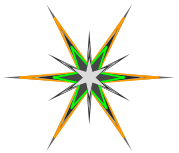


S T S A R C E S

Standards for Safety Related Complex Electronic Systems

Annex 1

Software engineering tasks - Case tools

Final Report of WP1.1

Florin Sobaru & Smaïn Bouazdi

CETIM

European Project STSARCES
Contract SMT 4CT97-2191

CONTENTS

FORWARD

The work described in this report was done within the framework of the European project STSARCES, acronym for “Standards for Safety Related Complex Electronic Systems”. This project groups together the main French and European organisations INRS, INERIS, CETIM, HSE, BIA, INSHT, SP, TÜV, and VTT, as well as the companies JAY ELECTRONIQUE and SICK AG, directly concerned by the safety of industrial systems. Five themes are treated by the various partners : software safety, equipment safety, validation of the safety of complex components, the connection between the European standard EN 954 and the project for an international standard CEI 61508, and the taking into account of technological innovations.

An in-depth study of the development methods and techniques for systems was also led. The contribution of the work done by CETIM concerns the drafting of safety software specifications and their validation. They highlight the importance of the global system approach.

Over the past years, traditional command systems have been replaced by programmed systems at an accelerated rate. The functionalities of these programmed systems has increased and become increasingly sophisticated, making them complex to produce. Their complexity indirectly causes an increase in potential faults in design and therefore failures in the systems designed.

The complexity and size of the present systems are such that it is impossible to eliminate all faults contained by a final control. To ensure that software development is mastered, it is necessary to established an adapted process.

Unlike mechanical systems, it is difficult to foresee the various failure modes. At the research office level, analysis of uncertain behaviour is not exhaustive and it is difficult to control and eliminate risks.

Since the software behaviour cannot be predicted and since potential incorrect behaviour cannot be quantified, it is impossible to analyse failure modes as for mechanical systems. Unlike equipment which may break down due to physical faults, software does not age and is really only affected by faults in design of human origin.

Introducing digital technology thus demands that designers make fundamental changes in their methods of approaching problems to be treated.

During the lifetime of software, one of the most delicate steps is to express needs in specifications. Drafting and evaluating specifications are important steps, especially for safety software.

Industrial fields in advanced technology (aeronautics, energy, railway transport, telecommunications) have integrated all of these concerns in the framework of developing their software. Concerning other sectors, such as mechanics, assistance, practical and easy operation documents must be made available to research offices,

encouraging the appropriation of safety software development techniques, and more especially, their specifications.

The main work today is essentially done in the universities. It is not easily accessible to non specialists and its implementation requires an intellectual effort and a significant investment in training and in computer tools.

The CETIM work is part of this “Safety software specification” vision. Our goal is to reduce the distance between the state of the art and present practices, thus making methods and tools easier to use.

1. ELECTRONIC SYSTEMS CONCERNING SAFETY

1.1 Dependable electronic systems

1.1.1 Need for a system approach

Often safety needs and the verification of their implementation are tasks accomplished afterwards. To take the safety aspect into account when expressing needs implies a modelisation of the system which is different from that obtained when only the performance and cost aspects are considered.

In fact, while defining the functional needs of a system provides a description of their service the system is to render, defining safety needs describes the behaviour which the system must avoid. This description leads to identifying the functions which the system must fulfil in order to reduce the possibility that the behaviour to be avoided occurs.

From a system point of view, the safety concept may be planned on various levels. At a global lever, which is that of the mission, safety expresses the absence of accidents or incidents, concerning people, needs or the environment, and it is associated with the safety function. At the component level, it expresses the absence of behaviour which could cause an accident for the specified mission. Finally, the safety concept may also be considered at an output piloted by a component.

To ensure a determined level of safety, risks must be analysed first. This analysis process is continuous and iterative. It intervenes early in system development. The idea is to identify dangerous phenomena and attempt to eliminate them. In order to do this, the dangerous state must be suppressed at the system operational level, or all dangerous phenomena must be suppressed.

Since it is impossible to eliminate all dangerous phenomena cannot be eliminated, the associated risks must be evaluated and estimated. When a risk is considered high, measures must be imagined to reduce it, either by decreasing its severity, or by decreasing its probability of occurring.

Risk analysis is led on four levels :

- Very early in the life cycle : preliminary analysis of risks identifies critical functions (safety functions) and highlights dangers.
- At the system level : makes it possible to identify risks introduced by interface between sub-systems and risks of human errors.
- At the sub-systems level : each sub-system is analysed and the safety criteria, during normal operation or in a degraded mode, concerning the sub-system is identified.
- At the support and operations level : analysis identifies the procedures to reduce danger during use and maintenance of the system.

1.1.2 Operation safety process

The most traditional process, in the automobile industry and in mechanics, to ensure operational safety of systems is based on experience feedback. Engineers collect and analyse operational dependability data in order to eliminate and control risks of failures. This systems safety approach is issued from an industrial culture mainly based on system testing, rather than on analysis.

A second process is based on system dependability. This approach measures the probability of random failures, rather than the probability of a risk of an accident. It is not efficient to test the safety of systems and software.

A consequence of these two approaches is the use of well-tried components. Safety is not the property of an isolated element, but the combination of the equipment, the software and the environment in which the system is used.

The system approach to safety mainly consists in identifying risks as early as possible and classifying them, in order to undertake corrective actions to eliminate or minimise them before system design choices are made firm.

Several models of life cycles have been developed :

The cascade model is simple. A certain number of steps (or phases) is agreed upon. A step must end with the production of certain documents or software. The results of the step are thoroughly reviewed, and the next step is taken only when this review is considered satisfactory.

The V model, which is more recent, presents a more realistic approach to the relationship between development activities and verification activities, when there is a software code.

Cascade and V models have disciplined the software development process, by identifying its main activity and by specifying their sequencing. However, the linear vision introduced by these models and their rigidity have called by modifications and extensions of these models.

The first evolutionary model is the incremental model. Only one sub-assembly is developed at a given time. Core software is first developed, then increments are successively developed and integrated.

Another form of evolutionary development consists in relying on modelling, a common practice in the field of engineering. Producing models makes it possible to specify the needs and desires of the user, either globally or by focussing on certain functions.

A representative model of this approach is the spiral model. Development according to this model begins with a preliminary analysis of needs which is refined during the initial cycles, taking into account constraints and risk analysis. The originality of this model is to surround the development itself with phases devoted to risk analysis and to the determination of safety objectives.

At present, there is a strong tendency to prefer the definition of system development models in order to master the development of complex systems (the standard MIL-STD 499B prepared by the American department of defence, version EIA/IS-632 of which applies to commercial systems).

In order to harmonise system safety evaluation methods, ITSEC criteria (Information Technology Security Evaluation Criteria) and a method of evaluation ITSEM (Information Technology Security Evaluation Manual) have been elaborated. In this method, the evaluation process is based on two aspects :

- The study of dependability, which strives to analyse whether or not the system is apt to fulfil its safety objectives, in its design principle,
- The study of compliance, which strives to analyse whether or not safety functions and mechanisms are correctly installed.

The standard IEC61508 presents a development model for critical electrical/electronic/ programmable electronic systems. This model presents a generic development process. The approach adopted distinguishes four levels of criticality.

Depending on the levels of criticality identified for a system and for the software, this standard recommends the application of operation safety methods and techniques.

The various models described above mix fundamentally different activities, development itself and verification, and conserve the strict sequencing of activities.

The standard DO-178B, specific to the aeronautics industry, makes this separation. It recommends system structures which use design techniques allowing for partitioning, heterogeneous redundancy and monitoring. It offers a new software development model, the process model [LAP95]. Its B review consider that information on the system level are necessary as an entrance point into the software development process.

An explicit operation safety development model is proposed by LAAS-CNRS [LAP95]. It presents a global view and summarises the main activities required to develop a dependable operating system : fault prevention, fault tolerance, fault elimination and fault forecast.

The state of the art shows that developing a dependable system requires the integration of operation safety activities throughout the life cycle. It therefore becomes necessary to be able to certify critical systems, no longer only software.

1.1.3 Obtaining and validating a dependable system

Figure 1.1 shows the relationship between use of means, in a “traditional” quality process, to strive for a system exempt from faults, and the operation safety process, which implements other operation safety means in order to strive for a system exempt from failures.

While fault prevention attempts to prevent faults from occurring or from being introduced, fault forecasting attempts to estimate the presence, creation and consequences of faults.

Certain methods of evaluation are entirely ordinal, such as AMDEC (Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité – Analysis of Failure Modes, their Effects and their Criticality) or APR (Analyse Préliminaire des Risques – Preliminary Risk Analysis); others are entirely based on probability such as MARKOV chains. Finally, certain methods may be used for both aspects, such as dependability diagrams and failure tree diagrams.

