Dependability engineering of complex computing systems

Mohamed Kaâniche, Jean-Claude Laprie, Jean-Paul Blanquart

Published in the Proceedings of the 6th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS-2000), Tokyo, Japan, 2000, pp. 36-46

Dependability Engineering of Complex Computing Systems

Mohamed Kaâniche[1], Jean-Claude Laprie[1], Jean-Paul Blanquart[2]

[1]*LAAS-CNRS / LIS*
7 av. du Colonel Roche
31077 Toulouse Cedex 4 — France
{kaaniche, laprie}@laas.fr

[2]*ASTRIUM / LIS*
31 rue des Cosmonautes
31402 Toulouse Cedex 4 — France
jean-paul.blanquar@astrium-space.com

Abstract

This paper presents a development model focused on the production of dependable systems. Three classes of processes are distinguished: 1) the system creation process which builds on the classical development steps (requirements, design, realization, integration); 2) dependability processes (i.e., fault prevention, fault tolerance, fault removal and fault forecasting); and 3) other supporting processes such as quality assurance and certification. The proposed approach relies on the identification of basic activities for the system creation process and for the dependability processes, and then on the analysis of the interactions among the activities of each process and with the other processes. Finally, to support the development of dependable systems, we define for each system creation activity, a checklist that specifies the key issues related to fault prevention, fault tolerance, fault removal, and fault forecasting, that need to be addressed.

1. Introduction

Design faults are generally recognized as being the current bottleneck for dependability in critical applications. As design faults have their origin in the development process, this leads naturally to pay attention to development models. Conventional development models, either for hardware or for software, do not explicitly incorporate all the activities needed for the production of dependable systems. Indeed, hardware development models traditionally incorporate reliability evaluation (see e.g., [5]), but pay less attention to verification and fault tolerance. On the other hand, traditional software development models (waterfall, spiral, etc.) incorporate verification and validation activities but do not mention reliability evaluation or fault tolerance.

Designing a dependable system that is able to deliver critical services with a high level of confidence is not an easy task. On the one hand, one has to face the increasing trend in the complexity of computer based critical applications that is related to the evolution towards large scale and distributed architectures. On the other hand, the diversity of the classes of faults at the origin of system failures (be they accidental or intentionally malicious) and of their consequences and severity, requires the implementation and integration of multiple overlapping and complex fault tolerance mechanisms. Therefore, there is a need for a systematic and structured design framework that integrates dependability concerns at the very early stages of the development process. This is especially important for systems that have to satisfy several, and sometimes conflicting, dependability objectives. Such a framework is also useful to support system providers in satisfying the certification requirements defined in application-sector specific standards (defense, avionics, nuclear plant control, etc.) or more generally in the IEC 61508 standard [11].

It is our opinion that the means for dependability (fault prevention, fault tolerance, fault removal and fault forecasting) should be explicitly incorporated in a development model focused at the production of dependable systems. In this paper, we present such a model, which can be termed as *dependability-explicit development model*.

This paper elaborates on our previous work reported in [14]. Our objective is not to give a tutorial on the techniques to be used to build dependable systems, but rather to define a generic framework allowing dependability-related activities to be structured and incorporated into each stage of the system creation process. As a matter of fact, all the techniques and methods proposed in the literature to achieve fault tolerance, fault forecasting, fault removal and fault prevention (see e.g., [9] for a recent summary of the state of the art) can be naturally integrated into our model.

The paper is structured into six sections. Section 2 presents the basic model proposed. Section 3 stresses the importance of fault assumptions in the development of dependable systems. Sections 4 and 5 give a list of guidelines focusing on dependability related key issues to be addressed during the requirements and the design development stages. Finally, Section 6 draws up the main conclusions of the paper.

2. Basic model

The production of dependable systems such that a justified confidence can be placed on the services delivered to the users requires the application of complementary activities aimed at fault prevention, fault tolerance, fault removal and fault forecasting. These activities can be carried out iteratively, in a sequence or in parallel, according to the lifecycle model chosen. The dependability related activities can be grouped into separate processes interacting with the system creation process and with other supporting processes. Such dependability focussed process-oriented development approach, that generalizes for instance the model proposed in the DO178B standard [20], provides a structured framework that is well suited to explicitly identify the dependability-related key issues that are needed during the development and to ensure that these issues are correctly implemented.

Following this concept, our basic model (Fig. 1) identifies three classes of processes:

- the system creation process, which builds on the classical development steps, i.e., requirements, design, realization, integration;
- the dependability processes: fault prevention, fault tolerance, fault removal and fault forecasting;
- other processes: quality assurance, certification, etc.

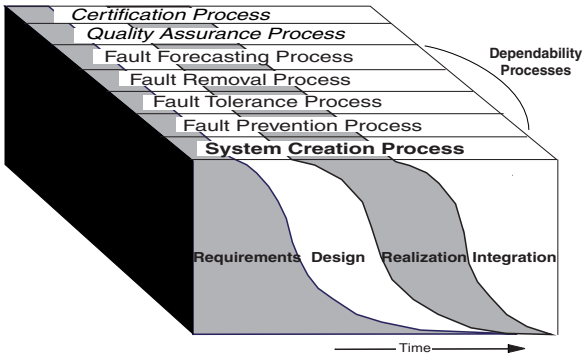

Figure 1. Dependability-explicit development model

2.1 Basic processes

The *system creation process* orchestrates the activities performed within the other processes. The requirements activities are aimed at the statement of users needs, and the (formal or informal) description of these needs in a specification. Both types of activities are differentiated, sometimes, by defining two distinct life cycle phases. Usually, this is supported by differences, on the one hand, in the levels of details and formalism and, on the other hand, in terms of responsibilities. Nevertheless, both activities are of the same nature and share a single objective, that of defining the needs and services that the system has to fulfil. Therefore, we decided not to make a distinction between them.

