Fusion Engineering and Design 88 (2013) 1912–1915

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Breaking down the requirements: Reliability in remote handling software

Pekka Alho*, Jouni Mattila

Department of Intelligent Hydraulics and Automation, Tampere University of Technology, Finland

HIGHLIGHTS

- We develop a set of generic recommendations for control system software requirements.
- We analyze ITER remote handling system requirements.
- Requirement specifications have major impact on software reliability.
- Reliability requirements need to be managed as a system measure.
- Systematically developed requirements can be used to form a dependability case.

ARTICLE INFO

Article history:
Received 13 September 2012
Received in revised form 22 October 2012
Accepted 7 November 2012
Available online 4 December 2012

Keywords:
remote handling, control system, software, requirements, reliability, dependability

ABSTRACT

Software requirements have an important role in achieving reliability for operational systems like remote handling: requirements are the basis for architectural design decisions and also the main cause of defects in high quality software. We analyze related recommendations and requirements given in software safety standards, handbooks etc. and apply them to remote handling control systems, which typically have safety-critical functionality, but are not actual safety-systems–for example the safety-systems in ITER will be hardware-based.

Based on the analysis, we develop a set of generic recommendations for control system software requirements, including quality attributes, software fault tolerance, and safety and as an example we analyze ITER remote handling system software requirements to identify and present dependability requirements in a useful manner. Based on the analysis, we divide a high-level control system into safety-critical and non-safety-critical subsystems, and give examples of requirements that support building a dependable system.

1. Introduction

ITER will feature a large number of remote handling (RH) systems, including divertor, blanket, port handling, viewing, neutral beam, transfer cask and hot cell. Proper maintenance and operation of ITER is not possible without these systems, and their reliable operation is necessary for ensuring that the plant is available for fusion experiments. Achieving this goal requires reliable mechanical components and designs, together with a suitable maintenance strategy. However, software failures have passed hardware as the most common source for computer system outages already in the last century [1], and modern control systems have complex functionality implemented with software. Software failures therefore present a major threat for ITER RH systems, which are safety-critical in the sense that a fault could damage research equipment or cause maintenance outages, potentially reducing experimental time.

Software requirement specifications have an important role in establishing safe and reliable RH operations, especially since the RH systems will be developed by several contractors. This is because: 1) requirements set targets that are used to verify software quality, including reliability; 2) quality attributes (i.e. non-functional requirements for "how well" the system should perform) drive significant architectural and design decisions [2] and 3) requirement specification is the largest source for defects in high-quality software [3]. In order to improve dependability in control systems, our research evaluates effective ways for RH system development teams to present related software requirements.

2. Comparison of RAMI process and dependability

Reliability is defined as the probability of failure-free operation for a specified period of time in specified environment [4]. It is also one of the key attributes in the RAMI process [5] used in the ITER project to manage risks in the facility development and design. In the RAMI process every system undergoes a risk-analysis to evaluate what can go wrong and to recommend spare components,

* Corresponding author.
E-mail address: pekka.alho@tut.fi (P. Alho).

0920-3796/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.fusengdes.2012.11.008

back-up systems, maintenance schedules, etc. to reduce the risk level of breakdown to minimum [6]. RAMI stands for:

- Reliability (continuity of correct service),
- Availability (readiness for correct service),
- Maintainability (ability to undergo modifications and repairs) and
- Inspectability (ability to undergo easy visits and controls) [6,7].

This is similar to the concept of dependability used in computing and communication systems. Dependability has same attributes, except instead of inspectability it has *integrity* (absence of improper system alterations) and *safety* (absence of catastrophic consequences on the user and environment) [7], being more relevant for RH software.

Specifications and standards usually implicitly or explicitly focus on hardware and are largely silent about software reliability and other quality attributes [8], and the RAMI process seems to be no exception. E.g. inspectability is essentially a requirement for mechanical systems. Dependability-related requirements for software-based systems need to take into account that failure mechanisms of software differ from mechanical systems. Hardware usually fails because of physical faults caused by wear and aging, whereas software failures are typically caused by human errors made in the development phase of the system and are deterministic in nature, making software faults harder to predict, locate and correct [4]. Because of these reasons, proving the reliability of software is not as straightforward as for mechanical subsystems and the related requirements for software need to reflect this.

3. System fault tolerance and dependability requirements

In this chapter we analyze the practices of developing dependability requirements and apply them to RH system software. RH systems typically need high reliability and have a combination of safety-critical and non-critical subsystems. Software systems are complex, which makes fault tolerant and dependable software costly: development often includes risk assessments, verification & validation procedures, and restrictions on design choices. However, use of a particular technique or techniques is not evidence of software quality, and even certified systems fail [9].

For high quality software–like control systems–the requirements specification is the most important source of delivered defects [3]. These defects can be due to errors, changes or omissions in requirements. Errors and changes can be usually discovered and managed with inspections (validation of requirements) and tools, but missing requirements can be considerably more difficult to detect. Possible sources for software requirements include system requirements specification (which includes system safety requirements), software hazard & risk analyses, hardware & environmental constraints and customer input [10]. To adequately define dependability and fault tolerance requirements for a system, several aspects of the software must be documented, including quality attributes, intended modes of operation, timing requirements, failure modes, and safety-related functionality which are briefly covered next.

3.1. Dependability objectives

The dependability objectives are documented in quality attributes (reliability, availability etc.). For control systems, important attributes include e.g. interoperability and evolvability (which has longer-term focus when compared to maintainability) because of the long expected lifetimes. Different subsystems may have different target levels of reliability.