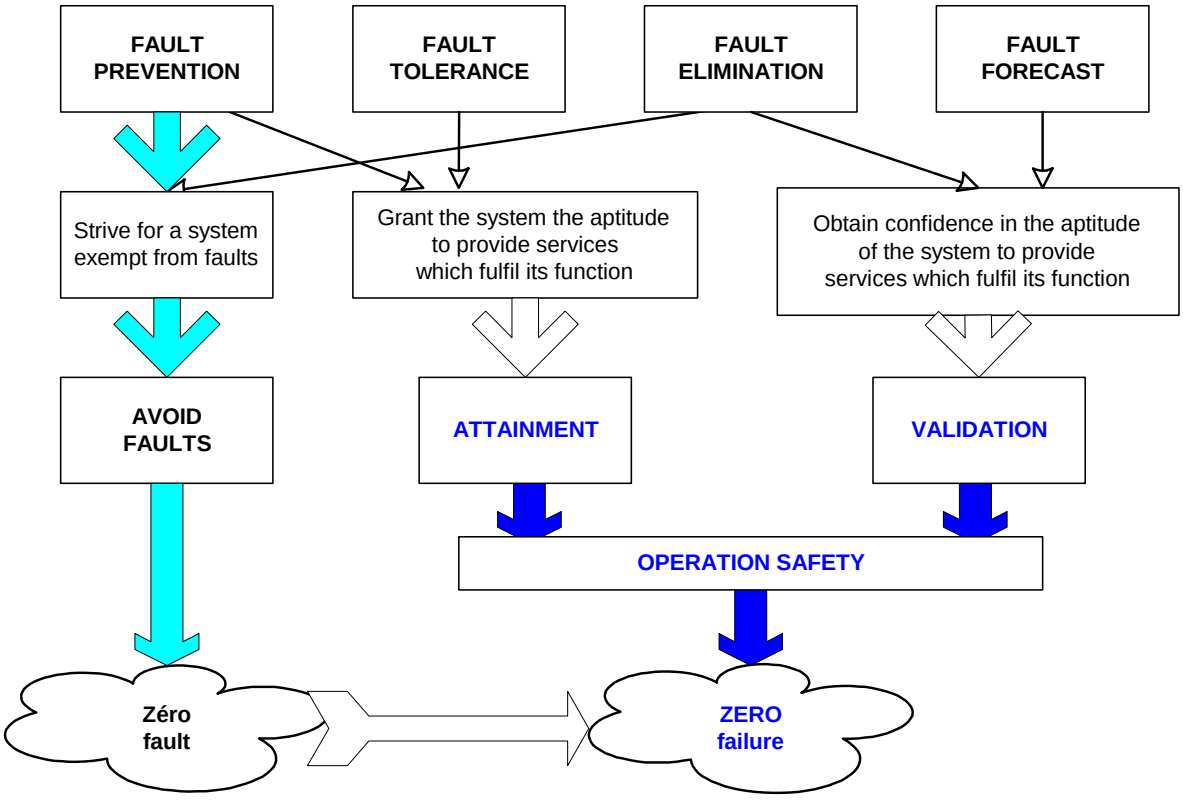


Figure 1.1 : Elements comprising operation safety.

Three types of processes may be distinguished from among the forecasting methods :

1. The inductive process moves from a particular situation to a more general situation. This is a detailed study of the effects consequences of failures have on a system,
2. The deductive process moves from a more general situation to a more particular situation. This is the study of the causes of a failure on a system,
3. The hybrid process is a combination of the two preceding processes.

Methods of evaluation based on probability require a modelling activity which consists in elaborating an analytical model parameterised with the rate of failure of each component in the system.

The two most recognised and most used models are MIL-HDBK-217F and RdF93. The MIL-HDBK-217F model is recognised on an international level and remains the reference in the electronic industry. The RdF93 model is a dependability collection published by CNET in France, more particularly for the telecommunications sector.

Fault elimination attempts to reduce the presence of faults, in quantity and severity, by three phases : verification, diagnostic and correction. After the correction phase, non regression must be verified, in order to ensure that elimination of the fault did not have any undesirable consequences.

Fault tolerance attempts to provide a service capable of fulfilling the function or functions despite the faults.

In many sectors of activity, the cost restriction does not allow for use of material redundancy. However, some fault tolerance techniques may be implemented in order to improve the dependability and safety of electronic systems :

- Watchdogs to check that the process is not blocked,
- Timer degradation to ensure processing speed,
- Inlet networks to filter parasites,
- Protection diodes, against overpressure (transitory or load dump),
- Inlet tests (limit values or loss of information),
- Outlet tests (intelligent power circuit for the diagnosis),
- Checksum on the read-only memory to detect memorisation errors which affect the software.

1.1.4 Explicit operation safety for mecatronic systems

A global mecatronic system is composed of two main sub-assemblies : the physical system which identifies the various mechanical hydraulic, pneumatic, electric, etc. parts, and the electronic system which integrates the actuators and the sensors, as well as the electronic command unit (equipment and software). The structure of the piloting system consists in expressing a global functional view (white box) of the system.

Operation safety studies must be applied when developing the system. The solution chosen must be justified and accompanied by the tracability of operation safety requirements on each of the three levels of knowledge of the system (global system, electronic piloting system and software).

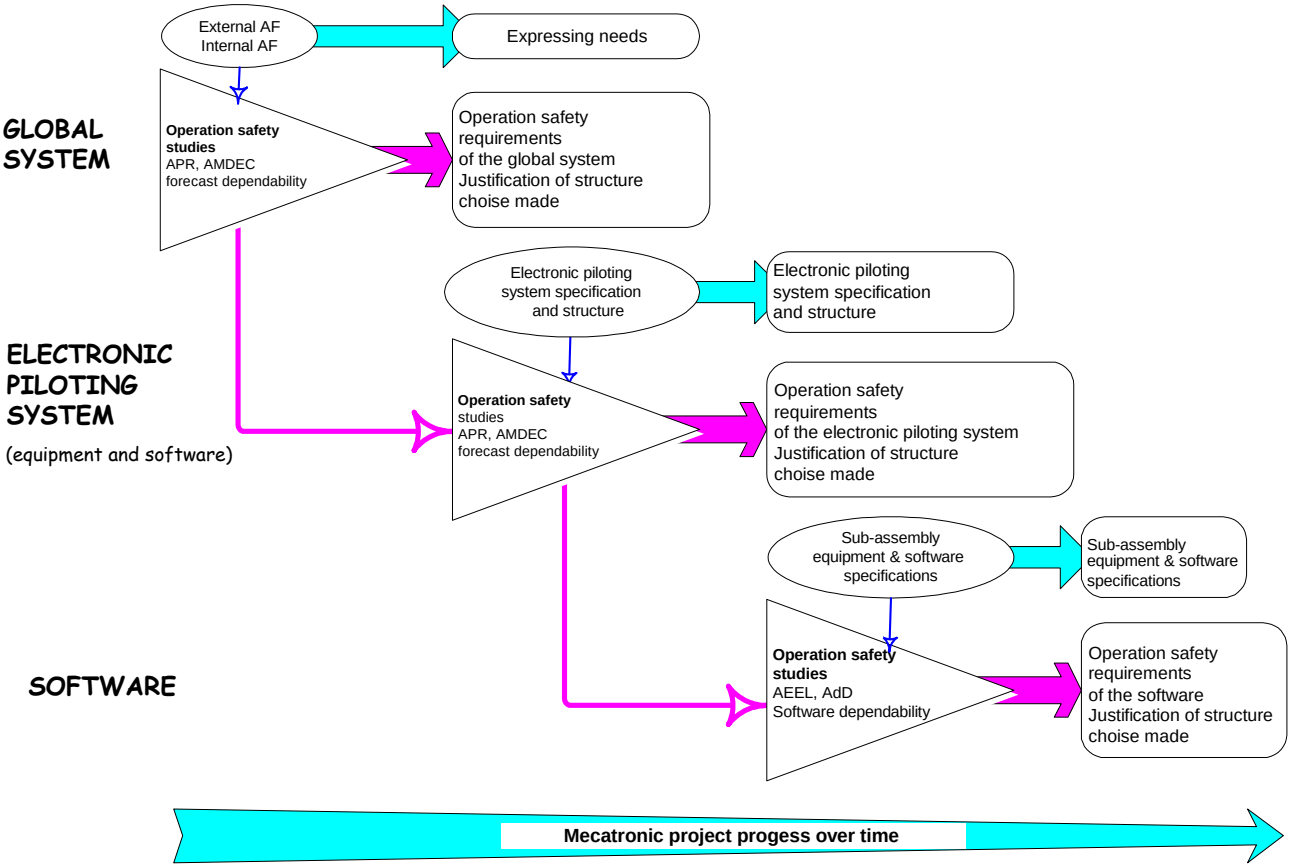


Figure 1.2 : Explicit operation safety process.