The design and realization activities correspond to the usual development steps leading respectively to the definition of the system architecture and the implementation of each component according to its specification. As regards integration, this system creation activity includes usual integration activities (i.e., assembling system components according to the architecture defined during the design stage and implemented during the realization stage) as well as the final system integration into its operational environment before delivery.

As regards the dependability processes, we have identified the key activities that best characterize each process, and we have searched for the minimum number of classes needed to group these activities according to their nature and objectives. This analysis led us to define three main classes of activities for each dependability process. Such classification is aimed at facilitating the identification of the key issues to be considered with respect to fault prevention, fault tolerance, fault removal and fault forecasting, and the analysis of the interactions that exist among the different processes. Each dependability process and the corresponding classes of activities are briefly presented in the following.

The *fault prevention process* is structured into three major classes of activities:

- choice of *formalisms* and languages,
- *organization* of the project,
- *planning* of the project and evaluation of risks incurred from the system development.

The *fault tolerance process* is composed of three main activities:

- the study of the *behavior* in the presence of faults, aimed at eliciting the faults against which the system will be protected,
- the *system partitioning*, aimed at structuring the system into error confinement areas, and at identifying the fault independence areas,
- the *fault and error handling*, aimed at selecting the fault tolerance strategies, at determining the appropriate mechanisms, without forgetting the

protection of those mechanisms against the faults which are likely to affect them.

The fault assumptions produced by the study of the behavior in the presence of faults constitute a basis for system partitioning, and inputs for the fault and error handling.

The *fault removal process* is composed of three main classes of activities:

- *verification,* that consists in checking whether the system adheres to properties termed the verification conditions, using techniques such as reviews, inspections, modeling and behavior analyses, testing, etc.
- *diagnosis*, that consists in searching for the faults preventing the verification conditions from being met,
- and system *modification* to perform the necessary corrections.

Finally, the *fault forecasting process* is composed of three classes of activities:

- statement of dependability *objectives*,
- *allocation* of objectives among system components,
- and, *evaluation* of dependability measures to assess whether the system satisfies the objectives or not.

2.2 Interactions between processes

The definition of the dependability requirements and the fault tolerance mechanisms to be implemented in the system should result from a global analysis and iterative refinements that take into account all the dependability processes as well as the system creation process. This leads to several interactions between these processes and among the dependability processes themselves. This can be illustrated for instance by the need, on the one hand, to verify evaluation results and on the other hand to evaluate the progress of verification activities (through the evaluation of test stopping criteria, test coverage, etc.). Another example concerns the interactions between the fault tolerance and the fault forecasting processes. In particular, the dependability properties to be taken into account for fault forecasting should be defined precisely and related to the dependability requirements derived from the analysis of the system behavior in the presence of faults performed within the fault tolerance process. This includes the definition of the acceptable degraded operation modes as well as of the constraints imposed on each mode, i.e., the maximal tolerable service interruption duration and the number of consecutive and simultaneous failures to be tolerated, before moving to the next degraded operation mode. This analysis should be done iteratively at each system decomposition step to define the criticality levels of system functions and components and the minimum level of fault tolerance to be implemented in the system. This also leads to the need to evaluate the system's capability to tolerate faults by assessing the fault tolerance coverage with respect to the fault assumptions as well as the validity of these assumptions (see §3).

2.3 Integration within the development cycle

The model proposed is in fact a *meta-model*. It is not *a* classical *life-cycle model* as it defines for each process the logical links between the activities to be conducted irrespective of their temporal sequencing. The system components can be developed according to various strategies as illustrated in the example given in Fig. 2. Indeed, similarly to the model of DO-178B, groups of

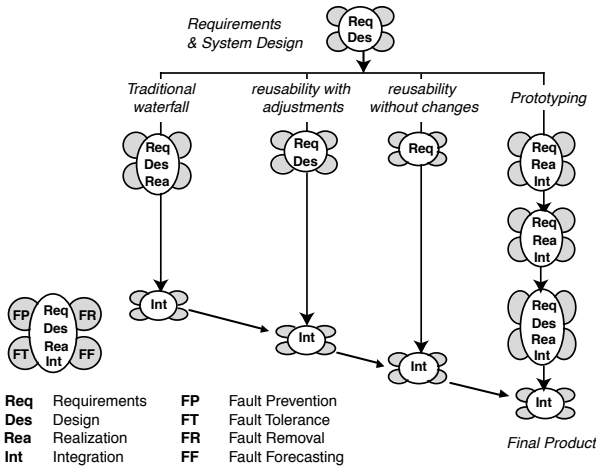

Figure 2. Example of application of the model

activities can be instantiated several times in order to accommodate several approaches in the development of a given system. In the example presented in Fig. 2, the system consists of four subsystems. The first one is developed in accordance with the waterfall model. The second one is reused but a number of customizations are introduced to meet the requirements. The third one is reused without modification. The last one is developed following a prototyping approach. Finally, the integration of the system is performed progressively, first within the different subsystems and then, between the subsystems, to arrive at the final product.

