDEs resulting from simplifying assumptions such as simplified geometries, constant material data, etc.

- PDE benchmark solutions

  These are highly accurate numerical solutions to special cases of the PDEs or of the boundary conditions.

  Published solutions (both ODE or PDE solutions) should only be considered a numerical benchmark solution if:

- The software used in producing the solution has been thoroughly verified and documented.
- A comprehensive numerical error estimation is reported with the solution.
- The solution has been accurately calculated by independent investigators, preferably using different numerical approaches and software.

- Conservation tests

  These are global or regional tests for the conservation of mass, momentum, and energy.

- Alternate coordinate system tests

  These are tests to determine whether the same numerical solution is obtained in different coordinate systems.

- Symmetry tests

  These are tests to evaluate whether certain symmetry features are preserved in the solutions, i.e. that the same solutions are obtained when symmetry conditions are imposed and when actually symmetric domains are solved as a whole. Also tests considering unbounded domains to check whether infinity boundary conditions are applied correctly can be subsumed here.

- Iterative convergence tests

  These are tests to check the convergence behaviour of iterative methods.

8.5.2 Software testing - Static analysis Methods

Useful static analysis methods presented in /15/ include:

- Control flow analysis to find errors such as unreachable code, endless loops, violations of recursion conventions, and loops with multiple entry and exit points

- Data-use analysis to find errors such as data used before initialisation, variables declared but not used, and redundant writes

- Range-bound analysis to find errors such as array indices outside the boundaries of the array

- Interface analysis to find errors such as mismatches in argument lists between called modules and calling modules

- Information flow analysis to check the dependency relations between input and output

- Verification of conformance to language standards (e.g. ISO C)

- Verification of conformance to project coding standards, such as departures from naming conventions

- Code volume analysis, such as counts of the numbers of modules and the number of lines of code in each module

- Complexity analysis, such as measurements of cyclomatic complexity and integration complexity

Tools that support one or more of these static analysis activities are available and their use is strongly recommended. Compilers often perform some control flow and data-use analysis.

Some compile/link systems automatically do interface analysis to enforce the strong type checking demanded by the language standards. Compilers for some languages, such as Fortran and C, typically do not do any interface analysis. Static analysis tools are particularly useful when compilers do not check control flow, data flow, range bounds and interfaces. The use of a static analyser may be regarded as an important step in C program development, for example.

Reverse engineering is another method of static analysis. Tools are available to generate an as-built structure chart of the analysed program which may then be compared with the as-designed structure chart in order to verify that ‘as-built’ conforms to ‘as-designed’. This should be performed especially to verify existing software, but can also be useful in the development of new software.

Manual techniques such as design walk-through/review and source code line by line review should be used during all applicable phases of the software development. For the verification of existing software, line by line review of the source code is a very important option to increase confidence in cases when the existing documentation of software verification is judged to be insufficient.

8.5.3 Software testing - Dynamic Testing Methods

Dynamic testing methods include:

- Black box testing

  Black box testing of code involves the creation of tests with out any knowledge of the internals of the coding. This can be used to check for deficiencies in interface coding and documentation. Black box testing can be used with quasi random input data to identify deficiencies in the data handling of code compared to the functional requirements specification.

- White box (glass box) testing

  White box testing of code involves the creation of tests to check the behaviour of specific parts of the code with the knowledge of how the internals are designed and coded. This can be used to test operation of specific state transitions, flow control etc.

  Debuggers are used in white box testing for controlling the execution of the software. They are often able to:

- Display and interact with the tester using the symbols employed in the source code
- Step through the code line by line
- Set watch points on variables
- Set break points
- Maintain screen displays of the source code during execution
- Log the session for inclusion in the test results

- Coverage analysis and code sensitivity analysis

  These are methods to assess the complexity of the software in order to help design collections of suitable test problems. Coverage analysis enables the determination of which of various options are actually exercised in the software. The relative importance of the selected component contributions to that calculation is then determined via a sensitivity analysis.

- Regression testing

  This is testing which checks that new code versions give the same results, or results with expected changes, when compared to results for the same test applied to an earlier code version.

Ideally initial code version testing will involve a mixture of white and black box testing. White box testing will normally be used together with regression testing for development of new code versions from existing code versions over time (where such development is required). Black box testing can also continue to be applied over successive code versions particularly if major new models are developed.

Software testing strategies are described in detail in /14/.

8.6 Validation - general guidance

Software programs that model physical processes should be validated in order to prove that they predict these processes correctly. There are several options for performing this validation and the combinations of options yielding the desired level of validation should be chosen and these documented and justified in the V&V documentation. The particular options are:

- Comparison of software results with analytical solutions:

  Depending on the nature of the phenomena to be analysed, analytical solutions for simplified cases or asymptotic solutions for limiting cases may exist and can be analysed with the software. In most cases, however, this will be rather a verification of the mathematical formulation of the software models rather than a validation effort.

- Comparison of software results with numerical benchmark problems:

  In some areas, as for instance structural mechanics or neutron physics, numerical benchmark problems have a long tradition. Again their use will provide information on the

correctness of the mathematical solution rather than on the adequacy of the physical/ chemical modelling and will therefore be rather a verification effort than a validation effort. Nonetheless it is important to ensure that numerical solution errors are small compared to modelling errors and benchmark problems may be a way of establishing bounds on these errors, albeit for limited types of problems.

- Comparison of software results with the results of other programs:

  This may provide useful information on the validity of the software. This applies particularly in those areas of severe accident modelling which cannot be covered sufficiently by experiments or commissioning tests. The following prerequisites should, however, be considered:

  - The comparison program should have been developed independently.
  - The programs should use different methods, wherever possible. Where common methods are employed for major models this V&V limitation should be discussed in the V&V documentation.
  - The program should have undergone an independent and traceable V&V for a code version and platform which is the same or closely linked to the version to be used for the comparison calculations.

- Comparison of software results with experiments:

  For acceptable validation of a program it should be able to satisfactorily predict, or at least to adequately reproduce, the results of experiments conducted on those physical processes which the software is designed to analyse. Validation should be achieved by use of both ‘separate effects’ experiments and ‘integral’ experiments.

  ‘Separate effects’ experiments are designed to examine, at the most, a few phenomena which the calculation is attempting to model, whereas ‘integral’ experiments are designed to examine preferably the complete physical processes governing the behaviour of a nuclear installation.

  For both types of experiments the following basic requirements should be satisfied:

  - The experiments should be well instrumented.
  - They should be designed at a scale close to the conditions of the PBMR or otherwise be conducted at different scales suitable to allow for a meaningful extrapolation.
  - The respective calculations should preferably be ‘pre-test’ or ‘blind’ calculations.
  - If adjustments are made in either numerical method, input parameters or calculation model in order to obtain sufficient agreement of experiment and calculation, the

general applicability of these adjustments, especially for the intended design calculations should be discussed.

- Comparison of program results with plant data, including tests carried out during commissioning or start-up and operational occurrences or accidents, is another important tool for the validation of the program.

  However, the advantage of analysing a full sized set up is often reduced by the fact that plants are usually not as well instrumented as specially designed experiments and also that, at most, only very limited data may exist on severe transients which would be important for the Safety Case.

  Since in the design phase (especially of a demonstration plant) no plant data can exist for a specific plant, data for existing similar plants may need to be utilised.

In cases where a satisfactory validation is not achieved or not achieved in time for particular licensing stages, additional margins have to be introduced into the respective safety relevant calculations. These margins should be justified dependent on the degree of validation achieved.

An example of the validation efforts for an established thermal-hydraulic code is given in Appendix 4.

8.7 Additional V&V aspects relevant to the Development of New Software

For software which is still to be developed, it is advantageous to perform verification (in particular) and validation (if possible) of software components as they are created throughout the overall development process.

In /4/ software development and associated V&V is divided into the following phases:

- In the Initiation Phase, a problem requiring solution is identified and a decision is made to develop software to address the problem. V&V planning should be initiated in this phase including:
  - Assignment of responsibilities between software developers and V&V teams.
  - Level of effort to be applied on V&V (as compared to level of software development effort).
  - General V&V activities to be performed and the level of effort to be applied for each activity
  - V&V documents to be provided.
  - Procedures to be followed for incorporating V&V findings back into the subsequent software development process.

- In the (functional) Requirements Definition Phase, the requirements that the software must satisfy should be identified and documented in a Requirements Specification. In addition, a V&V Plan and preliminary Test Plans, both for formal testing (verification) of the code and for code validation by recalculation of physical experiments or by inter code comparison, should be produced during this phase.

- In the Design Phase the logical structure, information flow, logical processing steps, data structures, etc. should all be defined and documented in a Software Design Specification. A Program Documentation process analogous to the approach given in Appendix 3 should be established. The V&V activities to be performed during this phase are design verification, verification of the preliminary Program Documentation, and update and verification of the Test Plans.

- In the Coding Phase the software design is implemented by coding it in a suitable programming language. The coding process should be accompanied by static checks such as compilation and by preliminary checking of the software components created (i.e. unit/module testing). Test Plans for code validation should be finalised at this stage and the test database of physical experiments should be verified.

- During the Integration and Testing Phase, the software components are combined to create the overall program. Testing should be performed to ensure correct integration of the components, and the integrated software should undergo formal testing. Besides these verification efforts the validation test plans should be executed and the results of the validation test calculations documented and reviewed.

- During the Installation Phase, the integrated software is installed in its operational environment. The acceptability of the software installation should be verified. At the same time the Program Documentation should be finalised and the Final V&V Report prepared.

Checklists and more details for the different stages of software development are presented in /4/.

8.8 Additional V&V aspects for Legacy Software

For the specific situation of PBMR, in many cases some V&V will have to be performed on existing software which eventually will need enhancements to serve their intended purpose. Verification and validation will generally follow the general guidance discussed in sub-sections 8.5 and 8.6 but some special factors need to be considered:

- Older software may include unstructured coding which is difficult to verify. This situation may be worsened by the remains of old code structures to limit memory use or speed up the code or resolve other problems prevalent in the past. Considerable care should be taken when altering such software.

- Compared to widespread COTS software the number of other users is normally low for legacy software. The benefit of frequent usage, some kind of implicit quality assurance is thus lower. Therefore legacy software without access to at least the significant parts of their source code allowing comprehensive independent verification of the actually used paths and models should not be used for PBMR safety analysis.

The scope of V&V efforts for Legacy Software should be based on an assessment of already existing V&V documentation. The following sub-sections detail recommendations for such an assessment.