1.1.4.1 Establishing safety objectives

A safety objective for the global system may be allotted according to the return of experience with systems previously developed, based on expert judgement. The operation safety study must set a realistic objective. The objective for electronic systems is expressed by a rate of failure.

Allotting a safety objective for each function must be in keeping with the dependability objective expressed for the overall electronic system. Other complementary objectives may be necessary for maintenance actions.

1.1.4.2 Mecatronic system specification

Once the various events feared have been identified for the mecatronic system, the operation safety demands must be specified. The first step is to express OPERATION SAFETY ASSURANCE CRITERIA which will be used to establish confidence in the operation safety level of the system.
The second step is to define the OPERATION SAFETY MEANS which must be implemented to ensure control of operation safety when designing the electronic system.
The operation safety study of function and physical structures of the electronic system is composed of various activities. It is mainly accomplished by AEEL (analyse des effets des erreurs sur le logiciel – analysis of the effect of errors on the software) and by failure tree diagrams.
Using the software failure tree diagrams makes it possible to complete the AEEL, in order to warn against faults in design. The study is run on the software structure (specification and/or general design), in order to study the potential failure combinations which cause feared events in the software. Analysis of minimum cuts makes it possible to rank the critical elements of the software.

TYPES OF SOFTWARE ERRORS	CLASSES OF ERRORS USED
Calculation error	Evaluation of an incorrect equation, incorrect result to an operation
Algorithm error	Error in instruction sequencing, (un)conditional incorrect branching, incorrect processing loop
Error in task synchronisation	Incorrect synchronisation primitive type, unexpected synchronisation parameter
Error in data processed	Error in definition, error in initialisation, error in manipulation, modification of the value of data
Error in interfacing between procedures	Error in procedure instruction, error in procedure outlet, error in parameter transmission between procedures
Error in transferring data with the environment	Error in defining data, error in data transmission,, incorrect transfer periodicity

Table 1.2 : Example of software error typology.

1.2 Evaluating safety software

1.2.1 Specific characteristics of safety software

Software is an intellectual creating including programs, procedures, rules and all associated documents, related to implementation of the programmed system. Software is materialised by specifications, a code (program) and documentation.

Software development is often difficult to control. Moreover, software is rarely a finished product; it evolves from one version to another, within very short periods of time. It is a paradoxical product, which may become obsolete, but is not subject to wear. On the contrary, it is best when used frequently. Finally, software development is essentially devoted to product design and testing and little emphasis is placed on series production.

One of the most important characteristics of software is that it is a product with countless inputs and which processes combinations far greater than the brain capacity. As a consequence, software behaviour cannot be fully apprehended by man. It is therefore separated into different modules. Nevertheless, it remains difficult to fully control the complexity of the product.

1.2.2 Evaluating safety software

The problem raised by software evaluation is to obtain justified confidence in the software behaviour. The software is often analysed according to the method used for its development.

The evaluation is then based on a wide variety of criteria such as its structure, its development process, or the manner in which it was written, even though, in fact, only its behaviour should be evaluated. This is why it is rather difficult to distinguish between development methods and evaluation methods. These two types of methods increasingly overlap one another.

Finally, it is interesting to note that there are no specific methods for critical software. The methods used for critical software and those used for traditional software differ by the requirements of the standards. The major difference, in fact, resides in the budget and the time devoted.

Evaluating software may have highly varied significations. In general, two levels of evaluation are frequently distinguished : validation and verification.

1.3 Software safety requirements

Expressing software safety requirements, as well as the taking into account and follow-up of these requirements throughout the software development cycle, remains within the field of avant garde projects to date. Information concerning these requirements is not actually diffused to the general public and often remains limited to a circle of experts.

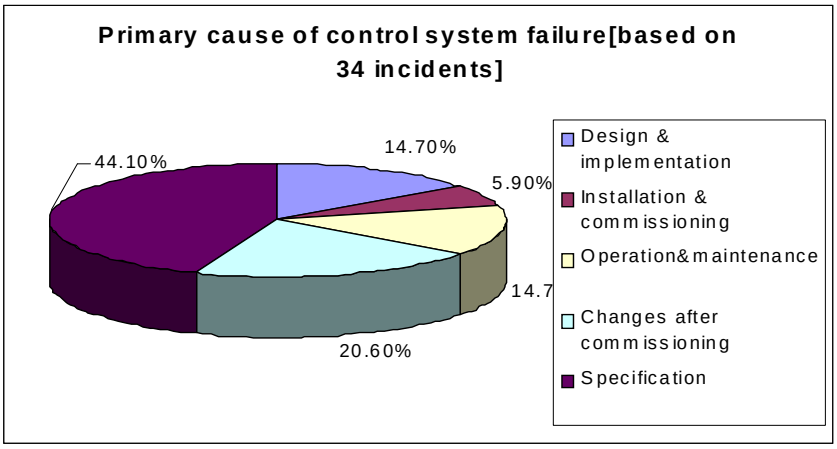
Work concerning software safety requirements has be initiated by organisations such as ISdF (Institut de Sûreté de Fonctionnement – Operation Safety Institute), INRS (Institut National de Recherche et de Sécurité – National Institute for Research and Safety), as well as by INRETS (Institut National de REcherche sur les Transports et leur Sécurité – National Institute for Research concerning Transport and Safety) within the framework of research projects such as CASCADE (Certification and Assessment of Safety Critical Application Development) and ACRUDA. In general, all of this work resembles the needs of avionics, nuclear and railway transport fields.

Thus, based on collected information concerning practices in the industrial environment in the field of software safety, mainly in the defence, transport and space sectors, and concerning use of work and reflections by European groups (PDCS and EWICS/TC7) and national groups (AFCET and ISdF), work done by IsdF reflection groups has led to the elaboration of two synthesis documents. An initial guide to elaborate the safety requirements for the software, for the provider, and a second guide to develop software with strict safety requirements, for the contracting party have thus been drafted.

2. SPECIFICATION AND VALIDATION OF SAFETY SOFTWARE

2.1 Specification requirements

A study run by HSE concerning primary causes of failures on a population of 34 accidents, shows that the main part (44.1%) is caused by poor specification.



Special attention must be paid to :

- **Adequation faults**. Adequation between the need recognised and the actual need must be ensured,
- **Over-specification**. This may lead to unnecessary restriction and exclude certain solutions,
- **Under-specification**. This may allow for a manoeuvre margin which is too wide concerning the chosen solutions, and may lead to unacceptable choices,
- **Unfortunate consequences of certain requests**. Impossible or non verifiable objectives should not be specified. ***Requests to apply standards or guides should only be made once their contributions and negative impacts have been carefully considered***,
- **The form**. It is recommended that enunciation remains precise, that styles "in keeping with the rules of the art" be avoided, that terminology be defined, that references from one document to another be avoided, that there be a constant concern for tracability and verification.

Requirements concerning the product and the processes, which may be applied to the software and its entities and auxiliary services, must be established by contract. A fair compromise must be made between contractual requirements which are necessarily severe, and the minimum freedom to be granted to the developer, i.e. which engages responsibility and motivation.

Formulating operation safety objectives must be **quantitative**, in terms of the rate or critical failures and/or **qualitative**, with a list of feared events, a qualitative analysis, the respect for specific procedures and regulations.
From the point of view of software output safety, feared events must be described perfectly. It is recommended that ***a preliminary list of accepted degraded operation modes be established as early as possible***.
It is highly inadvisable to simply ***formulate quantitative requirements for the software***.

2.2 Validation and specification methods

2.2.2 Specification methods

Three types of specification language may be distinguished : specification in ordinary language, semi-formal specification and formal specification.

Ordinary language is chosen as the specification language if it is usually used. It may prove to be ambiguous, contradictory and incomplete, since two major problems persist :

1. Difficulty of expression : man does not think with words only.
2. Difficulty of interpretation : a text need not be complex to be difficult to interpret. Specification may thus be interpreted differently depending on the definitions one possesses or those consider to be the definitions used by the drafter.

Informal specifications written in ordinary language are generally incomplete, incoherent, ambiguous, contradictory and erroneous; at best, errors introduced are discovered late in the life cycle of the software. As a consequence, it seems reasonable that they not be used for safety software.

Table 1.3 provides several specification methods and their main characteristics.

Contrary to ordinary language, specifications implemented by formal methods are not ambiguous, are precise, the semantics of notations is clearly defined. If one is familiar with the representation used, formal methods are a good means of communication and documentation.

In fact, formal methods are more than a tool for representation; they are also a technique for drafting specifications which restrains the designer to make abstractions and finally results in a better comprehension and modelisation of the specifications. It is sometimes even possible to make simulations.