3 Fault assumptions

Fault assumptions constitute a key point in the development of dependable systems. At each system description abstraction level, associated fault assumptions should be defined taking into account the fault assumptions established at the higher levels [21]. This leads to a hierarchy of fault models. Ensuring the consistency of these fault models is a difficult task that requires a thorough examination and in-depth analysis. In particular, it is essential to study how the faults that occur at a given level manifest and propagate to the higher and lower levels. Error propagation analysis is important for the

specification and the design of error confinement barriers. It is also important for the optimization of fault tolerance verification experiments based on fault injection [23].

It is worth noting that fault assumptions related to the fault prevention, fault tolerance, fault removal and fault forecasting processes are generally not identical. Faults to be prevented are not the same as those to be tolerated or as those to be removed. For example, some verification techniques are only suitable to reveal the presence of some specific faults, depending on the location of the faults and on observability and controllability constraints. On the other hand, the fault assumptions examined during fault forecasting activities should be, generally, weaker than those considered by the other processes because of the need to validate the system under pessimistic conditions that are as close as possible to the actual operation environment. Therefore, each activity of the dependability processes should have clearly stated and justified associated fault assumptions. The assessment of the results of the activity are to be performed in accordance with the corresponding assumptions and to be accompanied by an evaluation of the validity of these assumptions [17].

The following sections give an overall view of the main activities needed for the development of a dependable system, focusing on the Requirements and Design stages. Guidelines in the form of commented checklists focusing on dependability related key issues to be addressed during the requirements and the design are presented. The activities related to the fault prevention process are discussed for the requirements stage only. Similar activities are performed during the other development stages (See [14] for complete description).

4. Checklist for the "Requirements"

The checklist defined for the requirements phase is summarized in Fig. 3 and commented in the sequel.

Requirements elicitation begins with the detailed description of the main functions to be accomplished by the system together with the definition of the dependability objectives to be satisfied. This includes the identification of the functions that are required only by some categories of users[1] (e.g., when different operational profiles are possible) and those that are accomplished only during some phases of the system mission. Different mission phases generally lead to different utilization profiles and conditions. In some contexts, the requirements elicitation has to cover the entire system life cycle including system installation, system operation and maintenance, and system decommissioning phases. Also, at this stage, it is important to identify the development constraints related to the operation of the system, and those that arise from some certification standards.

The design and the realization of the system are strongly influenced by the knowledge of future evolution of end-users needs (evolution of functional services, performances, hardware execution environment, etc.). Anticipation of the requirements evolution will make future modifications of the system to match the new requirements easier, and therefore, less expensive. Similarly, for software systems, it is desirable that portability and interoperability requirements be expressed early in the development process. The expression of such needs allows the isolation, during the design, of the components and subsystems that are expected to evolve.

In order to match users' expectations, the system end-users should be involved, as much as possible, in the requirements' definition. This is especially important to highly-critical systems that rely on human operators to ensure the supervision and monitoring of system dependability during its operation and maintenance. For these systems, the tasks to be assigned to the operators and those to be accomplished by the computing system are to be defined as early as possible and to be validated by the end-users to ensure a wide acceptability of the system and to prevent as much as possible the occurrence of human interaction faults.

4.1. Fault prevention process

Fault prevention activities carried out during the requirement elicitation stage are aimed at the definition of the development environment formalisms and tools as well as the organization and the planning of the project. These activities lead to the choice of one or several development approaches, to the distribution of the development activities between project teams and to the planning of each development stage and the definition of transition criteria between stages. The success of the project depends on the decisions taken at this step. As different possibilities can be envisaged, the assessment of the risks associated to these decisions would help in reaching a satisfactory trade-off. Particularly, the selection of the different formalisms and tools to be used during the development may be determined by constraints imposed by the dependability objectives, the certification standards, and also by performance, cost and development delay constraints. Especially, the introduction of a new technology, or new development methods is to be preceded by an assessment of associated project management and development risks. Additional decisions need to be taken with respect to the independence between development and validation teams, and the level of training, competence, and experience of all persons involved in the development.

[1] A user is another system (human or physical) which interacts with the former [13].

4.2. Fault tolerance process

The main goals of fault tolerance process activities during the Requirement phase are: 1) describe the system behavior in the presence of failures (at system level) and identify the set of undesirable events that might have unacceptable consequences on the system and its environment, and 2) define the dependability related functions to be implemented in the system to satisfy the dependability objectives. Undesirable events might come from the system environment or from the system itself. For each system function, one has to establish a list of failures (failure assumptions) which might occur during system operation, maintenance, installation, and decommissioning and study their consequences. Several issues are to be considered for the definition of this list: 1) the system boundaries and its interactions with other systems, 2) the environmental conditions and their evolution during the system lifetime, 3) the dependability properties to be satisfied, 4) the availability of additional means provided by the system environment for the detection and recovery of unacceptable behavior of the system.