8.8.1 Verification

A summary of all program tests and test results carried out by the developers or by users should be compiled, which are available for use in the verification process. The summary should be reviewed to determine the adequacy of test coverage. The test coverage may be considered adequate if:

- A sufficient number of tests has been carried out to test all program requirements
- Tests representative of anticipated program applications are included
- Important design features and major logical paths are tested
- The results of the tests are satisfactory

Depending on the specifications in the V&V plan, the V&V team may decide to repeat some of the developer and user tests, and to conduct additional ones if the test coverage is found to be inadequate. These decisions should be justified and documented in the V&V documentation.

The test results should be reviewed to determine that all program requirements have been tested and that no significant discrepancies exist between results obtained by the V&V team, the developers or the users.

8.8.2 Validation

The V&V team should also compile a summary of all available validation calculations which have been performed by the developers and users and their results. The summary should be reviewed to determine whether it covers adequately the scope and range of application of the physical processes to be considered. The results of the available validation calculations should be evaluated with respect to their significance and the degree of agreement between calculation and experiment, benchmark or analytical solution.

Depending on the results of the review/evaluation of existing validation efforts and depending on the specifications in the V&V Plan, additional validation efforts may be required and should then be specified accordingly.

8.9 Additional V&V aspects for Commercial Off The Shelf (COTS) Software

V&V efforts for COTS software should be similar to those for legacy software. In both cases the software exists and some related documentation should also be available. As with legacy software, V&V activities for COTS software should start with a clarification of the software V&V status. The results of this clarification and emerging requirements for additional V&V efforts should be incorporated in the V&V Plan and subsequent V&V documentation.

If the existing Program Documentation including documentation of software verification and of validation test calculations is judged to be sufficient with regard to the intended application of the software, V&V may focus mainly on applicability of the system model, input, and user qualification.

8.9.1 Verification

Typically no source code is available for COTS software. In this situation verification activities independent of the software supplier are restricted to black box testing and those types of algorithm testing which do not depend on source code access. As the capability for independent software verification is typically limited, an assessment of the documented QA measures undertaken by the software supplier is appropriate.

Depending on the verification documentation provided by the software supplier, the V&V team should decide and justify in the V&V documentation which parts of the verification testing should be performed or repeated independently. Subject to their importance for the Safety Case, COTS Software without sufficient confidence in their verification process should be excluded from PBMR safety analyses.

8.9.2 Validation

The review of existing validation documentation should lead to the identification of those items which are not adequately covered by the existing validation calculations and which therefore require additional validation test calculations.

8.10 V&V for Particular Features of Complex Computer Codes

In the past, complex software have been developed covering a wide range of possible applications. In a particular application only parts of the software capabilities might be used, e.g. PMBR fluid flow calculations will need no two-phase flow models. In this situation some deviations from a full V&V process may be allowable:

- If the software is not fully verified, a verification must be performed for at least those parts of the software which are used for the actual applications. In addition, it should be proven in this case, that the parts of the software which are not verified, cannot influence the results of the validation or of the safety analyses.

- Documented validation should be supplied for those capabilities of the software which are actually used.

9 Evaluation Model, V&V of System Models/Data

9.1 V&V of System Models

The following sub-sections describes V&V of System Models. The desired level of conservatism is significant in this context and this is discussed in respect of all aspects of Evaluation Model conservatism in sub-section 6.2.

Some of the aspects discussed below are also addressed in requirements or recommendations defined in /1/. They are included here to give a complete picture of the System Model V&V. Recommendations provided here are overruled by the applicable requirements defined in /1/.

9.1.1 General considerations

System models comprise those aspects of the EM other than the software programs and so primarily comprise the modelling inherent in the input supplied to the software and other calculational tools in the EM.

System models are typically spatial and/or temporal models of the plant or test facility systems or components to be analysed. V&V of the system models is basically a validation effort and should account for the following items:

- Any nodalisation and time-step structure, if applicable, this should be detailed enough to allow a convergent and numerically correct solution.
- The detailing should also be sufficient to provide meaningful results in regions of the model which are of particular interest.
- The dimensionality of the model should fit the nature of the physical processes to be analysed. If simplifications are necessary with regard to the complexity of the model, they should be justified in separate analyses.
- Initial conditions and boundary conditions imposed on the calculation should be assessed to demonstrate their suitability.

The results of system model V&V should either be appended to the SVVP of the respective software or be incorporated in the respective Design Calculation Reports. For more complex system models a separate V&V Report may also be produced.

9.1.2 V&V of Input Data and Output Post-Processing

The processing for the derivation of the input data should follow auditable validation procedures assuring that all data used have a clearly defined origin within the design documentation or else that their source can be identified and justified. Details and justification should also be given of any embedded data in the software. The applicability of the data to the cases to be analysed should be justified and their degree of accuracy and uncertainty bounds should be given to allow a meaningful analysis of the uncertainties of a given calculation.

Empirical correlations may be used in place of more detailed physical expositions. For each correlation, statements should be provided on:

- the claimed accuracy, quantified wherever possible
- the range over which the correlation applies
- graphs showing the fit of the correlation to the data base
- relevance of the database:
    - the validity of correlation derived from experiments in scaled-down facilities should be demonstrated
    - the database should be wide enough to ensure applicability to all anticipated plant conditions

To facilitate output review, the output should contain checks of basic conservation laws globally as well as locally. Warnings should be issued in case of any violations. Relevant output data should be processed to easily understandable formats. Additional output should be available allowing for review and plausibility checks.

When analyses are performed employing individual programs consecutively or together, special care must be taken to assure a correct and transparent transfer of data at the interfaces between the individual programs.

If safety analyses are performed by means of software using Graphical User Interfaces (GUI), the program(s) should automatically record the program version details, all input used to control the program's operation, all data used for the calculation and a copy of significant output in log file(s) to be kept for the QA trail. The required content items for this file are analogous to those for non-GUI programs. GUI programs which cannot record these details should not be used for safety analysis.

The results of input data and output processing V&V should be incorporated in the respective safety analysis reports. In case of program embedded data and correlations this information should be part of the software documentation.

10 References

/1/ Requirements for Licensing Submissions Involving Computer Software and Evaluation Models for Safety Calculations
National Nuclear Regulator, License Document RD-0016 Rev. 0

/2/ Portability of Scientific and Engineering Software
ANSI/ANS-10.2-2000

/3/ Documentation of Computer Software
ANSI/ANS-10.3-1995

/4/ Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry
ANSI/ANS-10.4-1987 (R1998)

/5/ Manual on Quality Assurance for Computer Software Related to the Safety of Nuclear Power Plants
IAEA Technical Reports Series No. 282, Vienna 1988

/6/ Software for Computer Based Systems Important to Safety in Nuclear Power Plants
IAEA Safety Standards Series No. NS-G-1.1, Vienna (2000)

/7/ Review of Probabilistic Safety Assessments by Regulatory Bodies
IAEA Safety Reports Series No.25, Vienna (2002)

/8/ Safety Assessment and Verification for Nuclear Power Plants
Safety Standards Series No. NS-G-1.2, IAEA, Vienna (2001)

/9/ ISO 9001:2000 Requirements for Quality Management Systems

/10/ Quality Management Requirements for the Pebble Bed Modular Reactor
National Nuclear Regulator, Licensing Document LD-1094, Rev. 3

/11/ USNRC, “Verification, Validation, Reviews, And Audits For Digital Computer Software Used in Safety Systems of Nuclear Power Plants”,
Regulatory Guide 1.168, September 1997

/12/ Licensing Requirements for the Pebble Bed Modular Reactor
National Nuclear Regulator, Licensing Guide LG-1037 Rev. 1

/13/ W Oberkampf, T Trucano, C Hirsch, ‘Verification, Validation, and Predictive Capability in Computational Engineering and Physics’
Invited Paper for Session B1 of the Foundations for Verification and Validation in the 21st Century Workshop, John Hopkins University/Applied Physics Laboratory, Laurel, MD October 22-23, 2002

/14/ E Kit, 'Software Testing in the Real World'.
Publishers: Addison-Wesley ACM Press, 1995.
ISBN 0-201-87756-2

/15/ ESA Board for Software Standardisation and Control (BSSC), 'Guide to software verification and validation'
ESA PSS-05-10 Issue 1 Revision 1, March 1995

/16/ IEEE Standard for Software User Documentation
IEEE Standard 1063-1987

/17/ IEEE Standard for Software Configuration Management Plan
IEEE Standard 828-1983

Figure 1 V&V Route Map

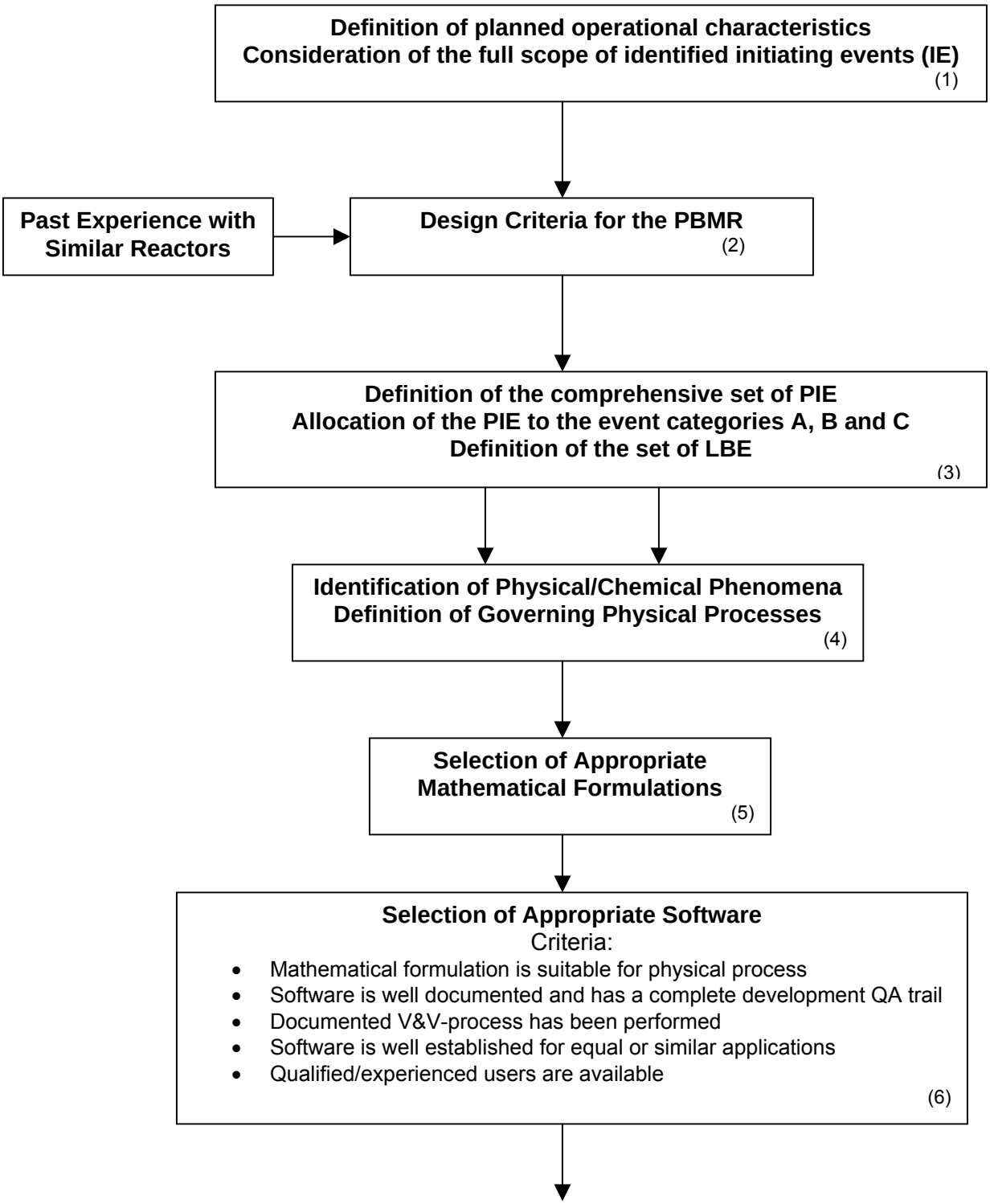


Figure 1 (continued)