Use of a formal method requires considerable investments in time and training. Formal methods are a significant step forward for the development and evaluation of critical software.

Table 1.4 provides a non exhaustive list of formal methods.

Name	Place in the Life cycle and Purpose of The method	Observations
RdP Réseau de Pétri Pétri Network	Specification Development	Method based on transition systems, using tokens and spaces. It makes it possible to demonstrate properties such as non-blocking, vivacity or equity of a set of co-operating processes. It is often used to specify parallelism and synchronisation.
Statecharts	Specification Development	Specification method based on transition systems.
SADT (Structured Analysis Design Technique)	Specification, design Development	Graphic specification method. It uses boxes to represent data or activities and arrows to represent flows between data or these activities. It is sometimes designated as a semi-formal design method and is often used in industry.
SART (Structured Analysis Real Time)	Specification Development	Real time extension proposed for the structured S.A. method of E. Yourdon and T. de Marco. One of the most widely used structured software analysis methods for real time applications.
Z	Specification, design Development	Formal specification language based on the Zermelo theory of sets. It makes it possible to express functional conditions of the problem to be translated into set notation.
LDS	Specification Development	Specification and functional description language. It is subject to a CCITT standard.
CCS	Specification Development	Formalism used to describe parallelism semantics. It is based on process algebra and remains very abstract and impossible to be used to make useful conclusions.
CSP	Specification Development	Presents the same characteristics as CCS.

Table 1.3 : Specification methods which contribute to evaluating software.

Name	Place in the Life cycle and Purpose of The method	Observations
VDM (Vienna definitio n method)	Specification, Design Development Static evaluation	The oldest and best established formal specification language. It is also a development method. It combines concrete notions such as types of data and abstract notions such as the theory of sets. Before – after predicates (pre and post conditions) where what does not change must be clarified, guides the refining of the specifications. Proof is required and written using a three value logic (True, False and Undefined). This particular logic does not simplify the establishment or the verification of proof. There is no mechanism to decompose or compose specifications or refinings. VDM has been chosen by the EEC, the English Standards Institution Committee and ISO to be used as the basis to develop a specification language standard.
B	Specification, Design Development Static evaluation	Formal method of specification based on the theory of sets and first order logic. Specifications are modelled using abstract machines. These machines, inspired by the object oriented design method, have three parts. The first describes the state and properties of the machine; the second specifies the operations which make it possible to modify the state; and the third records the composition connections with other machines. Specifications are developed using vertical iterations by refining and horizontal iterations by machine construction. Proof obligations are obtained by a substitution calculation. The B method is implemented by Atelier B, which strives to cover the entire development of software, from specifications and the production of proof obligations to code generation.
RAISE	Specification, Design Development Static evaluation	Set of tools which uses a specification formalism referred to as RSL and which combines the VDM method and process algebra.
CLEAN ROOM	Specification, Design Development Static and dynamic evaluation	Combines a formal method and a traditional software workshop. Specifications are written in PDL (Program Description Language) making it possible to define abstract machines. Development is done manually using refining and the generation of proof obligations. Finally, a series of statistic tests makes it possible to evaluate the dependability of the software

		developed.
FDM (Formal Development methodology)	Specification, Design Development Static evaluation	Combines a specification language (Ina jo) and an assertion drafting language (Ina Mod). It implements abstract machines, refinings justified by proof obligations. It has the support of an interactive demonstrator, ITP, which takes care of proof, but remains rather limited. FDM is certified by the US National Computer Security Centre for safety applications.

Table 1.4 : Formal methods which contribute to software evaluation.

2.2.3 Methods of validation

The first step is to make certain that software specifications are in compliance with user needs. This verification is relatively difficult to make since the user often expresses his needs in an informal, incomplete, imprecise or yet incoherent manner. This activity therefore mainly rests on the experience and the know-how of experts in the field.

The second level of evaluation corresponds to moving from specifications to the final code; the final code must be in compliance with the software specifications. This evaluation is in fact devoted to the software development process. Its success depends on the methods and tools issued from the software engineering.

The third level of evaluation, corresponding to moving from the final code to the software behaviour, consists in executing the final code to check the software behaviour. This level of evaluation is based on dynamic methods and is automatically controlled for the most part.

The fourth level of evaluation consists in making certain that the final code is in compliance with the user needs. For the same reasons as those expressed for the first level of evaluation, which are inherent to the nature of the user needs, this compliance is very difficult to demonstrate.

The fifth level of evaluation, corresponding to moving from specifications to software behaviour, consists in checking that the software behaviour is in compliance with what is described in the specifications. This activity was previously referred to as VERIFICATION. It is also mentioned in documentation as the answer to the question, “Have we built the software correctly”. To date, this evaluation may be done by tests for which the initial sets have been elaborated based on specifications.
Over a longer period of time and by the intermediary of more elaborate methods, this evaluation may be done by program synthesis techniques or specification simulation techniques.

Finally, the sixth level represents the total evaluation activity.

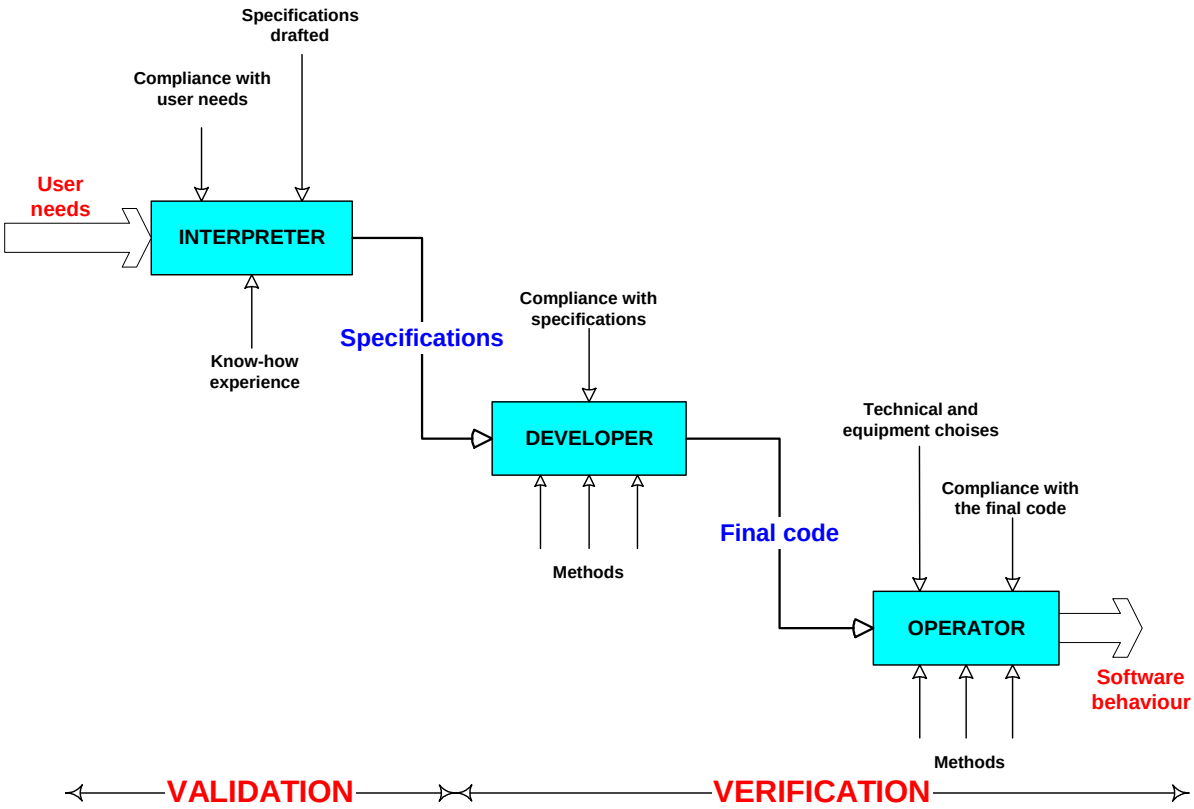


Figure 1.3 : Software evaluation activities.

2.3 CASE tools for specification and validation

The first CASE tools (Computer Aided Software Engineering) were developed as of 1985 in order to help software developers better understand and apply functional analysis methods and specification methods.

Despite the diversity of the tools available on the market today, the research we have done to identify those which enable safety software specification has remained sterile.

Appendix 4.2 contains the sheets of the 9 most renowned products.

Although oriented towards safety software development, the SCADE product (Safety Critical Application Development Environment) by VERILOG, does not have a sheet since it is not well adapted to small and medium sized applications.

2.4 Specification and validation procedure

The role played by specification for operating safety is to explicit “what to do?”, resulting from refining the specifications after functional analysis and preliminary risk analysis. It forms the interface between the analyst and the software designer, specifying the safety restrictions, such as execution time, inputs and outputs, the behaviour desired in case of failure, etc.