System Creation Process

- Functional specifications
  - Definition of system functions (expected values and timing behavior)
  - Description of mission phases and their sequencing (phased systems)
  - Preliminary task distribution between the operators and the computing system
- Definition of the operational environment
  - System boundaries, utilization environment and user profiles
  - Operation and maintenance modes
- Development, validation and operating constraints
  - Physical constraints (weight, technology, etc.), Maintenance and operation constraints
  - Foreseeable evolutions, Reusability, Portability, Interoperability, Testability

Fault Prevention Process

- Formalisms and languages
  - Standards, rules and certification requirements, development environment, formalisms and tools
- Project organization
  - Definition of life-cycle models, Assignment of tasks to project teams, Resource management
- Project planning and project risk assessment
  - Identification of risks and means for risk reduction
  - Selection of development strategies and technologies
  - Planning of development stages and definition of transition criteria between stages
  - Project reviews planning, Configuration management planning

Fault Tolerance Process

- Description of system behavior in presence of failures
  - Identification of relevant dependability attributes and necessary tradeoffs
  - Failure modes and acceptable degraded operation modes
  - Maximum tolerable duration of service interruption for each degraded operation mode
  - Number of consecutive and simultaneous failures to be tolerated for each degraded operation mode

Fault Removal Process

- Verification Planning
  - Static verification techniques, Testing strategies (testing criteria, test generation techniques)
  - Specification of test-beds and environment simulators
- Verification assumptions
  - Classes of functions, behavior and expected faults to be analyzed by each verification technique
  - Predicates to be verified
- Verification of the requirements
  - Functional and behavioral analyses , Reviews and inspections of the specification
  - Prototyping, User-based validation, Expert reviews
- Definition of functional verification scenarios

Fault Forecasting Process

- Expression of dependability objectives
  - Definition of quantification measures and assignment of quantified targets
- Analysis of failure modes and their consequences on delivered service
  - Identification of failure modes
  - Classification of failures by severity
  - Specification of the classes of faults and failures to be addressed
- Fault forecasting assumptions
  - Modeling assumptions and parameters
- Function-by-function dependability allocation
  - Classification of system functions by criticality levels
- Fault forecasting planning
  - Selection of appropriate methods and tools for qualitative analysis and quantitative evaluation
  - Definition of a data collection environment
- Data collection and analysis
  - Feed-back from existing products, Follow-up of the product under development

Figure 3. Checklist for the Requirements

The list of the failure assumptions is to be completed by the specification of the acceptable degraded operation modes as well as of the constraints imposed on each mode, i.e., the maximal tolerable service interruption duration and the number of consecutive and simultaneous failures to be tolerated, before moving to the next degraded operation mode. The analysis of the impact of the simultaneous loss or degradation of multiple functions and services requires particular attention. Depending on the dependability needs and the system failure consequences on the environment, the need to handle more than one nearly concurrent failure mode could be vital. Such analysis is particularly useful for the specification of the minimal level of fault tolerance that must be provided by the system to satisfy the dependability objectives. It also provides preliminary information for the minimal separation between critical functions that is needed to limit their interactions and prevent common mode failures.

Due to the antagonism that exists between some dependability attributes (for example, availability and safety, availability and security), the analysis of system

failure consequences, the definition of acceptable degraded operation modes and the specification of the minimal levels of fault tolerance to be provided by the system, should take into account, globally, the different dependability properties to be satisfied by the system. Therefore, the consequences of each functional failure assumption are to be stated with respect to each relevant system dependability attribute. This will help in searching for appropriate tradeoffs between the different fault tolerance techniques and mechanisms to be implemented in the system during ulterior development stages.

Finally, the dependability needs identified at the requirement stage should be described in the form of a specification. This specification could be refined depending on the dependability objectives and lead to a set of consistent specifications where each specification focuses on a specific objective: a safety specification, a security specification, etc. These specifications will need to be refined during the design and lead to the development of more detailed fault assumption models for system components. Particularly, it will be necessary to verify during ulterior development stages that the assumptions established at this stage are all considered, and possibly update them to ensure the consistency between the fault assumptions established at each system decomposition level.

4.3. Fault removal process

The primary purpose of verification activities is to ensure that the specified requirements are compliant with the stated objectives and expected system users needs. The specified functional and dependability related requirements are to be checked for their correctness, completeness, consistency and verifiability. Verification activities also cover the results of the evaluation and dependability assessment activities performed within the fault forecasting process. At this stage of the development, the verifications will be based mainly on functional and behavioral analyses using static verification techniques (reviews, inspections, simulation, model-checking, etc.), and on engineering judgments. The validation of the dependability requirements and failure assumptions require particular attention. Data collected from similar systems and the state of the art of the corresponding application sector practices are needed to support the justification of the validity of the assumptions. The verification assumptions defining the scope of each verification technique used at this level should be clearly stated. This will help, in particular, in identifying the classes of behavior and functions that need extra checking during ulterior stages of the development because, for instance, they cannot be covered during the requirement phase.

The functional models that are generated during the requirement development stage, to describe and verify the behavior of the system, will also serve to generate functional and dependability verification scenarios to be used later for system testing.