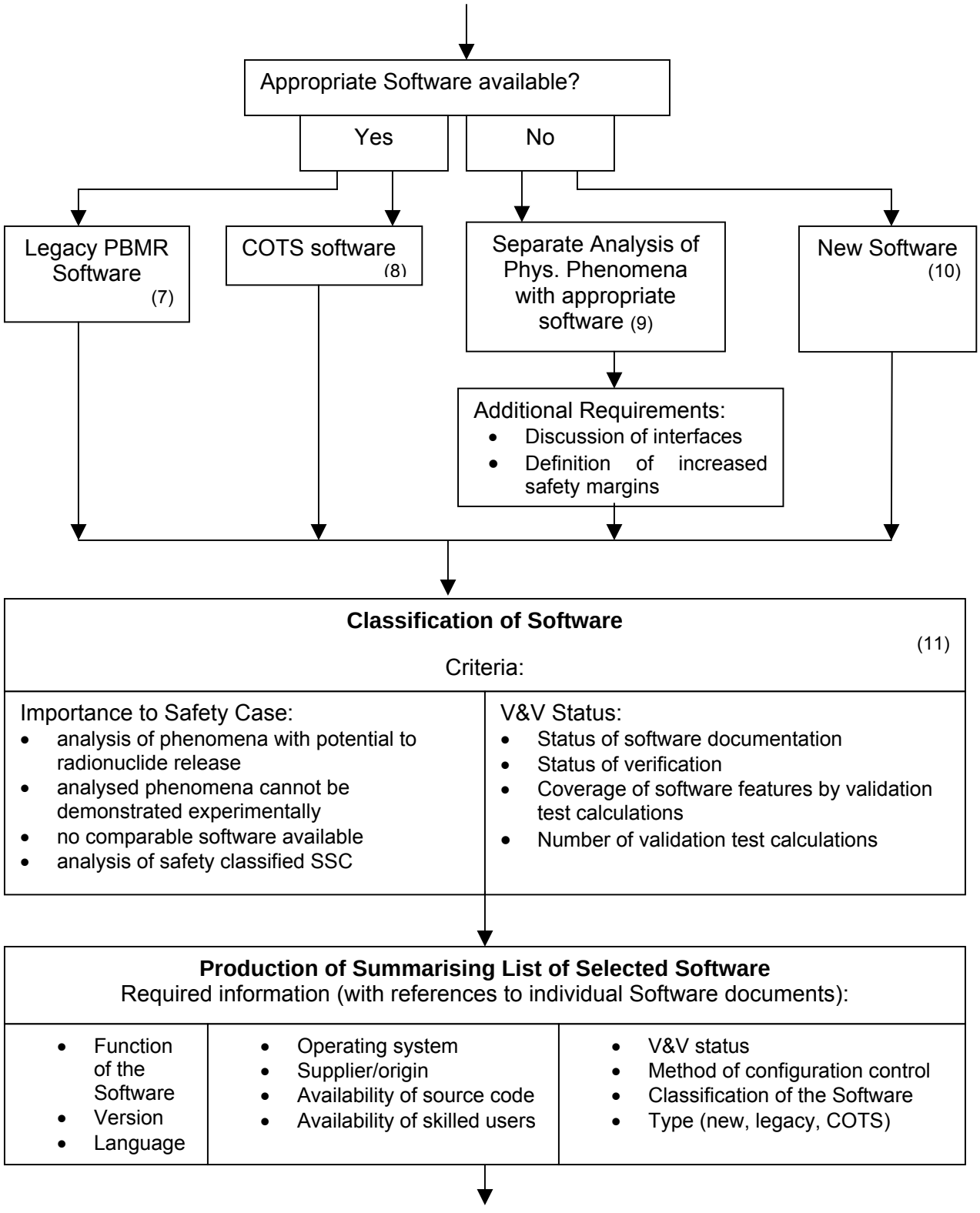
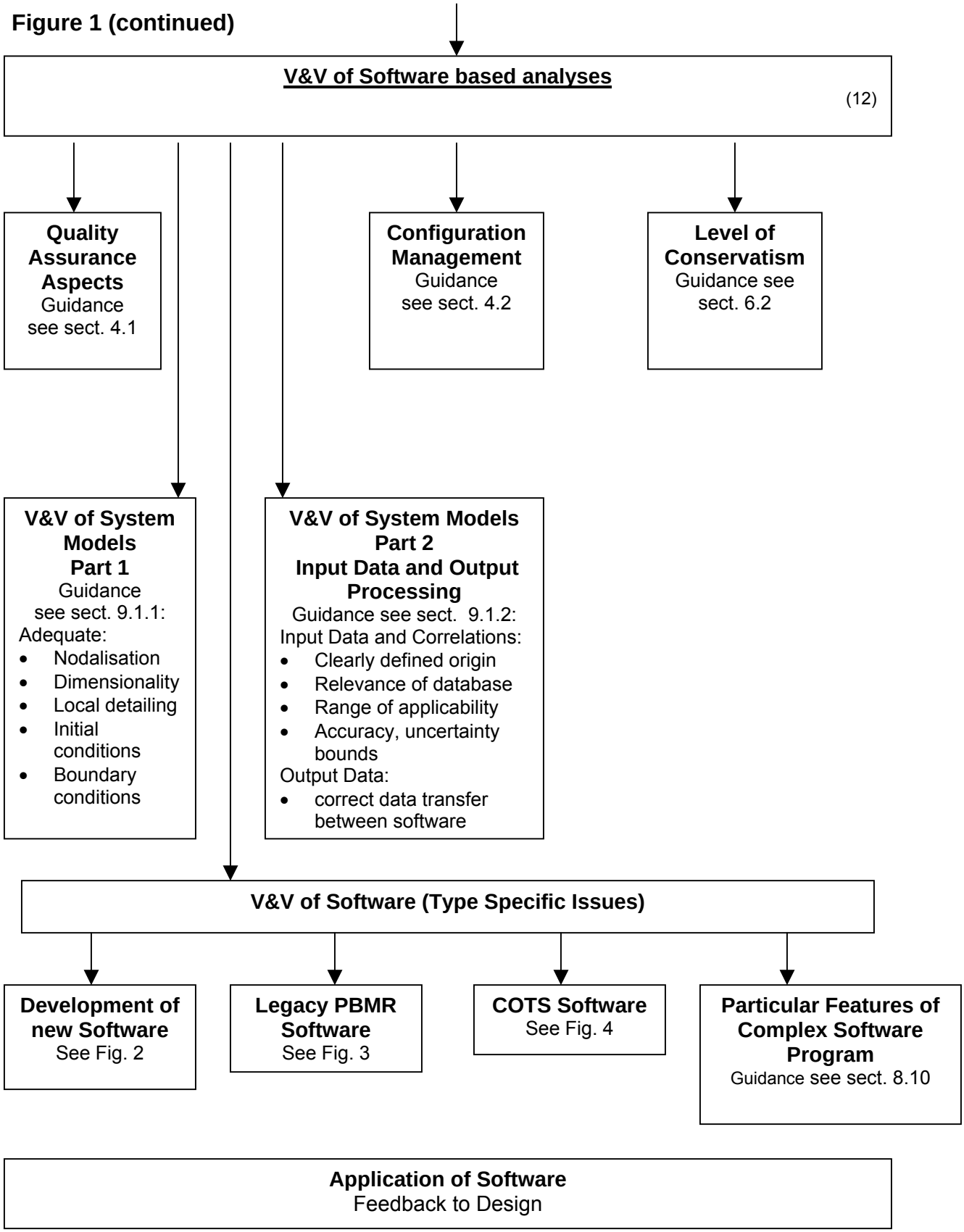


Figure 1 (continued)



Legend to Figure 1:

(1) To satisfy the requirements of LG-1037 /12/ for the plant intended to be constructed, the full scope of possible occurrences and events should be considered. This includes the definition of the planned operational characteristics and the comprehensive compilation of identified initiating events (IE).

(2) With the operational characteristics and IE to be covered and with experience from similar reactors and general regulatory requirements given, the Design Criteria for the plant should be defined.

(3) To proceed with the design process, a comprehensive list of Postulated Initiating Events (PIE) should be produced which covers all events to be considered according to LG-1037.

Based on justified frequencies, taking uncertainties into account, the PIE will be allocated to the event categories A, B, and C as defined in LG-1037.

In order to demonstrate compliance with LG-1037 regarding doses, risk, ALARA and DiD, a set of Licensing Basis Events (LBE) will be defined for subsequent detailed analysis, which will envelope all events from categories A, B and C.

See section 2 for definitions of IE, PIE, LBE and Category A, B, and C events.

(4) With the compilation of LBE given, the physical/chemical phenomena should be identified that can be observed in the courses of the event sequences. The governing Physical Processes, i.e. particular physical/chemical phenomena or inseparable combinations of interacting phenomena should be defined. See section 5 of this document.

(5) For the identified Physical Processes, appropriate mathematical formulations should be selected, see section 6.1.

(6) Appropriate software must be selected using as criteria the suitability of the underlying mathematical formulation, the status of the software documentation and V&V process, and whether the software is well established for applications similar to its intended application. See section 7.1.

(7) Legacy PBMR software is defined here as software which has been developed in the past for specialised applications used in the development and design of predecessors of the PBMR. Due to its history it is sometimes poorly documented and will often need substantial V&V efforts.