We contacted 7 French companies in order to report on the methods and tools used in the software development process, notably for critical software. Very few companies use specific methods. Among the companies which systematically implement software engineering methods and tools are companies involved in the automobile and nuclear industries.

It was very difficult for us to establish a procedure based on industrial practices concerning critical software specification. The interviews obtained with specialists from different horizons, enabled us to synthesise the following procedure for specification :

The specification phase is often based on the experience of the person in charge of drafting the specification.
It is necessary to :

- Provide a precise definition of the composition and role of the analysis and specification team
  - have final users intervene early
  - plan project reviews and their contents
  - have the client express his needs as extensively as possible
- Facilitate “client”-developer” communication
- For specification, in so far as possible, use an adequate method and possibly an adequate tool
  - (CASE tools ensure the unity of the dictionary and make it easier to avoid systematic transcription faults)
  - a good drawing is worth 1,000 words
- imagine being in the client’s position and adopt his logic and his manner of expressing himself
- use neutral vocabulary for both parties, client / specification person or team
- incite the client to enter into the developer logic, in order that he may formalise his need better
  - the client will thus express the needs he considers “evident” and therefore not necessary to be stated
- begin by making a functional analysis and a specification of the overall system
- make as complete a description as possible of the environment-operator-application interactions
- define the role of the operator
- take into account the application ergonomics : screens, alarm control, diagnostic, interventions for maintenance, etc.
- divide to rule better
  - wait for the right moment in the system description before dividing specification tasks among the members of a team
  - decrease the complexity by carefully chosen divisions

    - minimise the information exchange flows
    - do not decompose the system into more than three level, since complexity increases quickly and overall control may be lost
    - decompose critical functions into primitive functions. Impasses may thus be highlighted.
- in parallel, during specification, specify the manner in which to check objective expectations (acceptance files) and the means necessary to complete the verification
- take the referential into full consideration : standards, guides, technical documents, etc., before referring to them in the specifications
    - select elements which may be realised and measured in relation with the size of the application, the structure and culture of the company which develops the software
- validate the specifications by an internal audit type action done by a person / team other than that concerned by specification and development
- include the final user of the application

3. CONCLUSION

Faced with the fact that few software fault models exist, essentially design faults, the software operation safety procedure to be established must be based on the combined implementation of various complementary techniques.

Over and above the development process, it has been explained that organisation and management dimensions play an important role in safety software projects. The need to express safety requirements for the software, to take into account and follow-up these requirements throughout the development cycle is absolutely necessary. In addition to operation safety activities, the procedure must also include “quality” activities depending on the criticality of the software developed.

There is a great difference between software development practices and theoretical works which treat the subject.
It is difficult to elaborate an operation safety methodology for small and average sized companies. Organisational restriction, and more particularly the lack of operation safety culture, greatly complicate the elaboration of an operation safety process. There are no specific software safety specification tools for small and average sized applications.

Safety software specification is a very important phase in the life cycle. The procedure proposed highlights the prime importance of the role of communication between the person (team) doing the specification and the final user (client) and the necessity to use adapted methods and tools.

4. APPENDIXES

4.1 "SPECIFICATION METHODS" Sheets

AMDE LOGICIEL [Analysis of Failure Modes and their Effects]	**METH1**

PURPOSE

The purpose of the software AMDE is to identify likely software failure modes, possible causes for each mode and the effects on the system, on the one hand, and to control the failure risks of a software product, by implementing preventive or corrective actions, on the other hand.
The software AMDE is sometimes referred to as AEEL (Analysis of the Effects of Software Errors).

PLACE IN THE LIFE CYCLE

AMDE is a software design assistance method for systems for which dependability and/or safety is a main component. It may be used during the specification phase or during the software design phase.

NECESSARY RESOURCES

The software AMDE requires perfect knowledge of the software to be analysed. No specific tool need be used. An Excel table, or even a Word table are sufficient.

ADVANTAGES AND INCONVENIENCES

Before making an AMDE, the level of detail to be reached in the software tree chart must be defined. The AMDE is extremely efficient when centred on software components which way cause failures in the overall system. It makes it possible to optimise verifications by differentiating the levels of tests run depending on the criticality of the software components.
For complex systems with a large number of components, it is preferable to make analyses at several levels (identification of the system parts to be make dependable, restriction of the analysis to certain functions and to certain software failures, such as blocking tasks, for example).

REPRESENTATION OR THEORIES USED

AMDE are usually realised by tables presented as columns containing information, for each elementary software component studied (failure mode, causes of failures, effects of failures, means of detecting failures).

PRODUCTS

AMDEC-SOFIA, AMDEC-PRO.

FOR FURTHER INFORMATION

- CEI 812-1985 : Techniques d'analyse de la fiabilité des systèmes/Procédures d'analyses des modes de défaillances et de leurs effets. (System dependability analysis techniques / Analysis procedures of failure modes and their effects.)
- A. Villemeur-1988 : Sûreté de fonctionnement des systèmes industriels (Operating safety of industrial systems) - Eyrolles.
- C. Hourtolle-1987 : Conception de logiciels sûrs de fonctionnement, Analyse de la sécurité des logiciels (Designing dependable operating software, Analysis of software safety) – Doctorate thesis in computer science at LAAS-CNRS.

FAILURE CHART | METH2

PURPOSE

The purpose of the Failure chart is to find combinations of events, based on a feared event, which lead to occurrence of this event. The failure chart makes it possible to identify fairly rapidly all risks to the execution environment of software (equipment, software, even human).

PLACE IN THE LIFE CYCLE

The failure chart is a method which may be used during the specification and design phases of software.

NECESSARY RESOURCES

The principles of failure charts are relatively simple and do not require specific tools. However, it is recommended that a tool be used to determine certain characteristics of the chart, such as minimum cuts.

ADVANTAGES AND INCONVENIENCES

The purpose is to rapidly identify the parts of the software which combine events to be avoided by the system.

In order not to burden the analysis, it is advisable to define the level to be reached by the analysis beforehand. Only chart branches corresponding to software failures are detailed to the extent that they may be allotted to a software component.

The failure chart is not adapted for the representation of dynamic events and leads to repetitions if the feared events are not well separated from one another.

REPRESENTATION OR THEORIES USED

Based on a list of feared events, the failure chart consists in decomposing each event (chart root) by successive levels of events, connected by AND, OR, etc. type logical operators, with precise symbolism.

On the basis of the failure chart thus established, particular techniques (preferably requiring the implementation of tools) are used in order to identify the weak points of the software, such as reducing the chart to minimum cuts.

PRODUCTS

FOR FURTHER INFORMATION

- *A. Villemeur-1988 : Sûreté de fonctionnement des systèmes industriels (Operating safety of industrial systems)- Eyrolles..*
- *C. Hourtolle-1987 : Conception de logiciels sûrs de fonctionnement, Analyse de la sécurité des logiciels (Designing dependable operating software, Analysis of software safety) – Doctorate thesis in computer science at LAAS-CNRS.*
- *N. Limnios-1991 : Arbres de défaillances (Failure charts) - Hermès.*

B LANGUAGE	METH3

PURPOSE

The purpose of the B language is formal development of software components and the supply of correction verification means in relation to the specifications.

PLACE IN THE LIFE CYCLE

The B language covers all specification, design and realisation phases of the software, the ADA or C code may be generated automatically.

RESOURCES NECESSARY

Implementation of the B language for industrial development of software requires a set of assistance tools, notably for the verification (proof) phases and code generation phases.

ADVANTAGES AND INCONVENIENCES

The B language is a homogeneous language used from the specification phase to the realisation phase, allowing for formal verifications.
Mathematical demonstrations, referred to as "proof", carried out while using the B language, guarantee the internal coherence of the various modules, the coherence of their interactions, as well as the validity of the realisation modules in relation to the specification modules.
Failure of a mathematical demonstration highlights a lack or incoherence in the specifications.
The B language is well adapted to realising automatic controls, control-commands, or to the establishment and verification of critical algorithms. However, it is not well adapted to realise parallel or critical real time software systems. Several weeks of training are required to become operational.

REPRESENTATION OR THEORIES USED

The B language is a formal specification language enabling modelisation of a system (with software parts) by abstract machines. It is a modular language which offers data encapsulation, ranking and decomposition mechanisms. The properties of the modelled system are expressed by mathematical formulas. The formal semantics of the B language guarantee that these properties are met when the software is installed.

PRODUCTS

Atelier B (STERIA).