The specification of the system is to be completed by the specification of fault removal techniques to be used during the development. Static verification techniques (inspections, reviews, walkthroughs, etc.) and testing strategies (by specifying the testing criteria and the test inputs generation methods) should be defined at this stage to allow the control of the dependability of the future system at the beginning of the development. The development of simulators and support environments for system and fault tolerance verification (e.g., by means of fault injection) may be as long and as complex as that of the system itself and may affect its development. Therefore, it is desirable, when possible, that the development of the simulators occurs concurrently with that of the system to ensure an efficient integration of the two systems and to better take into account any modification of the requirements when they occur.

4.2. Fault forecasting process

A key issue during the requirement phase is to define the dependability objectives and requirements based on the assessment of the potential levels of occurrence of undesired events and failures, and the classification of the system functions by criticality levels. The higher the rating of the consequences of an undesirable event, and the higher the probability of its occurrence, the lower is its level of acceptability. Each application sector typically has its own methods for rating the acceptability of hazards and risks, which depend on laws, regulations or an assessment of public perception. The results of failure analysis and evaluation activities include lists of failures and undesirable events with the corresponding levels of acceptability. Unacceptable ones are those which must be eliminated (prevented, avoided) by the system design. Clearly, failure analysis and evaluation activities provide critical inputs to the analysis performed within the fault tolerance process to identify the minimum levels of fault tolerance to be implemented in the system. For each function and operation mode, one has to allot a probability of success (or failure). It is important to explain how these probabilities are derived and what are the assumptions that are considered. Any validation of the system has to be completed by an evaluation of the validity of the failure assumptions and the corresponding probability allocation. Such a validation should take into account the possibility of 1) multiple failure occurrence (due, for example, to a high fault detection latency), 2) a lack of fault tolerance coverage resulting from deficiencies of fault tolerance mechanisms, or 3) a lack of independence between failures.

On the basis of the definition of system functions, the tolerable degraded operation modes and the associated

success probabilities, the criticality of the different functions can be assessed. This classification is derived from the analysis of the failure modes and their severity. The classification of functions by criticality levels (sometimes called integrity levels [11]), together with the maximal tolerable duration of service interruption associated to the services to be delivered will allow, during the design, to select the appropriate error handling mechanisms and, especially, to choose reconfiguration policies for fault processing.

The probability allocations used in the evaluation process can be derived from data collected from similar systems and from applicable certification standards. The definition of a data collection procedure is necessary to ensure the control of the system development process and to collect necessary data to carry out qualitative and quantitative evaluations. The data may be collected on the system under development or on similar systems. Particularly, these data could be used to estimate some characteristics of the system utilization environment at the beginning of the development, to establish a preliminary list of failure assumptions and to perform preliminary dependability assessments before data on the product under development are available. Also, data should be collected to analyze the activity of the operators involved in the system's operation and maintenance, and to support the definition and optimization of the tasks' distribution between operators and the computing system. These data can be collected from the field during the operational use of similar systems or during the testing of mock-ups and initial prototypes.

5. Checklist for the "Design"

The key issues to be considered during the design are summarized in Fig.4 and commented in the sequel.

The main objective of the design activity is to define an architecture allowing the system requirements to be met. Three major issues need to be addressed: 1) the system structure, 2) the system behavior and 3) the inputs and outputs for each component and data flow between components. The architecture definition consists in decomposing the system into hardware and software components, identifying the functions to be allocated to the operators responsible for system operation monitoring and maintenance, with the definition of the modes of interaction between the operators and the computing system. A component is, in itself, a system that has to be specified and designed. The specification-design recursion ends when the components from the lowest level are regarded as atomic. The relationships between system components can be described in terms of relations such as: *"uses the service of", "is composed of", "inherits from"*[7, 10, 15, 16]. Although no general method is available to decompose a system into components, the design of the system has to be thought by taking into account the future characteristics of the system such as its anticipated evolution, the desired portability, reusability and fault tolerance. Moreover, taking into account the testability of the system early in the design will facilitate the verification of the system and improve its maintainability. The architecture has to be defined by taking into account the components that can be reused with or without adjustments to meet the requirements of the system. In particular, the decision to use commercial off the shelf (COTS) components should be examined thoroughly especially to address dependability related problems [12, 22].

5.1. Fault tolerance process

The design of a system leads to the definition of several levels of abstraction. For each level of abstraction, it is necessary to establish the corresponding fault assumptions taking into account those defined for the other levels. The progressive refinement of fault assumptions and the traceability of the assumptions established at different design levels is particularly a difficult problem that needs to be addressed carefully to allow better control of the dependability related functions design. The identification of the fault classes to be handled by the system for each system component (and those that could be discarded) results from the analysis of their impact on system operation modes, taking into account the system level dependability objectives. These analyzes are supported by qualitative and quantitative evaluations and various viewpoints should be considered, for example, the nature of the faults, their origin, their persistence, etc. As temporary faults constitute a predominant cause of systems failure in operation, particular attention should be put on the error and fault handling mechanisms that are designed to address this type of faults.