(8) COTS software are commonly used tools in other technical areas, e.g. fluid dynamics or structural mechanics. Usually they are well documented programs potentially with sufficient existing V&V for PBMR use.

(9) If appropriate software for the analysis of a particular physical process is not available, the underlying physical/chemical phenomena may be analysed separately with software appropriate for these specific tasks. This may involve the use of increased safety margins. The levels of such safety margins should be documented and justified.

(10) New Software is defined here as software which is being developed at present and which allows for a full scale V&V during all stages of its development.

(11) In order to structure the scope of V&V to be performed on the totality of software to be applied in the safety analysis, the particular programs can be classified with respect both to their importance to the Safety Case and to their individual V&V status. See section 7.2 and Appendix 5.

(12) V&V of software based analyses comprises V&V efforts for the software itself, for the calculation models and the input data used in the calculations, and for the qualification of the software users. For the software itself the efforts divide into verification, i.e. assessment that the mathematical formulation has been correctly translated into the software and that the software works correctly, and validation, i.e. assessment that the software describes the physical processes properly when applied to relevant physical experiments. See section 8 and Appendix 3 for details and for guidance on the content of the V&V documentation.

Figure 2 V&V for Development of New Software

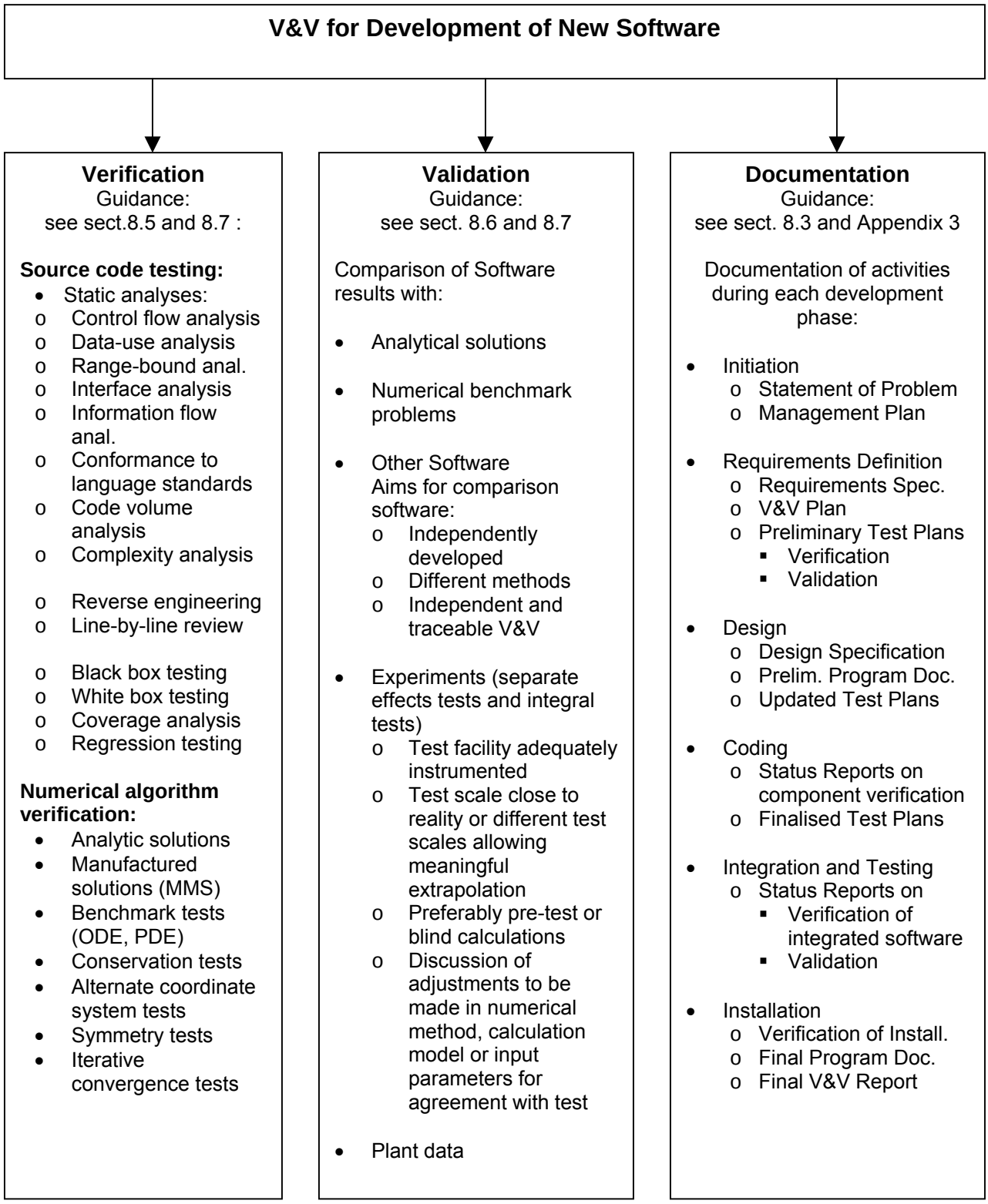


Figure 3 V&V for Legacy PBMR Software

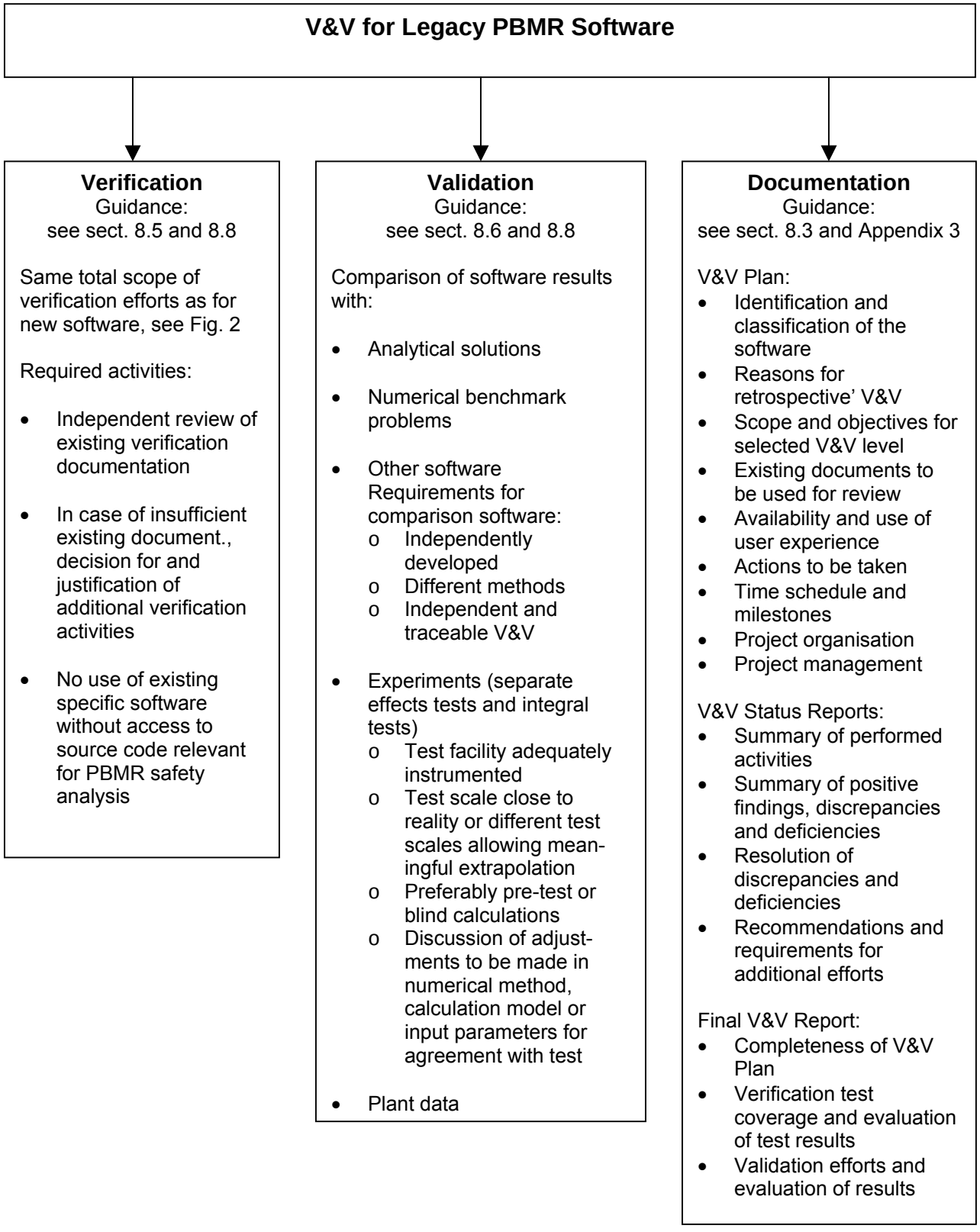


Figure 4 V&V for COTS Software

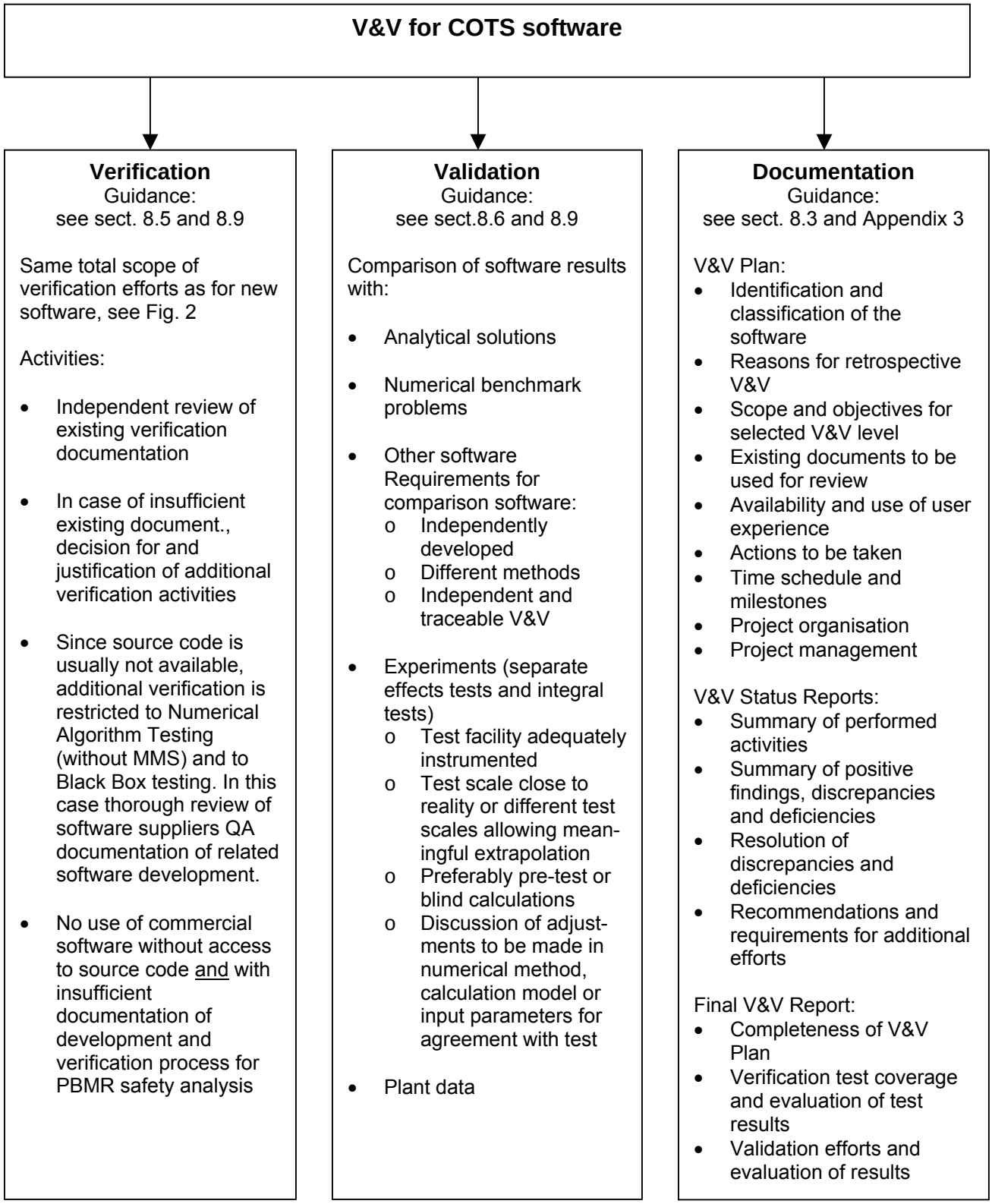
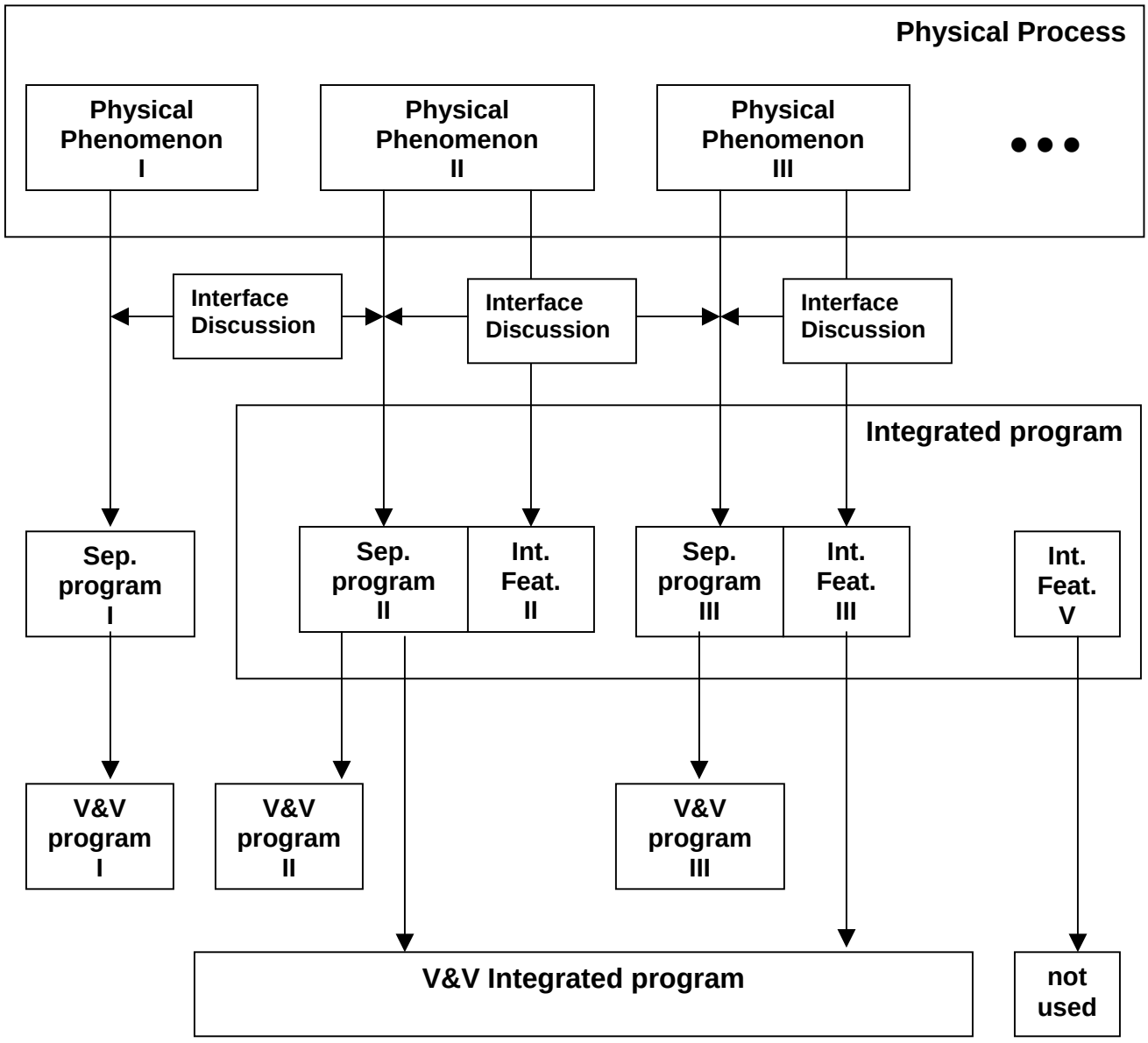


Figure 5 Separate Analysis of Physical Phenomena



Legend to Figure 5:

Physical Process inseparable combination of physical or chemical phenomena which describes a particular sector of the totality of processes or a particular time range in the behaviour of the PBMR

Integrated program complex software program designed for the integral analysis of a variety of physical/chemical phenomena

Int. Feat. integrated feature: capability of an Integral software to analyse a particular physical/chemical phenomenon together with others

Sep. program software program designed for the separate analysis of a particular physical/chemical phenomenon

Appendix 1. Examples for Physical and Chemical Phenomena

Guidance for phenomena modeling is discussed in section 5 and examples are presented below.

- Neutron physical phenomena related to
    - Interactions between neutrons and materials, reaction rates
        - Fission
        - Absorption
        - Scattering
    - Neutron flux distribution
    - Excess reactivity and reactivity control
    - Temperature coefficients
    - Nuclear transient behaviour of the core
    - Build-up of fission products
    - Fission product decay, rate of residual heat production
- Characteristics of pebble flow through the core and the discharge tubes
- Heat transfer by
    - forced convection,
    - natural convection,
    - thermal radiation and
    - conduction of heat
- Pressure drop in the primary coolant due to
    - Friction
    - Change in flow cross section
    - Absorption of heat in the fluid, phase changes
- Chemical reaction of hot graphitic surfaces (fuel elements, reflectors) with intruding air or water/steam or other chemical impurities of the primary circuit.

- Behaviour of fuel and fission products
    - Pressure increase and fission product retention in coated particles (source term of coated particles)
    - Fission product retention in the fuel elements (source term of graphite spheres)
    - Depletion in the primary circuit and in building structures
    - Propagation in building structures and environment (source term of primary circuit, plate out and distribution)