FOR FURTHER INFORMATION

- J.F. Monin-1996 : Comprendre les méthodes formelles - Panorama et outils logiques (Understanding formal methods – Panorama and logical tools)- Masson.
- S. Taouil-Traverson-1997 : Stratégie d'intégration de la méthode B dans la construction de logiciel critique (Method B integration strategy in critical software construction) – Doctorate Thesis in Computer Science - Télécom. Paris - July 1997.

LUSTRE LANGUAGE	METH4

PURPOSE

The purpose of the Lustre language is to develop critical real time software, in fields such as control-command, automatic control and signal treatment.

PLACE IN THE LIFE CYCLE

The Lustre language covers specification and design phases. It is also possible to automatically generate the C code, from a "Lustre" description.

RESOURCES NECESSARY

Implementing the Lustre language for industrial development require use of a CASE tool (graphic editor, simulator, C code generator).

ADVANTAGES AND INCONVENIENCES

Lustre is a synchronous data flow language. A "Lustre" program may be graphically represented by a network of operators acting in parallel with the rhythm of inputs. The Lustre execution model guarantees functional determinism and ensures a limited execution time.
Lustre facilitates the manipulation of time phenomena for the user. The graphic aspect of the language reinforces readability of the descriptions.
It is possible to generate the directly compilable C code from a Lustre description. This description may be simulated, thus making it possible to validate the functional behaviour.
The mathematical semantics of the language makes it possible to formally check the properties.
The Lustre language is adapted to automatic controls in general (equation or function block representation). The Lustre language is not well adapted for asynchronous type applications (communication protocol type applications). It is best to be familiar with equation and functional block representations to implement the Lustre language.

REPRESENTATION OR THEORIES USED

The Lustre language has high level language properties. Modularity and ranking concepts are part of the language and concern both data and processing. Completeness and coherence are guaranteed by the semantics of the language (verification of the causality principle (the output of a program cannot depend on future inputs), detection of cyclical definition, etc.).

PRODUCTS

SCADE (Verilog).

FOR FURTHER INFORMATION

- N. Halbwachs-1991 : Programmation et vérification des systèmes réactifs - le langage Lustre (Programming and verification of reactive systems – the Lustre language) - Techniques et Sciences Informatiques (Computer Science and Techniques) - Vol.10 n°2 - 1991.
- C. Dubois-1995 : Approche synchrone et logiciels critiques pour l'automatique (Synchronous approach and critical software for automatic control) - REE n°1 - June 1995.

PETRI NETWORK	METH5

PURPOSE

The purpose of a Petri network is the dynamic description of software behaviour, in order to check the properties of completeness and coherence, etc. The Petri network also makes it possible to test the degraded modes of the software, as well as the efficiency of the fault tolerance techniques.

PLACE IN THE LIFE CYCLE

Behavioural modelisation by the Petri networks is a precious help in specifying and designing, aside from the verification it enables to accomplish. It is used at the end of static specification and makes it possible to refine this phase.

RESOURCES NECESSARY

The interest of dynamic specification lies in its simulation, which may be interactive and more exhaustive (all behaviour of the model are simulated and memorised in a state graph of the model). This simulation requires implementation of powerful computer tools.

ADVANTAGES AND INCONVENIENCES

Within the framework of malfunction analysis, verification is based on the model form and makes it possible to highlight incoherent behaviour or behaviour which blocks, such as non-reinitialisation, non-determinism, dead code (transitions which may not be drawn), blocked states.
It is one of the methods most often used to specify parallelism and synchronisation.
Often used to design models, it fails in the specification of complex systems. In fact, the networks become to large and are difficult to use.
Within the framework of model verification in relation to needs, specific invariable logic type properties of the model are verified, as is coherence of the actual behaviour in comparison with the expected behaviour, and calculation of global properties.
Within the framework of time evaluations, certain products provide an automatic simulation mode referred to as "stochastique", in which transitions are drawn at random dates, depending on the law of distribution associated with them. This makes it possible to evaluate certain quantitative characteristics of the model, such as its performance (average response time, etc.), as well as its dependability (average frequency of breakdown occurrence).
Within the framework of test generation, the model becomes a reference according to which the system is to be validated. A tool allows for the automatic generation of validation test scenarios, which are representative, based on specifications.
Implementation of Petri networks requires a major investment in terms of tools.

REPRESENTATION OR THEORIES USED

The method of representation by Petri networks is based on transition systems by using tokens and spaces.

PRODUCTS

FOR FURTHER INFORMATION

- C. Hourtolle-1987 : Conception de logiciels sûrs de fonctionnement, Analyse de la sécurité des logiciels (Designing dependable operating software, Analysis of software safety) – Doctorate thesis in computer science at LAAS-CNRS.

SA [Structured Analysis]	**METH6**

PURPOSE

The purpose of the SA method is the static specification of the software. This method was defined by E. Yourdon and T. de Marco in 1978-1979. It allows for structured analysis of software by successive refining of processing until all are described in terms of logical data flows.

PLACE IN THE LIFE CYCLE

The SA method covers the specification phase.

RESOURCES NECESSARY

Implementation of the SA method requires no computer tools. However, for relatively large projects, it is recommended to implement CASE tools, especially since these tools supply assistance for construction and model validation.

ADVANTAGES AND INCONVENIENCES

The SA method is especially well adapted for information exchange and order transmission problems. It is used by many software engineering workshops. Due to its concepts, it concentrates on the software functional specification phase.

It is not well adapted for representation of the dynamic behaviour of software. In fact, it presents a static view of the various functions which the software to be realised must assume.

REPRESENTATION OR THEORIES USED

The SA method language is composed of graphic tools and/or textual tools.

The graphic tools implement DFD (Data Flow Diagrams), which are interconnection "process" networks or function connected by data which circulates.

The textual tool is comprised of a DD (Data Dictionary). This is created, is interpreted at the same time as the data flow diagrams. The DD group together the semantics and structure of all data present in the system. The data dictionary includes the description of the data flows.

The graphic and/or textual tool implements DSD (Data Structure Diagrams) and PSPEC (Process Specifications). The description of complex data requires the creation of DSD. Information contained in the DSD may be textual or graphic.

The last refining levels reveal elementary processes referred to as PSPEC function primitives. These may be expressed by abstract algorithms, decision charts, decision tables, Michael Jackson diagrams or Nassi Shneiderman diagrams.

PRODUCTS

Teamwork (CAYENNE Software).

FOR FURTHER INFORMATION

- *P. Jaulent-1990 : Génie logiciel (Software engineering) - Armand Colin.*

SADT [Structured Analysis and Design Technique]	**METH7**

PURPOSE

The SADT method of analysis is a multiple disciplinary language which strives to encourage communication between users and designers. Designed according to simple concepts and based on graphic and textual formalism which is easy to learn, SADT makes it possible to model the problem raised before attempting to expose a solution, on the one hand, and, on the other hand, to ensure efficient communication between the various people concerned by the problem to be solved.

PLACE IN THE LIFE CYCLE

The SADT method cover the system specification and design phases.

RESOURCES NECESSARY

Implementing the SADT method requires no computer tools. However, for relatively large projects, it is advisable to implement CASE tools, since these tools provide assistance for model construction and validation.

ADVANTAGES AND INCONVENIENCES

The SADT method is a simple method, often used in industry, especially to decompose a system into sub-systems. Its power of expression nevertheless remains weak. This method offers good readability, due to the clear presentation of its graphics, the limited number of its basic concepts making it easy to learn, its aptitude to communicate in a non computer language, the hierarchical and modular structure of the model obtained.

However, SADT is missing certain elements, such as specification from the point of view of performance and temporal restrictions, coherence between actigrams and questionable datagrams, and it presents a very limited dynamic point of view.

REPRESENTATION OR THEORIES USED

SADT is a graphic method. It implements boxes to represent data or activities and arrows to represent flows between this data or these activities. A SADT model is composed of a set of hierarchically arranged diagrams. At the most general level, there is a context diagram which shows the sources and destinations of the various information arriving to or leaving the "box to be analysed". Each element of the diagram may be decomposed into another diagram. Each diagram of lower levels provides a limited number of details concerning a well defined subject.

PRODUCTS

ENVISION (CASE France).

FOR FURTHER INFORMATION

- *P. Jaulent-1990 : Génie logiciel (Software engineering)- Armand Colin.*

SART [Structured Analysis Real Time]	**METH8**

PURPOSE

The SART method is a real time extension of the SA structured analysis method. This extension offered by the Ward-Mellor and Hatley-Pirbhai schools partially solves the problem of dynamic and factual modelisation of a real time application, thus making it possible to elaborate a model while considering the "response to events", from the "action-perception area" of the system.
The SART method helps to specify the software to be implemented on two types of real time systems, combination systems and sequential systems.

PLACE IN THE LIFE CYCLE

The SART method covers the specification phase.

RESOURCES NECESSARY

Implementation of the SART method requires use of CASE tools, since these tools provide assistance for model construction and validation.