System Creation Process

- Architecture definition
  - Structure: Decomposition into layers and/or components (men, hardware, software)
  - Behavior: System states and events for each layer and interaction between layers
  - Data types, data flow and interfaces between components
- Selection of development technologies
  - Identification of hardware and software components
- Identification of reusable components
  - Definition of necessary adjustments
- Operation and maintenance procedures preparation
- Integration plan preparation
  - Architecture integration strategy
  - Planning of system integration in its operational environment

Fault Prevention Process

- Formalisms and languages
- Project management
- Project planning and project risk assessment

Fault Tolerance Process

- Description of system behavior in presence of faults
  - Fault assumptions (faults considered, faults discarded)
- System partitioning
  - Fault tolerance structuring: (Fault containment regions, Error containment regions)
  - Fault tolerance application layers
- Fault tolerance strategies
  - Redundancy, Functional diversity, Defensive programming, Protection techniques, etc.
- Error handling mechanisms
  - Error detection, Error diagnosis, Error recovery
- Fault handling mechanisms
  - Fault diagnosis, Fault passivation, Reconfiguration
- Identification of single points of failure

Fault Removal Process

- Verification assumptions
  - Classes of expected faults to be analyzed, Predicates that should be verified
- Verification of the design
  - Behavioral analyses
  - Reviews and inspections of the design
  - Prototyping and simulation for man-machine interfaces verification
  - Design verification against the requirements
- Verification of fault tolerance mechanisms
  - Error and fault handling algorithms (formal) verification
  - Simulation-based fault injection (fault models)
- Unit and integration testing planning
  - Testing strategies (testing criteria, test case generation methods)
  - Fault injection strategies
- Definition of functional and structural verification scenarios
- Verification of evaluation results

Fault Forecasting Process

- Fault forecasting assumptions
- Analysis of failure modes and their consequences on the services to be delivered
- Component-by-component dependability allocation
- Preliminary assessment of the system dependability
  - Analytical models, Simulation
  - System scaling: Redundancies (men and computing system), Coverage
- Data collection and analysis

Figure 4. Checklist for the Design

The fault and failure hypotheses strongly depend on the type of architecture. The case of distributed architectures is particularly difficult [6, 18]. Distributed systems require a detailed description of the failure modes of the distributed nodes as well as of the communication network interconnecting these nodes. On the other hand, fault assumptions condition the choice of the fault tolerance mechanisms and the architecture of the system. A trade-off should be found between fault assumptions coverage (compared to the faults that really occur in operation) and the complexity of the mechanisms to be implemented to address these faults [17]. Conservative fault assumptions (e.g., Byzantine faults) will result in higher assumption coverage at the expense of an increase in the level of redundancy and the complexity of fault tolerance mechanisms. The search for a satisfactory trade-off should be guided by the dependability requirements.

Ideally, the design should list for each system component or for each system function all classes of faults that need to be covered by fault tolerance mechanisms and those for which only fault avoidance is planned. Especially, the single points of failure (the functions or components that are not fault-tolerant; for example, fault tolerance mechanisms themselves) have to be clearly identified and, ideally, formally specified and verified.

An important aspect of fault tolerance structuring is the definition of error confinement areas and fault inde-

pendence areas. Error confinement leads to the definition of error propagation barriers (between components). The definition of fault independence areas corresponds to the identification of components or sets of components whose faults are supposed to be not correlated. Fault independence areas serve also to define the granularity of fault diagnosis and to specify the on-line replacement units. The number of fault independence areas that constitute an error confinement area depends on the number of simultaneous faults to be tolerated and the type of error recovery mechanisms adopted (backward recovery or forward recovery). Particularly, it is important to identify the error propagation channels that may result from any information communication coming from a given fault independence area. Common mode failures may result from the presence of such channels. The choice between backward error recovery and forward error recovery can be influenced by the maximal tolerable service interruption duration as defined in the requirements. Depending on the maximum tolerable service interruption duration, backward or forward error recovery can be attempted, otherwise, fault masking is the alternative. The choice should also take into account the available possibilities for communication with the environment. The advantages of fault masking techniques against backward and forward recovery concerning the time needed for error recovery may be lost if redundant output interfaces (actuators) with the environment are not available.

The structuring of systems into layers offers another point of view to address error propagation and to define the fault tolerance mechanisms. A fault that affects a given layer may lead to the propagation of errors, not only to higher layers (the provided service is incorrect), but also to lower layers (for example through illegal service solicitation). The first objective of error propagation barriers is to stop the propagation of errors to the higher levels (i.e., the final users). Better error processing efficiency is obtained when these barriers are close to the layer containing the faults producing the corresponding errors. The error coverage can be improved by the implementation of successive barriers; each barrier aiming to process the errors coming from the same layer and the errors which propagate from the lower layers. However, the error propagation to lower levels, where possible, should be taken into account (e.g. through the design of access control mechanisms, the use of defensive programming, etc.).

Besides the definition of error handling mechanisms, the design should also lead to the elaboration of fault processing mechanisms. These mechanisms include techniques and algorithms for fault diagnosis, passivation of faulty components, and if needed, system reconfiguration allowing the restoration of the redundancy level or the transition to a more degraded operation mode.

Error and fault handling functions can be centralized or distributed among various components or groups of components. In the latter case, the fault and error assumptions should take into account the distribution of error and fault processing algorithms. In particular, the communication and synchronization between distributed nodes, and state consistency of concurrent processes are difficult problems that should be thoroughly studied. Moreover, the large size and distributed nature of new systems lead to the possibility of multiple fault manifestations occurring at nearly the same time. This, in turn, requires multiple recovery mechanisms to be active at the same time, with the resulting risks of multiple interference, deadlocks, and unpredictable behavior [3].