- Radiation, activation build-up in and shielding of components
- Behaviour of metallic shells regarding
    - Strength to sustain loads resulting from weight, pressure, thermal expansion and external loads in terms of induced vibrations due to earthquake, tornado, airplane crash, etc.
    - Fatigue and creep under the above mentioned loads
    - Fracture mechanics effects related to the assumption of incredibility of failure
- Behaviour of metallic and non-metallic support structures
- Behaviour of civil structures in cases of
    - Operational loads
    - External loads in terms of induced vibrations due to earthquake, tornado, airplane crash, etc.
    - Internal loads in terms of pressure build-up, shock waves, fire, etc.

Appendix 2. Good Programming Practises

Procedural Programming (based on /2/)

- A source and revision control system should be used to track, document and control changes to the source as well as to re-locatable and executable files and to maintain a known configuration.

- Programming languages should conform to the respective ANSI/ISO standards, the use of extensions to standard language and the use of assembly language should be avoided.

- Variable names should be unique and consistent throughout the program. Meaningful names should be used. The number of specification statements for a variable should be minimised and localised. Arrays should be dimensioned and used in only one way.

- Code dependence on word size or endian type should be avoided.

- The use of hardware or software configuration specific features such as system service calls, graphics calls, inter-processor communications, database interfaces, etc. should be avoided. If necessary for program functionality, they should be localised within individual subprograms, should be fully documented, and should make use of any de facto standards.

- Source documentation should be achieved by means of meaningful and concise comment statements and by external documentation in form of a text file delivered with the source.

  Comment statements should be used to:
  - Define the purpose of the program
  - List the name, author(s), edition, modification level, date of program preparation, original program hardware/software, and compiler
  - Explain the function of each subprogram and define its arguments
  - Include a table of key subprogram variables, defining their meaning, context, units, default values, and identifying any word size dependencies
  - Describe adjustable dimension techniques and explain storage allocation requirements
  - Explain major decision branches and functions of logical blocks of statements within subprograms

  Comment statements should not include out of date details relating to previous code hardware, operating systems, compilers, etc.

  The external documentation should:
  - Provide a program history listing, including conversion (platform porting) efforts or modifications
  - Identify and describe the content and format of all external files, including input files, output files, and required data files or libraries
  - Identify any design and test tools used in development of the software
  - Provide a brief description of all files on machine-readable media

- Recommendations for program structure and flow:
  - A program should consist of a carefully organised collection of subprograms, each of which performs a specific, well-defined function.
  - The computational flow should be an orderly progression through the program beginning with input data processing, checks and edits, followed by computation and data processing, and ending with final edits, output processing and data saving.
  - The overall program control should be contained in a single routine.
  - The program structure has to permit algorithms to be changed easily.
  - Subprograms should be of reasonable size and perform a well-defined function.
  - All interfaces with external packages such as mathematical libraries, plotting packages, special routines, etc. should be fully documented. Packages which are themselves intrinsically non-portable have to be avoided.