ADVANTAGES AND INCONVENIENCES

The real time extensions offered by the SART method to model the dynamics of software with high real time restrictions seems well adapted. However, the definition and the syntax rigour of the method remain somewhat ambiguous (the rules for data connection and circulation between the processes are not clearly established, the DFED, the PSPEC and the DD are too dependent on human interpretation).
Finally, although several CASE tools offer partial simulation of the system based on the analysis model, most actually only verify the "balancing" of data and the exactitude of the syntax.

REPRESENTATION OR THEORIES USED

The SART structured analysis language is composed of :
- SA structured analysis tools
The real time extension offered to the SA method by the Ward-Mellor and Hatley-Pirbhai schools includes :
- a graphic tool which is the control flow diagram, referred to as CFD (Control Flow Diagrams),
- a textual tool which is the DD dictionary of structured analysis data,
- two graphic and/or textual tools which are the control specifications referred to as CSPEC (Control Specifications), and the timing specifications, referred to as TSPEC in the Hatley-Pirbhai notation.

PRODUCTS

*AXIOM/SYS (*ASTREE PERFORMANCE), ENVISION (*CASE France), STP-SE (AONIX), TEAMWORK (CAYENNE SOFTWARE).*

FOR FURTHER INFORMATION

- *P. Jaulent - 1990 : Génie logiciel (Software engineering) - Armand Colin.*
- *D.J. Hatley, I.A. Pirbhai - 1991 : Stratégies de spécification des systèmes temps réel (Specification strategies for real time systems) (SA-RT) - Masson.*

STATE CHARTS	METH9

PURPOSE

The purpose of the State chart method is to describe and validate the behaviour of reactive systems.

PLACE IN THE LIFE CYCLE

State charts are implemented in the specification and design phases for reactive systems.

RESOURCES NECESSARY

Implementation of State charts requires knowledge of client requirements, which are analysed to define the system, its environment and its functions.
Implementation of State charts requires knowledge of the formalism of the finished machines.
Implementation of a tool is necessary for efficient processing of state charts (Statemate).

ADVANTAGES AND INCONVENIENCES

The "State chart" method is very efficient to express the behaviour of a system.
"State chart" simulation makes it possible to obtain non ambiguous specifications which may be validated by running it.
However, the "State chart" system may become relatively heavy to implement, for complex systems.
Implementation of "State charts" requires a rather long training period.

REPRESENTATION OR THEORIES USED

The "State chart" method is issued from work accomplished by David Harel in the 1980's.
The system is described according to 3 views :

1- Specification of the structure (module-charts) which describe the logical and physical modules, as well as the environment,
2- Specification of activities (activity-charts) including a data flow part and a control part,
3- Specification of the control (state charts).

The "State chart" is an extension of the diagram at a finished state. It makes it possible to represent a hierarchical and structured description of simultaneous actions or activities, complex transition conditions. The "State chart" makes it possible to :
- refine by detailing a state using one or several automatic controls. When several automatic controls are used, there is parallel evolution,
- explicit or default specification of an initial state for a set of automatic controls, and multiple synchronisation,
- association of time restrictions and states.

PRODUCTS

Statemate (i-Logix).

FOR FURTHER INFORMATION

- J.J. Valloton-1991 : Statemate : un outil pour la spécification des systèmes réactifs, (Statemate : a tool for reactive system specification) Génie logiciel (Software engineering) n°25-December 1991.

4.2 "SPECIFICATION TOOLS" Sheets

ATELIER B	ATEL1
METHODS	
Set of tools dedicated to development according to the B method for critical safety systems. Created in collaboration with GEC-ALSTHOM, RATP, SNCF INRETS, Atelier B may be used with the B formal method. It notably integrates the following tools : a syntax and type controller, a generator of proof obligations, a demonstrator, "C" and "ADA" translators. The tools are activated from a graphic interface and automatically control the development phases.	
DESIGNER	
STERIA (F), GEC-ALSTHOM Transport (F)	
INTERFACE SOFTWARE PACKAGES	
INTERLEAF, LATEX, FRAME MAKER, WORD, VCG	
PLATFORMS	
UNIX (HP-UX-LINUX-SOLARIS)	
DISTRIBUTORS	
STERIA *12, rue Paul Dautier, BP58* *78142 Vélizy-Villacoublay* *Tel. 01.34.88.60.00* *Fax. 01.34.88.62.00* *atelierb.aix@steria.fr*	
PRICE	
Experimentation version : 40 KF *Basic version (1 licence) : 130 KF*	
OBSERVATIONS	
40 sites installed in France. *First installation 03/1995* *Most recent version 3.0 (12/96)*	

AXIOM/SYS	ATEL2
METHODS	
Analysis, specification of real time systems or software with SART and modelisation of equipment structure. Uses the D.J. Hatley I.A. Pirbhai method.	
DESIGNER	
STG (US)	
INTERFACE SOFTWARE PACKAGES	
PLATFORMS	
MICROSOFT (Windows 3.xx, Windows 95, Windows NT Workstation)	
DISTRIBUTORS	
ASTREE PERFORMANCE 92100 Boulogne Billancourt Tel. 01.47.02.63.09 Fax. 01.47.02.63.22 www.astree-performance.fr	
PRICE	
10 KF to 25 KF	
OBSERVATIONS	
800 in the world. Most recent version 5.0D (01/1997)	

DESIGN IDEF	ATEL3
METHODS	
BPR (Business Process Reengineering) tool, which may be used with the methods IDEF0 (SADT) and IDEF1X, making it possible to model complex processes with a rigorous, hierarchical procedure. Offers a bridge towards "Workflow" and simulation.	
DESIGNER	
METASOFTWARE (US)	
INTERFACE SOFTWARE PACKAGES	
VISUAL WORKFLOW	
PLATFORMS	
APPLE (Mac/OS), MICROSOFT (Windows 3.xx, Windows 95, Windows NT)	
DISTRIBUTORS	
AONIX *92247 Malakoff Cedex* *Tel. 01.41.48.10.00* *Fax. 01.41.48.10.20* *foulquier@aonix.fr*	
PRICE	
User rights : 30 KF	
OBSERVATIONS	
400 in France. *1200 in the world.* *Most recent update : 12/97*	

ENVISION	ATEL4
METHODS	
Modelisation and simulation tool, multiple method, multiple user. May be used with the software engineering methods SART, SD, SADT, UML, real time, entity-relation.	
DESIGNER	
FUTURE TECH. SYSTEMS (US)	
INTERFACE SOFTWARE PACKAGES	
All software under Windows, Excel, MS Project, WORD.	
PLATFORMS	
MICROSOFT (Windows 3.xx, Windows 95, Windows NT Server, Windows NT Workstation), NetWare	
DISTRIBUTORS	
CASE France *75015 Paris* *Tel. 01.45.54.31.28* *Fax. 01.45.54.29.98* jacquiot@artinternet.fr	
PRICE	
User rights : 10 KF to 59 KF.	
OBSERVATIONS	
650 in France *40 in GB* *Most recent version 5.2 (10/96)*	

OBJECT GEODE	ATEL5

METHODS

Set of tools dedicated to the analysis, design and validation of real time and distributed applications. It allows for simulation, code generation and testing. OBJECT GEODE may be used with coherent integration to complementary object oriented approaches based on standards such as OMT (Object Modelling Technique) by Rumbaugh, SDL (Specification and Description Language) and MSC (Message Sequence Chart). The tool has graphic editors, a simulator operating in an interactive, random or exhaustive mode, a C/C++ code generator for the real time operating systems on the market.

DESIGNER

VERILOG (F)

INTERFACE SOFTWARE PACKAGES

PLATFORMS

UNIX (AIX, HP, UX, SOLARIS, SUN/OS)

DISTRIBUTORS

VERILOG
92220 Bagneux
Tel. 01.45.36.57.00
Fax. 01.46.65.77.38

PRICE

As of 50 KF

OBSERVATIONS

1126 in France
2169 in Europe
2623 in the world

ORCHIS	ATEL6
METHODS	
Functional specification tool which responds to the standard. IDEFO. Offers a graphic editor for actigrams, glossaries, a rule check and a document generator for the author-reader cycle.	
DESIGNER	
TNI (F)	
INTERFACE SOFTWARE PACKAGES	
PLATFORMS	
MICROSOFT (MS/DOS), UNIX(AIX, HP, UX, SOLARIS)	
DISTRIBUTORS	
TNI *54600 Villiers Les Nancy* *Tel. 03.83.44.01.41* franck.corbier@tni.fr	
PRICE	
5 KF and 10 KF	
OBSERVATIONS	
Most recent version 2.12 (7/95).	

STP-SE	ATEL7

METHODS

Analysis and design environment implementing the De Marco, Yourdon, Hatley, Pirbhai et Constantine SART and SD methods. This environment offers graphic and table editors, code generation and documentation functions.