Besides the issues above, it is important to emphasize that the choice of a specific fault tolerance mechanism also depends on the type of the components considered, the faults to be addressed, and the dependability attributes to be satisfied. For example, error detection and correction codes are well suited to address memory and bus errors, whereas duplication and comparison is often used for processors. Also, security related requirements lead to the implementation of some specific protection mechanisms (e.g., identification and authentication) even though the same mechanisms can be used for other purposes, for instance for safety. Additional design decisions need to be taken concerning which mechanisms should be used to deal with design faults, how to protect the fault tolerance mechanisms themselves, and how to cope with interaction faults due to the operators. Other critical issues related to the cost of the system, its performance, the constraints imposed by certification authorities, and the risks associated with the design decisions should be considered to find a final optimal solution.

5.2. Fault removal process

The design phase can be seen as a succession of specification and design steps. Each design step should be verified against its requirements. All the issues mentioned in the requirements should be addressed. Particular attention is to be put on checking the refinement of fault assumptions and the adequacy of fault processing and error detection and recovery mechanisms. Common cause analysis can be used to support the examination of failure independence assumptions and verify the correct partitioning of the system into independent fault and error confinement areas. The design of single point of failure components should be thoroughly examined and verified using, for example, formal verification techniques. Verification should also address the feasibility and completeness of the planned tests, and analyze the planned procedures for system maintenance and operation.

Besides the assumptions considered in the design, assumptions are also to be specified for the verification activities. Particularly, when formal verification techniques are used to verify, for example, the fault tolerance algorithms, it is important to define the assumptions behind the proofs and verification conditions. Concurrently with the definition of the system architecture, a verification plan specifying the techniques and methods to be used for unit and system integration testing should be prepared. Particularly, the system validation tests to be applied before the delivery of the system to the customers should be planned. The choice of an integration strategy leads to the definition of the testing environment that will be used for system integration testing. Such choice may have an impact on the organization of integration tests. Indeed, a bottom-up integration strategy requires the development of drivers that simulate the environment of the components to be tested. This testing strategy allows the early testing of elementary components. However, the interface errors between components can only be revealed later in the development process. On the other hand, a top-down integration strategy requires the development of possibly sophisticated test stubs that simulate the functions accomplished by the lower-level components.

5.3. Fault forecasting process

The choice of an optimal architecture and the allocation of dependability requirements to system components are supported by preliminary dependability evaluations to analyze if the dependability objectives can be met by the selected architecture. Analytical modeling and simulations are two main techniques that can be used to derive the quantitative evaluation measures. Several methods can be used to build the evaluation models [19] (fault trees, FMECA, Markov chains, Stochastic Petri nets, etc.). Each one of these methods is based on some modeling assumptions that need to be stated and justified in the context of the evaluation performed. The classes of faults to be considered in the evaluation of system behavior (for example independent and correlated faults) and the assumptions about the statistical distribution and the parameters used in the evaluation models have to be clearly specified. These assumptions are to be defined early in the development process to identify the techniques and tools that will be used for system dependability analysis and evaluation. The specification of these assumptions leads to better interpretation of evaluation results. For instance, to avoid any ambiguity, the specification of a target fault coverage rate should be completed by the definition of the classes of faults to be considered. It should be specified whether the target fault coverage rate concerns physical faults of a specific type (e.g. stuck-at faults only) or any type of faults. If some simplifying assumptions are considered, the approximations made should be justified using, for example, sensitivity analyses.

At early stages of the development, there is no precise knowledge of the reliability of the components that will be integrated in the architecture. The preliminary dependability evaluations consist in sensitivity studies assuming a range of variation of the parameters used in the evaluation models: for example, different component failure rates can be assumed, and their impacts on the global reliability studied. These evaluations will be facilitated when the product to be developed results from improvements of previous similar products. As a consequence, several components for which reliability data are available are reused in the new architecture. However, as the statistics and data collected from other products might correspond to different utilization environments and conditions, the uncertainty related to environmental condition variation should be assessed.

The risk that may arise during the design of fault tolerance mechanisms and error propagation barriers is to introduce too many mechanisms to compensate the lack of fault tolerance coverage. The evaluation of the contribution of each fault tolerance mechanism to the global fault tolerance coverage enables this risk to be reduced [2, 4]. Coverage evaluation can be supported by fault injection experiments [1] or by analytical modeling [8]. During the design phase, faults can be injected in simulation models to check the efficiency of fault tolerance mechanisms. These experiments enable early analysis of the efficiency of fault tolerance mechanisms on the system dependability. The evaluation of error detection latency and error recovery completion time is particularly important to assess the likelihood of occurrence of multiple nearly coincident faults.

6. Summary and conclusions

The dependability-explicit development model presented in this paper provides a general framework allowing the different activities performed during the development of dependable systems to be structured and organized. Grouping fault prevention, fault tolerance, fault removal and fault forecasting activities into supporting processes that interact with the system creation process and with each other, aims to ensure that dependability related issues are not overlooked, but rather considered at each stage of the development process. It is noteworthy that the framework proposed within the dependability-explicit development model to structure dependability related activities does not rely on a specific life cycle model. Any life cycle model that fits the needs and the constraints of the target system can be used.