- Data handling:
  - Array dimensions should be conveyed to subprograms by use of symbolic parameters. Storage allocation methods should be used only to the extent they are part of the language used.
  - Only data transfer techniques defined in the programming language should be applied. If a special technique is required, it should be contained in one or in few service routines. Any use of special techniques should be thoroughly documented. Opening, reading, writing and closing of data files should be localised.

- Program features regarding initialisation and input processing:
  - organisation of related data into well-defined blocks
  - initialisation of arrays and variables to predefined values
  - visual listing or ‘echo’ of input data
  - visual representation of problem geometry based on input data, where applicable
  - identification of all input files utilised for the problem
  - checking of input data for consistency (out-of-range, null sets, wrong type, too many or not enough data points, etc.)
  - assignment and definition of program constraints only once.

- Error detection and diagnostics during computation and data processing:
  - Information about the computational path, intermediate results and progress of calculation should be optionally available
  - Invalid calculations caused by inconsistent data should be prevented by appropriate checks
  - Information should be issued when a program is executing outside its range of validity
  - Error detection or exhaustion of allocated resources should result in descriptive and meaningful diagnostic information and appropriate termination processing

- Output consisting of formatted results and external data files should be provided for both normal and abnormal termination. It should permit the user to obtain information selectively and in forms appropriate to its end use. Abnormal termination should provide information as to the nature of the problem and where it occurred.

- Where data is to be kept over a longer period or used on different platforms, input and output files in ASCII text should be preferred to binary files. Where binary files are used, the reasons for this should be documented. Output files containing data needed for later processing should be structured with respect to this processing.

Additional requirements for software using Graphical User Interfaces (GUI)

It is recommended that GUI programs require some additional constraints to be used satisfactorily for nuclear safety analysis work. Due to the extra program complexity introduced by using a GUI the program design process should:

- enforce the development and documentation of complete state transition diagrams for the operation of the GUI and its interactions with the rest of the software and

- have a debug mode in which each screen state is uniquely identifiable;

- include a facility to record keystroke history and to allow this history to be replayed for debugging and/or standardisation of operations;

- enforce icon disablement when appropriate for the current program state so that choices are clear to the user and inadvertent program processing actions do not occur. [This is good practice for all GUI programs].

Object Oriented Code Development Guidelines and Rules

The guidelines presented here are mainly at a relatively ‘high’ level since entire books are already available for guidelines at lower level and the latter tend to be somewhat specific to the programming language in use. It is judged to be most useful to provide recommendations to link some of these lower level guide-lines with the special requirements for nuclear safety analysis.

A useful discussion of object oriented methods and guidelines which is not too programming language specific and which includes many low level details is provided by /A2-1/.

Requirements analysis and design

Experience suggests that there needs to be more emphasis on system requirements analysis and design activities for successful object oriented systems development than for procedural development. An object oriented design can often be extended readily, with fewer knock-on effects than a procedural system model, due to better encapsulation and the related looser coupling of modules than in a procedural model. However, this is only true where good choices have been made for the classes used to form the basic architecture of the system model so that these truly reflect the structure, interactions and relationships of the system itself.

It is also noted that an iterative design process is often needed for more complex systems as a deeper understanding of the requirements can sometimes occur whilst the system model is being developed and applied. Thus the spiral model of the software development lifecycle may be more appropriate than the traditional waterfall model. The spiral lifecycle model is discussed in many software engineering texts including /A2-2/ and /A2-3/.

The design and coding should only include those features which are deemed to be strictly necessary to improve simplicity and to limit the scope for errors. Development ‘feature creep’ should be avoided.

Some level of modularisation of the design classes into groups, should be performed on a systematic basis, to simplify tracing execution paths and relationships.

Modelling

Where the programming language in use supports multiple inheritance and it is proposed to use this feature, very careful consideration should be given to whether this is appropriate and necessary since if mis-applied it can lead to significant complexity with development over time and the related increased potential for errors. It is recommended that adoption of this feature should be discussed and decided in a software design ‘walk-through’ review meeting and, where adopted, that the rationale should then be immediately documented in full, together with the views of the staff involved.

It is important to make the appropriate use of class inheritance and object containment in the model of the system being created. If there appears initially to be a choice the relative advantages of one to the other may appear minimal but if the system is later developed further an inappropriate choice can lead to a far more complex model and consequently potential errors. This is particularly the case if the inheritance is introduced to serve some short term goals of the model rather than being a true manifestation of the system being modelled. The correct choice of inheritance or containment can often be determined by comparing ‘is a ‘ and ‘has a’ relationships with the system relationship in question.

Importance of interface design

The encapsulation and polymorphism attributes of object oriented software development mean that it is quite likely a (probably lower level) class may get used in future in a completely different situation to that which it was originally written for. This places an increased emphasis on the correct and detailed documentation of the class interface in particular. Documentation describing the relationships and interface between base and derived classes is particularly important to help avoid problems such as breaking some implicit assumption of the base class when developing a derived class, for example.

Generalisation

The balance between class size, generality and complexity needs to be considered and struck carefully to limit the scope for errors arising from over complexity of individual classes whilst also avoiding complexity and potential for mistakes arising from using too many small classes with a consequently large number of interfaces to control. Use of a large number of smaller classes can also cause problems with a large object creation cascade and related debugging and lengthy execution time problems. As an additional guide it should be possible to summarise the purpose of a class in a short sentence.

Design patterns

Various object oriented model design ‘patterns’ have been identified which recur in many different systems and which have been documented as such e.g. /A2-4/.The system model can (and probably should) utilise these patterns if they reduce the complexity of the overall model provided this does not require the representation of the system to be distorted at all to fit in with the pattern and provided the developers are sufficiently experienced to develop with them.

Class design

Classes should be designed to be :

Relevant, logical, loosely coupled, lightweight, portable, reusable and sub-classable.

This list is from /A2-1/ but has been re-ordered to have the most significant aspects with regard to nuclear safety analysis software first (although the later aspects are judged to have roughly equal priority). Note that re-usability and sub-class-ability should not be chased at the expense of unnecessary complexity since clarity and simplicity are judged to be more significant for safety analysis software.

Pluggability

Some object oriented software is designed to provide the capability to accept software plug-in modules. For new software developments this capability should only be added where absolutely necessary since otherwise it is just an un-necessary extra complexity and a potential source of errors and problems.

If the plug-in capability is added the software design should ensure the interface is well encapsulated and thoroughly documented using state and data flow diagrams as appropriate.

Where incorporated in existing or new software the testing strategy should ensure all tests are run with and without each software plug-in needing to be used in the safety analysis and changes in behaviour identified, documented and justified. Validation calculations should be run with and/or without the plug-in as required for the subsequent safety analysis so that a consistent software system is applied for validation and analysis.

Methodologies

Significant new code/part code developments should consider creation of a model for the software system using one of the recognised modelling methodologies where the system involved has significant complexity or where the system and model are expected to develop over time to a situation where significant complexity is present. There are a range of modelling languages which can be considered. Unless features of a specific methodology are needed for a particular application a fairly widely used methodology should be applied. This will allow the same methodology to be applied over the range of different applications requiring such an approach (and staff can easily become familiar with the technology involved). One fairly widely used methodology is UML (Unified Modelling Language) and a good initial introduction to this is provided by /A2-2/.

Language choice

A range of programming languages offer object oriented features, for example C++, Objective-C, Smalltalk, Eiffel, Python and Ruby. The single most important aspect with regard to choice of language for use in the development of nuclear safety analysis applications is the availability of staff thoroughly experienced in their use. This is arguably more important for the object oriented software development than for procedure software development as some of the former can be more complex to learn. Be wary of choosing any language which develops over time and which does not have a good record of forward and backward compatibility or has a supplier with such a record. It is recommended that languages with open internationally agreed or de-facto standards be adopted.

Some special considerations apply to the Java language and any proposed use of this language should be identified to the regulator prior to any significant developments taking place.

Object oriented development can also be applied to some extent with languages not providing direct support of object oriented methods but this is not generally recommended unless there are strong reasons for doing so which should be documented and justified. A discussion of possible problems with such an approach is presented in /A2-5/.

Use with legacy coding

Where object oriented software needs to be used in conjunction with legacy coding, special consideration needs to be given to the design details to ensure that successful development occurs. It is generally better to keep the new coding and legacy coding segregated to a degree to simplify debugging, documentation and improve portability for example. One strategy often applied is to build an object ‘wrapper’ around relevant segments or modules of the legacy coding so that the latter is essentially encapsulated. This approach is discussed in /A2-5/.

Coding notes

- Only overload method names when the entities or functionality is related, not as a short-cut which may immediately or later distort the design. Variable names should usually be distinct for clarity.

- Limit the use of global data to improve modularity.

- Ensure class methods access class data via accessor methods when appropriate to reduce coupling levels within classes. This becomes more relevant if a class becomes larger/more complex over time and can simplify any later re-structuring involving the break-up of such a class.

- Make use of namespaces if these are supported by the programming language in use to avoid the potential for erroneous name clashes.

- Various identifier case conventions are in use. The important point is to be consistent for clarity. Where a particular convention is recommended as part of the language specification this should be used to avoid confusion (particularly with staff changes over time). Strike a

balance between short over abbreviated identifier names and over-long names which retain meaning but which can obscure the operations being applied to the identifiers. Choose variable identifiers which will still be meaningful if a method or function is re-used in another context in future.

- Make explicit use of public, private and protected visibility or similar if available for clarity. Include comments to justify selection if needed.

- Use arrays rather than pointers if possible for clarity.

- Avoid functions with variable numbers of arguments as far as possible for clarity.

- Avoid use of default arguments as far as possible for clarity.

- Avoid repeated use of particular hard-coded numeric values. Use a variable or constant definition instead to reduce the scope for typographical errors.

- Ensure class encapsulation does not 'leak' implementation details, particularly as classes are developed over time.

- Ensure all required methods and functionality are coded into the classes, do not confuse incomplete prototype coding and final coding.

Frameworks and class libraries

Frameworks can be regarded as libraries of cooperating classes for the purposes of this discussion though frameworks are often described as being applied in a program at a higher level of design abstraction.

Use of existing frameworks and class libraries provides the benefit of using software which has been more widely used than that developed specifically for a single application but has several potential disadvantages including :

a) The V&V available for the framework/library may be insufficient and difficult to complete by virtue of size, complexity and generality of the software.

b) The application programming interface (API) information may be inaccurate but this could be difficult to establish and check by virtue of library complexity.

c) The framework/library may be under the control of a separate organisational entity and so to some degree out of the control of the nuclear safety analysis code developer. Thus, for example, a new version might be forced on a user by the requirement to apply a particular bug correction but bring with it a range of other changes which may not initially be fully understood, documented etc.

d) A significant limitation in the framework/library may be noted late in the development cycle which increases the risks to the development of a satisfactory quality on the timescale required by the analysis programme.

e) The framework/library may not be portable to a different platform. If a replacement is introduced this may yield different results.