DESIGNER

AONIX (US)

INTERFACE SOFTWARE PACKAGES

INTERLEAF, FRAMEMAKER, TOOLTALK, SOFTBENCH, RTM DE MARCONI, CLEAR CASE

PLATFORMS

MICROSOFT (Windows NT Server, Windows NT Workstation), UNIX (AIX, DIGITAL UNIX, HP , UX, IRIX, SOLARIS)

DISTRIBUTORS

AONIX
92247 Malakoff Cedex
Tel. 01.41.48.10.00
Fax. 01.41.48.10.20
foulquier@aonix.fr

PRICE

Under Windows NT : 20 KF
Under UNIX : 35 KF

OBSERVATIONS

880 in France
12000 in the world
Most recent version : 6.4 (12/1997)

SYNCCHARTS/ESTEREL	ATEL8

METHODS

Graphic software for the specification and display of software applications or reactive equipment. It associates a synchronous language with graphic formalism which allows for the description of complex applications without programming (real time systems, communication protocols, etc.). It includes a graphic editor a synchronous language compiler ESTEREL v. 5, a graphic verification tool and a multiple target code generator.

DESIGNER

SIMULOG, CMA

INTERFACE SOFTWARE PACKAGES

PLATFORMS

UNIX (AIX, DIGITAL, UNIX, HP-UX, IRIX, SOLARIS)

DISTRIBUTORS

SIMULOG
78286 Guyancourt Cedex
Tel. 01.30.12.27.00
Fax. 01.30.12.27.27

PRICE

OBSERVATIONS

2 in France
2 in the world

TEAMWORK	ATEL9
METHODS	
Software engineering workshop for analysis, design, code generation and code simulation of real time systems.	
DESIGNER	
CAYENNE SOFTWARE (US)	
INTERFACE SOFTWARE PACKAGES	
SOFTBENCH, DOORS, RTM	
PLATFORMS	
DIGITAL VAX (Open VMS), MICROSOFT (Windows NT Server, Windows NT Workstation), OS2 Warp, UNIX (AIX, DIGITAL UNIX, HP-UX, IRIX, SOLARIS)	
DISTRIBUTORS	
CAYENNE SOFTWARE *92108 Boulogne Billancourt Cedex* *Tel. 01.41.10.25.50* *Fax. 01.41.10.80.11*	
PRICE	
User rights : 50 KF to 300 KF	
OBSERVATIONS	
39000 in the world.	

4.3 Bibliography

4.3.1 Standard references

NF EN 954-1
Sécurité des
machines. Parties des systèmes de commande relatives à la sécurité. Partie 1 : Principes généraux de conception. (Machine safety. Parts of command systems concerning safety. Part 1 : General design principles). 1997-02.

NF EN 1050
Sécurité des machines. Principes pour l'appréciation du risque. (Machine safety. Risk appreciation principles). 1997-01.

IEC 300-3-1
Gestion de la sûreté de fonctionnement. Partie 3 : Guide d'application. Section 1 : Techniques de la sûreté de fonctionnement, Guide méthodologique. (Operating safety management. Part 3 : Application guide. Section 1 : Operating safety techniques, Methodological guide). - 1991.

IEC 812-1985
Techniques d'analyse de la fiabilité des systèmes. Procédure d'analyse des modes de défaillance et de leurs effets (AMDE). (System dependability analysis techniques. Analysis procedure for failure modes and their effects). 1985-12.

IEC 61508-1
Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité. Partie 1 : Prescriptions générales. Première édition. (Functional safety of electric/electronic/programmable electronic systems concerning safety. Part 1 : General instructions. First issue.) 1998-12.

IEC 61508-3
Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité. Partie 3 : Prescriptions concernant les logiciels. (Functional safety of electric/electronic/programmable electronic systems concerning safety. Part 3 : Instructions concerning software). 1998-11.

4.3.2 References and technical documents

ARAGO 15
Informatique tolérante aux fautes. Observatoire Français des Techniques Avancées. (Fault tolerant computers. French Observatory for Advanced Techniques). Masson 1994.

[CAL90]
Spécification et conception des systèmes, une méthodologie. (System specification and design, a methodology). J.P. Calvez. Masson 1990.

CASCADE-1
Generalised Assessment Method (GAM). Part1 : Rules. Esprit 9032 Cascade - 1997.

CASCADE-2
Generalised Assessment Method (GAM). Part2 : Guidelines. Esprit 9032 Cascade - 1997.

CASCADE-3
Generalised Assessment Method (GAM). Part3 : Examples. Esprit 9032 Cascade - 1997.

[DAR96]
Méthodes contribuant à l'évaluation des logiciels de sécurité : Etude bibliographique. (Methods contributing to the evaluation of safety software : Bibliographical study). M. Darricau. INRETS 1996.

M. DARRICAU
Acquisition et représentation des connaissances impliquées dans les spécifications de logiciels de sécurité : Application au système de pilotage automatique (Acquisition and representation of knowledge implied in safety software specifications : Application of the automatic piloting system) - INRETS 1997.

EWICS-TC7
Guidelines for the use of Programmable Logic Controllers in Safety-related Systems. EWICS TC7 - Version 13 1997.

T. FORSE
Qualimétrie des systèmes complexes. Mesure de la qualité du logiciel. (Quality measurement of complex systems. Measuring software quality). Les Editions d'Organisation 1989.

J.C. GEFFROY
Sûreté de fonctionnement des systèmes informatiques. (Operating safety of computer systems) InterEditions 1998.

D.J. HATLEY
Stratégies de spécification des systèmes temps réel (Specification strategy for real time systems) (SA-RT). Masson 1991.

[HEN96]
Méthodologie de développement des systèmes électroniques embarqués automobiles, matériels et logiciels, sûrs de fonctionnement. (Development methodology for electronic systems installed in automobiles, equipment and software, operation dependable). V. Hénault. Doctorate Thesis, Ecole Doctorale Sciences pour l'ingénieur de Nantes - 1996.

C. HOURTOLLE
Conception de logiciels sûrs de fonctionnement : Analyse de la sécurité des logiciels, mécanismes de décision pour la programmation N-Versions. (Designing dependable operating software : analysis of software safety, decision mechanisms for N-Version programming) Doctorate Thesis, Laboratoire d'Automatique et d'Analyse des Systèmes (LAAS) - 1987.

[PAG80]

ISdF-14/92
Guide pour le développement de logiciels à hautes exigences de sécurité. (Guide for the development of high safety requirement software). ISdF project n°14/92.

ITSEC
Information Technology Security Evaluation Criteria (ITSEC). Provisional Harmonized Criteria - June 1991.

ITSEM
Information Technology Security Evaluation Manual (ITSEM). Provisional Harmonized Methodology - September 1993.

P. JAULENT
Génie logiciel, les méthodes. (Software engineering, methods). Armand Colin 1990.

J.C. LAPRIE
Sûreté de fonctionnement des systèmes informatiques, matériels et logiciels (Operating safety of computer systems, equipment and software) - J.C. Laprie, B. Courboie, M.C. Gaudel, D. Powell - 1989.

[LAP95]
Guide de la sûreté de fonctionnement. (Guide to operating safety). J.C. Laprie. Cépaduès Editions 1995.

D. LIGHTFOOT
La spécification formelle avec Z. (Formal specification with Z). Technea 1994.

N. LIMNIOS
Arbres de défaillances. (Failure charts) Hermès 1991.

J.F. MONIN
Comprendre les méthodes formelles, Panorama et outils logiques. (Understanding formal methods, Panorama and logical tools). Masson 1996.

C. NICOLAS
Etat de l'art sur les méthodes et les outils de validation des logiciels. (State of the art concerning software methods and validation tools) MASI-EDF 93.14 - 1993.

J.P. PEREZ
Systèmes temps réel : méthodes de spécification et de conception. (Real time systems : methods for specification and design) Dunod 1990.

C. SOURISSE
La sécurité des machines automatisées. Tome 1 : Notions de base - Réglementation - Normes - Techniques de prévention. (The safety of automatic controlled machines. Volume 1 : Basic notions – Regulations – Standards – Techniques of prevention). Groupe Schneider 1996.

C. SOURISSE
La sécurité des machines automatisées. Tome 2 : Techniques et moyens de prévention opératifs - Systèmes de commande - Utilisation des machines. (The safety of automatic controlled machines. Volume 2 : Techniques and means of operational prevention – Command systems – Use of machines). Groupe Schneider 1997.

S. TAOUIL-TRAVERSON
Stratégie d'intégration de la méthode B dans la construction de logiciel critique. (B method integration strategy in the construction of critical software). Doctorate Thesis, Ecole Nationale Supérieure des Télécommunications - July 1997.