The dependability-explicit development model is generic and can be applied to a wide range of systems and application domains. For a given system, a number of

customizations might be needed. Some activities could be discarded if, for example, some system components are reused or if the dependability objectives to be satisfied do not require the implementation of these activities. The list of key-activities and guidelines proposed in the paper for the requirements, design, realization and integration development stages can be applied, irrespective of the development methods used (conventional functional approach, object-oriented, etc.). These guidelines focus on the nature of activities to be performed and the objectives to be met, rather than on the methods to be used to reach these objectives. Indeed several complementary techniques and practices could be used to reach the same objectives. The selection of optimal solutions depends on the complexity of the system, the dependability attributes to be satisfied, the confidence level to be achieved, and the constraints related to cost limitation or imposed by the certification standards. Especially, the model proposed can be used to support the ongoing standardization efforts towards the definition of application sector specific standards focused on the development and certification of dependability related issues. Indeed, it can be used as a baseline to define and to structure the objectives and the requirements related to dependability to be satisfied by the product to be assessed as well as the evidence to be provided to show that the product satisfies the dependability requirements assigned to it. These requirements are to be defined taking into account the specific constraints and needs of each application sector.

References

[1] Arlat J., Crouzet Y. and Laprie J.-C., "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", in *Proc. 19th Int. Symp. on Fault Tolerant Computing (FTCS-19),* pp.348-355, IEEE Computer Society Press, Chicago, IL, USA, June 1989.

[2] Arnold T. F., "The Concept of Coverage and its Effect on the Reliability Model of a Repairable System", in *Proc. 2nd Int. Symp. on Fault-Tolerant Computing (FTCS-2),* pp.200-204, IEEE Computer Society Press, Newton, MA, USA, June 1972.

[3] Avizienis A., "Infrastructure-Based Design of Fault-Tolerant Systems: How to Get High-Confidence Computing for All", in *Proc. 1998 IFIP Int. Workshop on Dependable Computing and Its Applications (DCIA98),* pp.51-69, Johannesburg, South Africa, January 1998.

[4] Bouricius W. G., Carter W. C. and Schneider P. R., "Reliability Modeling Techniques for Self-Repairing Computer Systems", in *Proc. 24th. National Conference,* pp.295-309, ACM Press, 1969.

[5] BSI, *Reliability of Constructed or Manufactured Products, Systems, Equipment and Components, Part 1. Guide to Reliability and Maintainability Programme Management*, British Standard Institution, Report N°BS 5760, 1985.

[6] Cristian F., "Understanding Fault-Tolerant Distributed Systems", *Communications of the ACM*, 34 (2), pp.56-78, 1991.

[7] Davis A. M., *Software Requirements: Objects, Functions, and States,* PTR Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[8] Dugan J. B. and Trivedi K. S., "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems", *IEEE Trans on Computers*, 38 (6), pp.775-787, 1989.

[9] FTCS-25, *Proc. 25th Int. Symp. on Fault-Tolerant Computing (FTCS-25). Special Issue,* IEEE Computer Society Press, 1995.

[10] Ghezzi C., Jazeyeri D. and Mandrioli D., *Fundamentals of Software Engineering,* Prentice-Hall, 1991.

[11] *Functional Safety of Electrical/Electronic/Program-mable Electronic Safety-Related Systems*, International Electrotechnical Commission SC 65A 1998.

[12] Koopman P., Sung J., Siewiorek D. and Marz T., "Comparing Operating Systems Using Robustness Benchmarks", in Proc. *Symp. on Reliable and Distributed Systems (SRDS'97),* pp.72-79, IEEE Computer Society Press, Durham, NC, USA, 1997.

[13] Laprie J.-C. (Ed.), *Dependability: Basic Concepts and Terminology (in English, French, German, Italian and Japanese),* Dependable Computing and Fault Tolerance, 5, 265p., Springer-Verlag, Vienna, Austria, 1992.

[14] Laprie J.-C. *et al*., *Dependability Handbook,* Cépaduès, Toulouse, France, 1995-96.

[15] Neumann P. G., "On Hierarchical Design of Computer Systems for Critical Applications", *IEEE Trans on Software Engineering*, SE-12 (9), pp.905-920, 1986.

[16] Parnas D. L., "On the Criteria to be Used in Decompos-ing Systems into Modules", *Communications of the ACM*, 15 (12), pp.1053-1058, 1972.

[17] Powell D., "Failure Mode Assumptions and Assumption Coverage", in Proc. *22nd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22),* pp.386-395, IEEE Computer Society Press, Boston, MA, USA, July 1992.

[18] Powell D., "Distributed Fault Tolerance: A Short Tutorial", in Proc. *1998 IFIP Int. Workshop on Dependable Computing and its Applications (DCIA98),* pp.1-12, Johannesburg, South Africa, January 1998.

[19] Reibman A. L. and Veeraraghavan M., "Reliability Modeling: An Overview for System Designers", *IEEE Computer*, 24 (4), pp.49-57, 1991.

[20] RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA/EUROCAE, Paper 591-91/SC167-164, N°DO-178 B.5, November 1991.

[21] Siewiorek D. P. and Swarz R. S., *Reliable Computer Systems - Design and Evaluation,* Digital Press, Bedford, MA, USA, 1992.

[22] Voas J. M., "Certifying Off-the-Shelf Software Components", *Computer*, 31 (6), pp.53-59, 1998.

[23] Yount C. R. and Siewiorek D. P., "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Trans on Computers*, 45 (8), pp.881-891, 1996.