For these reasons it is recommended that particular care should be exercised in the decision to adopt any framework or class library. Such dependencies should be identified in the code documentation available to the regulator.

Unit testing

The encapsulation offered by classes and objects should encourage the unit and group testing of such software. Some languages provide the capability for test harnesses to be built into the classes directly and used to, for example, provide simple regression testing. Where such facilities are used care should be exercised that the some of the test harness software does not get activated inadvertently in actual safety analysis calculations.

References:

/A2-1/ B.F. Webster, 'Pitfalls of Object-Oriented Development'.
Publishers: M&T Books division MIS:Press Inc. 1995.
ISBN 1-55851-397-3.

/A2-2/ R. Pooley and P. Stevens, 'Using UML: software engineering with objects and components'.
Publishers: Addison Wesley Longman Limited. 1999.
ISBN 0-201-36067-5.

/A2-3/ B. Boehm, 'A Spiral Model of Software Development and Enhancement'.
Proc. of an Int'l Workshop on the Software Development Process and Software Enhancements.
Coto de Caza, Trabuco Canyon, California, 1985.

/A2-4/ E. Gamma et al, 'Design Patterns: elements of reusable object-oriented software'.
Publishers: Addison Wesley Longman, Inc. 1995.
ISBN 0-201-63361-2.

/A2-5/ 'Developing Object-Oriented Software : An Experience Based Approach'.
Publishers: Prentice-Hall PTR, Prentice Hall, Inc. 1997
ISBN 0-13-737248-5.

Appendix 3. Documentation of Computer Software

According to /3/ the documentation of computer software should consist of four essential parts (respective guidance is also provided in /16/):

- Abstract
- Application Information (User Manual)
- Problem Definition Information
- Programming Information

In detail these parts should contain the following information:

- **Abstract:**
    - Description of application or system area and of specific functions included
    - Sufficient information for a potential user to determine if the software is of interest
    - Identification of the required computer environment, i.e., system software and hardware configurations
    - Paths to further information on the software
    - Statements on what software materials are available and how they may be obtained
- **Application Information (User Manual):**
    - Description of the nature of the problem solved, the processing tasks performed, and of the methods and procedures employed
    - Schematic display showing the flow of calculations
    - Program considerations:
        - Description of the functions of each major program option with special attention to effects of combinations of options
        - Discussion of alternative paths which may be dynamically selected by the program depending on calculated results
        - Restrictions on the range of values of variables
        - Restrictions on data array size
        - Dependence of data storage requirements on problem input parameters
        - Specification of values assigned to constants built into the program, basis for their selection
        - Discussion of restart and recovery procedures from an application point of view
        - Documentation and explanation of error-checking procedures and error messages

- Discussion of interactive aspects of program execution (e.g., screens and menu options)
- Information to assist in estimating execution time and other computer resource usage
- Indication of any special forms of output (e.g., microfilm, graphics display)

o External Data Files:

- Contents and organisation of each external data file
- Usage of data files related to program execution
- Reference to auxiliary programs which create, modify, display, or edit these files. Alternatively indication of availability of such programs

o Input Data:

- Description of special input techniques and requirements, e.g., blank field treatment, order of items field delineation
- General conventions governing default values
- Description of the handling of consecutive cases, conditions for data retention or re-initialisation for the next case
- Specifications for input variables: name, description, format, dimensional unit, and default value, if appropriate

o System control requirements

- List of operating system control commands
- Interdependence of commands and input options or data files

o Output information

- Discussion of program output
- input options
- Relation of output to appropriate equations
- Description of any normalisation of results and listing of associated dimensional units

o Sample problems

- Description of the physical problem and associated data files
- Presentation of the input data and results generated from execution of the software with these data
- Computer environment for the sample problem and resource usage

Benchmark problems or other well-defined and documented examples should be selected which exercise the major program options. The output should be representative for the options exercised

- **Problem or Function Definition Information:**
    - Comprehensive description of the problems solved and data processing functions performed
    - Self-contained (as far as reasonable) description of the physical theory in terms of a mathematical model. Sufficient detail is needed to permit users to judge the suitability of any model to a particular application. Any assumptions made should be noted, and limitations imposed by each assumption should be discussed.
    - References to the sources for the models and mathematical formulations incorporated
    - Algorithms and numerical techniques
        - Description of the computational algorithms used (in conventional terminology rather than programming language syntax)
        - References to documented algorithms and numerical techniques
        - Discussion of results obtained by important algorithms
        - Identification of any known dependency of the results on the particular computer environment used for program development
        - For iterative solutions, discussion of use and interpretation of convergence tests and recommendation of input values for convergence criteria
        - For probabilistic solutions, discussion of the precision of results having a statistical variance
    - Background information about the source, contents, verification, and use of data in library files
    - Description of applied validation methods, reference to relevant validation efforts undertaken by others
    - Inclusion of mathematical error bounds determined from an error analysis to assist the user in interpreting results
- **Programming Information:**
    - Identification of source language
    - Diagrams showing the overall program structure and logic, detailed flowcharts where appropriate
    - Optionally a listing of source code containing comprehensive comment lines and a cross-reference list of subprogram names, parameters and entry points
    - Documentation of any known dependency upon the local computer environment
    - Program and subprogram details:

- Definition and function of main program and each subprogram, identification of argument lists and their use
- For a particular subprogram, indication of those subprograms which call it and those which it may call
- Relation of problem variables and constants to the program labels
- Description of shared storage assignments(e.g., COMMON in Fortran
- Detailed description of the functions performed by machine-dependent subprograms that are unique to this program
- Discussion of programming conventions and special features such as re-entrant code

o Details of data files:

- Specification of file name, usage, structure, mode, and data elements
- Discussion of the procedures related to the use and maintenance of data libraries, databases, and files
- Statements on data file retention and allocation considerations
- List of logical devices employed. Description of the use of each device, any associated data blocking or compression schemes, and any networking protocol assumed. Identification of the information and its format resident on each device. Requirements for use of related physical devices

o Program implementation requirements:

- Hardware:
  - List of computer environments on which the software has been tested successfully including descriptions of the equipment used and minimum requirements for interactive processing
  - Central memory requirements, amount and type of auxiliary storage and peripheral equipment
  - Identification of any special hardware utilised
  - Description of unique networking requirements
- Software:
  - Identification of the required operating system components, language processors, and subroutine libraries
  - Listing of proprietary and commercial software products required
  - Producer and vendor information including version numbers release dates
- Programming considerations:
  - Examples of system control command sets for installation and use of the software
  - Documentation of overlay or segmentation schemes used
  - Description of storage allocation and data management procedures.
  - Discussion of restart, recovery, and successive case capabilities

Appendix 4. Example - Validation of the ATHLET Code

The thermal-hydraulic computer code ATHLET (Analysis of THermal-hydraulics of LEaks and Transients) is being developed by the Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) for the analysis of anticipated and abnormal plant transients, small and intermediate leaks as well large breaks in light water reactors.

The aim of the code development is to cover the whole spectrum of design basis and beyond design basis accidents (without core degradation) for PWRs and BWRs with only one code. The main code features are:

- advanced thermal-hydraulics
- modular code architecture
- separation between physical models and numerical methods
- pre- and post-processing tools
- portability

The code development is accompanied by a systematic and comprehensive validation program. A large number of integral experiments and separate effects tests, including the major International Standard Problems, have been calculated by GRS and by independent organisations.

OECD/NEA/CSNI (Organisation for Economic Co-Operation and Development/ Nuclear Energy Agency/ Committee on the Safety of Nuclear Installations) issued several reports on the validation of thermal-hydraulic computer codes and selected therein experiments suitable for the validation of such codes.

Report /A4-1/ deals with integral experiments and contains validation matrices for different classes of LOCA and transients. In these matrices the dominating physical phenomena are contrasted with the test facilities. In addition details regarding the types of experiments are given. Fig. A4-1 shows as an example the cross reference matrix for experiments related to large breaks in PWRs.

The reports /A4-2/ and /A4-3/ contain a total number of 185 separate effects test facilities (among them also integral experiment facilities, if they are sufficiently equipped to isolate separate effects) which are suitable for the evaluation of 67 separate thermal-hydraulic phenomena. Information sheets are available for 113 test facilities. Cross reference matrices are included depicting the physical phenomena versus the test types, the test facilities versus the physical phenomena, and the test types versus the test facilities. Fig. A4-2 shows as an example a test facilities versus phenomena cross reference matrix for test facilities located in Germany.

Based on these reports a well-balanced selection of integral experiments and relevant separate effect tests was compiled for the systematic validation of the ATHLET code. In this appendix only that part of validation shall be reported that covers PWR applications, not considering VVER reactors. Since the validation of ATHLET is a continuous task accompanying the development of the code, a certain point in time had to selected for this appendix. The following information reflects the status of ATHLET validation in June 1997.

The ATHLET validation matrix of integral experiments for PWR applications is shown in Fig. A4-1. It is differentiated regarding the following classes of events:

- large breaks
- small and intermediate breaks
- transients at system power
- shut-down transients
- severe accident for evaluation of accident management activities

For each class of events, experiments were selected with the aim to examine the relevant thermal-hydraulic phenomena, and each test type contained in the matrices, in at least three experimental facilities with different scales. In addition transients from existing plants have been incorporated.

Regarding separate effects tests a number of 91 relevant tests for both PWR and BWR has been selected from the totality of tests compiled in /A4-2/ and /A4-3/.

Besides the depicted total validation, particular validation is performed whenever a new code version is released. This particular validation can be divided in:

- “Update” validation for the release of new code versions without significant changes of the code, comprising recalculation of:
    - standardised samples
    - Relevant separate effect tests relevant for the code modifications made
    - one selected PWR integral experiment

- “Basic” validation for the release of new code versions with major modifications of the code regarding for example the system of conservation equations and the constitutive equations or the code structure. This case comprises the recalculation of:
    - standardised samples
    - Relevant separate effect tests relevant for the code modifications made
    - four selected PWR integral experiment

References:

/A4-1/ OECD/NEA/CSNI: CSNI Integral Test Facility Validation Matrix for the Assessment of Thermal-Hydraulic Codes for LWR LOCA and Transients; OCDE/GD(97)12, Paris 1997

/A4-2/ N. Aksan, F. D’Auria, H. Glaeser, R. Pochard, C. Richards, A. Sjöberg: QECD/NAE-CSNI Separate Effects Test Matrix for Thermal-Hydraulic Code Validation, Volume I: Phenomena Characterisation and Selection of Facilities and Tests; OCDE/GD(94)82, Paris 1994

/A4-3/ N. Aksan, F. D’Auria, H. Glaeser, R. Pochard, C. Richards, A. Sjöberg: QECD/NAE-CSNI Separate Effects Test Matrix for Thermal-Hydraulic Code Validation, Volume II: Facility and Experiment Characteristics; OCDE/GD(94)83, Paris 1994

Fig. A4-1: Cross Reference Matrix for Large Breaks in PWRs (Fig. 1 of /A4-1/)

- Phenomena versus test type
  - + occurring
  - o partially occurring
  - - not occurring
- Test facility versus phenomenon
  - + suitable for code assessment
  - o limited suitability
  - - not suitable
- Test type versus test facility
  - + performed
  - o performed but of limited use
  - - not performed or planned

		Test Type			Test Facility and Volumetric Scaling						
		Blowdown	Refill	Reflood	CCTF 1:25	LOFT 1:50	BETHSY 1:100	PKL 1:134	LOBI 1:712	SEMISCALE 1:1600	UPTF 1:1 (a)
Phenomena	Break flow	+	+	+	+	+	+	+	+	+	+
	Phase separation (condition or transition)	o	+	+	+	+	+	+	+	+	+
	Mixing and condensation during injection	o	+	+	o	o	o	o	o	o	+
	Core wide flow + flow distribution	o	+	+	o	o	o	o	o	-	+
	ECC bypass and penetration	o	+	o	o	+	-	o	o	-	+
	CCFL (UCSP)	o	+	+	o	o	o	o	o	-	+
	Steam binding (liquid carry over, etc.)	-	o	+	o	o	-	o	o	o	o
	Pool formation in UP	-	+	+	o	o	o	o	o	o	+
	Core heat transfer incl. DNB, dryout, RNB	+	+	+	+	+	+	+	o	o	-
	Quench front propagation	o	o	+	+	+	+	+	-	+	-
	Entrainment (Core, UP)	o	o	+	o	o	o	o	o	o	+
	Deentrainment (Core, UP)	o	o	+	o	o	o	o	o	o	+
	1- and 2-phase pump behaviour	+	o	o	-	o	-	o	+	+	-
	Noncondensable gas effects	-	o	o	-	+	+	-	-	-	+
Test Facility	CCTF	-	o	+							
	LOFT	+	+	+							
	BETHSY	-	-	+							
	PKL	o	+	+							
	LOBI	+	+	-							
	SEMISCALE	+	+	+							
	UPTF	o	+	+							

Important test parameter
- break location / break size
- pumps off / pumps on
- cold injection / combined injection

Fig. A4-2: Separate Effects Test Facility Cross Reference Matrix (Table 4.4 of /A4-2/)

Phenomena		Separate Effects Test Facilities															
LEGEND x suitable for model validation o limited suitability for model validation - not suitable for model validation					4. Germany												
		UPTF	HDR Vessel	BATTELLE PWR RS 16	BATTELLE BWR 150396	Blowdown Heat Transfer RS 37	Heat Transfer Refill / Reflood RS 36	Steady State DNB Exp. RS 164	Trans. Boil. Ins. Tube (Freon) RS 370	Rewet RS 62/184	Thermodyn. Nonequilibrium RS 77	LOCA Pump Behaviour RS 92	Thermalhydr. UP-BBR RS 373	Pressuriser-Valve RS 240,347,636	Steam / Water Disch. Flow RS 93,397		T-Junction Test Facility (KfK)
	Facility No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Info Sheet available	a	a	a	a	a			a	a	a		a		a		a
0 BASIC PHENOMENA	1 Evaporation due to Depressurisation	-	x	x	x	-	-	-	-	-	x	-	-	-	o		-
	2 Evaporation due to Heat Input	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 Condensation due to Pressurisation	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	4 Condensation due to Heat Removal	-	-	-	-	-	-	-	-	-	x	-	-	-	-		-
	5 Interfac. Frict. Vertic. Flow	-	o	o	o	-	-	-	-	-	-	-	-	-	-		-
	6 Interfac. Frict. Horiz. Flow	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	7 Wall to Fluid Friction	-	-	-	-	-	-	-	-	-	-	-	-	-	-		o
	8 Press. Drops at Geometr. Discontinuities	o	o	-	o	-	-	-	-	-	-	-	-	-	-		x
	9 Pressure Wave Propagation																
1 CRITICAL FLOW	1 Breaks	-	x	x	x	-	-	-	-	-	-	-	-	-	x		x
	2 Valves	-	x	-	-	-	-	-	-	-	-	-	-	x	-		-
	3 Pipes	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
2 PHASE SEPARATION / VERTICAL FLOW	1 Pipes / Plena	x	o	x	x	-	-	-	-	-	-	-	-	-	-		-
	2 Core	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 Downcomer	x	-	o	o	-	-	-	-	-	-	-	-	-	-		-
3 STRATIFICATION IN HORIZ. FLOW	1 Pipes	x	x	-	-	-	-	-	-	-	-	-	-	-	-		x
4 PHASE SEPARATION AT BRANCHES	1 Branches	-	-	-	-	-	-	-	-	-	-	-	-	-	-		x
5 ENTRAINMENT / DEENTRAINMENT	1 Core	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	2 Upper Plenum	x	-	-	-	-	-	-	-	-	-	-	x	-	-		-
	3 Downcomer	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	4 SG-Tube	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	5 SG-Mix. Chamber (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	6 Hot Leg with ECCI (PWR)	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
6 LIQUID-VAPOUR MIXING WITH CONDENSATION	1 Core	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	2 Downcomer	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 Upper Plenum	x	-	-	-	-	-	-	-	-	o	-	-	-	-		-
	4 Lower Plenum	o	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	5 SG-Mix. Chamb. (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	6 ECCI in Hot and Cold Leg (PWR)	x	x	-	-	-	-	-	-	-	-	-	-	-	-		-
7 CONDENSATION IN STRATIFIED CONDITIONS	1 Pressuriser	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	2 SG-Primary Side (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 SG-Secondary Side (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	4 Horizontal Pipes	x	x	-	-	-	-	-	-	-	-	-	-	-	-		-
8 SPRAY EFFECTS	1 Core (BWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	2 Pressuriser (PWR)	-	-	-	-	-	-	-	-	-	o	-	-	-	-		-
	3 OTSG Second. Side (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
9 CCF / CCFL	1 Upper Tie Plate	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	2 Channel Inlet Orifices (BWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 Hot and Cold Leg	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	4 SG-Tube (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	5 Downcomer	x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	6 Surgeline (PWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
10 GLOBAL MULTIDIMENSIONAL FLUID TEMPERATURE, VOID AND FLOW DISTRIBUTION	1 Upper Plenum	x	-	-	o	-	-	-	-	-	-	-	o	-	-		-
	2 Core	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
	3 Downcomer	x	x	-	o	-	-	-	-	-	-	-	-	-	-		-
	4 SG-Secondary Side	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
11 HEAT TRANSF.: NAT. FORC. CONV.	1 Core, SG, Structures	-	-	-	-	x	x	-	-	x	-	-	-	-	-		-
SUBC. / NUCL. BOIL.	2 Core, SG, Structures	-	-	-	-	x	x	-	-	x	-	-	-	-	-		-
DNB / DRYOUT	3 Core, SG, Structures	-	-	-	-	x	x	x	-	x	-	-	-	-	-		-
POST CHF	4 Core, SG, Structures	-	-	-	-	x	x	-	o	x	-	-	-	-	-		-
RADIATION	5 Core	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
CONDENSATION	6 SG, Structures	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
12 QUENCH FRONT PROPAG. / REWET	1 Fuel Rods	-	-	-	-	-	x	-	-	o	-	-	-	-	-		-
	2 Channel Walls and Water Rods (BWR)	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
13 LOWER PLENUM FLASHING		o	o	o	o	-	-	-	-	-	-	-	-	-	-		-
14 GUIDE TUBE FLASHING (BWR)		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
15 ONE AND TWO PHASE IMPELLER-PUMP BEHAVIOUR		-	-	-	-	-	-	-	-	-	-	x	-	-	-		-
16 ONE AND TWO PHASE JET-PUMP BEHAVIOUR		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
17 SEPARATOR BEHAVIOUR		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
18 STEAM DRYER BEHAVIOUR		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
19 ACCUMULATOR BEHAVIOUR		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
20 LOOP SEAL FILLING AND CLEARANCE (PWR)		x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
21 ECC BYPASS / DC PENETRATION		x	-	-	-	-	-	-	-	-	-	-	-	-	-		-
22 PARALLEL CHANNEL INSTABILITIES (BWR)		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
23 BORON MIXING AND TRANSPORT		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
24 NONCONDENSABLE GAS EFFECT (PWR)		-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
25 LOWER PLENUM ENTRAINMENT		o	-	-	-	-	-	-	-	-	-	-	-	-	-		-

Fig. A4-3: ATHLET Validation Matrix, Integral Experiments for PWR Application (Extract)

Facility or plant	Scale	Large breaks	Small breaks, intermediate breaks	Transients	Shutdown transients	Accident management
UPTF/TRAM	1:1	6	2			3
CCTF	1:25	2				
LOFT	1:50	1	4	1		
LSTF	1:50		2	2		
BETHSY	1:100		7		3	5
PKL	1:145	3	8	6	1	6
LOBI	1:712	2	7	4		2
GERDA	1:1686		1			
German Konvoi				3		
Brokdorf				4		
total		14	31	20	4	16

Appendix 5. Example for a Multi-level Software Classification scheme

The following description presents information that is relevant when a software classification scheme is applied to track the relative levels of appropriate V&V and the actual available V&V for each software program as introduced in sub-section 7.2.

When a classification scheme is applied it is important for the number of classification levels to be the same for all software programs and for both safety case importance and validation status classification.

In terms of the licensing process a program is of high importance for the Safety Case, if

- it analyses phenomena which have the potential to substantially increase the release of fission products when they are underestimated in the analysis or
- it analyses physical processes whose influence cannot be sufficiently demonstrated at test rigs or in commissioning tests on site
- there are no comparable programs available to analyse the respective physical processes
- it is designated to analyse the behaviour of SSC which are safety classified. The classification of the program shall correspond to the classification of the SSC.

The classification of the V&V status should consider:

- the status of software documentation and
- the status of verification
- the coverage of program features by validation test calculations
- the number and range of validation test calculations
- the acceptability of the validation calculation results

The level of importance and the V&V status are each to be graded or classified into a number of levels (e.g. 2 or 3) from high to low importance/status and mismatches in the levels for Safety Case importance and V&V status should be used as an aid to the prioritising of V&V activities between different programs.

The classification levels should be updated regularly as progress is made in the V&V of the software and in the establishment of the Safety Case.