Dependability objectives must be defined for a given environment, i.e. operation conditions. No system can be dependable under all conditions, so the claims must be made explicit [11]. These include not only environmental factors, but also expected interaction with external systems and humans.

3.2. Operation modes

Modes of operation are based on operational conditions or mission phase. By specifying operation modes we can limit the amount of functionality that has to be considered at a time.

Operation modes can also affect enabled commands or allowable limits for parameters, which has safety implications. Examples of operation modes important to dependability include automatic & manual, degraded operation and recovery modes.

3.3. Timing requirements

Timing requirements include communication deadlines, sampling rates, time to criticality etc. If the system has timing requirements that include hard deadlines, this has major impact for the system architecture design.

Safety and reliability can also be in odds–reliability can cause non-determinism for communications, as resent information could already be old. Especially in safety-critical systems it is often more important to keep sending up-to-date information.

3.4. Fault tolerance and responses to undesired events

Even though software developers work to create correct requirements and code, software will always have faults–and the number of delivered defects per function point goes up with software size and complexity [3]. Thus we also need to consider responses to undesired events, even if the software has low number of defects. This includes needs for fault tolerance (error detection, recovery, redundancy), specifying failure modes, i.e. how the system should fail, and what the system is not allowed to do in the case of failure.

Another factor that has to be considered in the case of errors is the tradeoff between robustness and correctness: robust software function tries to return some value (even if inaccurate) and correct software will return no results, which is usually better for safety-critical systems since faults will be easier to detect.

3.5. Safety-critical requirements

Reliability focuses in costs of failure and downtime, whereas safety focuses in dangerous failure modes. When a potentially unsafe command is detected, safety system inhibits the hazardous command and initiates transition to a known safe state. E.g. ITER will have a hardware-based plant interlock system which implements investment protection functions [12].

Safety-critical software covers software that has *impact on hazards* (cf. safety systems that are used for avoidance or control of hazards). Plant subsystems like RH may have complex safety-related functionality which must be implement with software. Examples of such functions include stability of machines, anti-collision systems and reduced speed & restricted space for robots [13]. Any such software feature identified as a potential hazard should be designated as safety-critical to ensure that future changes and verification processes can take them into consideration [10]. Candidates for safety-critical items list can be found e.g. with software FMEA.

4. Example: ITER RH software requirements & recommendations

4.1. Safety standards in ITER

Codes and standards applicable at ITER are French standards for a basic nuclear plant. Main standard is IEC61513 (Nuclear power plants–instrumentation and control systems important to safety–general requirements for systems) [14] which has been derived from the functional safety standard IEC61508.

Safety systems in ITER fall into three categories, which are nuclear safety, occupational safety and personal access safety. However, RH systems do not implement safety functions for these categories, even though they may have some safety-related functionality (i.e. economical hazards for research equipment and plant availability). It is therefore seen that RH systems are not "instrumentation and control systems important to safety" as intended in the standards, and therefore not in their applicable domain. The developers therefore have more flexibility to choose most efficient practices, but also need more expertise to do so without compromising dependability of the system.

4.2. Non-functional requirements

In this section we analyze some dependability-related requirements chosen from the system requirements document for remote handling control systems [15].

Reliability:

- "RH operations without causing significant damaging to ITER components shall have a target reliability of greater than 98% over a 120 day operational period."
- "The RH high-level control system shall have target reliability against significant failures of greater than 90% over a 120 day operational period."

Target level for RH operations without significant damaging equals safety integrity level (SIL) 1 in the IEC61508 functional safety standard (probability of failure per hour $\geq 10^{-6}$ to $< 10^{-5}$). The high-level control system itself has lower reliability requirements so only safety-critical operations need to be considered for higher reliability level, instead of the whole control system. To be useful, the requirements specification should also define "significant damaging" and "significant failures" precisely.

For hardware components it is often possible to evaluate their reliability based on their historical data and/or subcomponent specification. However, for software components such data is not usually available. Approach used in the IEC61508 is based on giving recommendations on the use of specific techniques in the development based on the target SIL. Other sources recommend building a dependability case for the software, which should explain why the critical properties hold e.g. with (formal) requirements, testing results and verification [11].

Maintainability:

- "The RH high-level control system shall have a maintenance system that ensures that significant failures have an average recovery time of less than 1 day."

For software, maintainability means how easy it is to correct defects and make changes. To achieve high maintainability, software needs to be well documented and easy to understand. Especially the latter can be hard to achieve with a complex system like RH. Maintainability can also be greatly affected by choice of programming environment and language, as e.g. licensed programming languages are generally lacking in 3rd party tool support like version control.

Software updates and fixes may also increase failure rates as new faults can be injected to the system. This is critical for ITER because the RH systems need to be able to accommodate new and changing functional requirements over time, caused by evolving experimental changes.

4.3. Safety-critical subsystems

High-level control system for ITER RH [15] is used to demonstrate division to safety-critical & non-safety-critical subsystems in Fig. 1. Software subsystem is considered safety-critical if it controls hazardous or safety-critical hardware or software or provides information upon which a safety-related decision is made [10]. Systems that monitor safety-critical hardware or software as part of a hazard control are considered as safety-systems and are presumed to be implemented with hardware (interlocks).

Safety critical subsystems include:

- RH input device and computer assisted teleoperation (CAT) which are used for control in manual control.

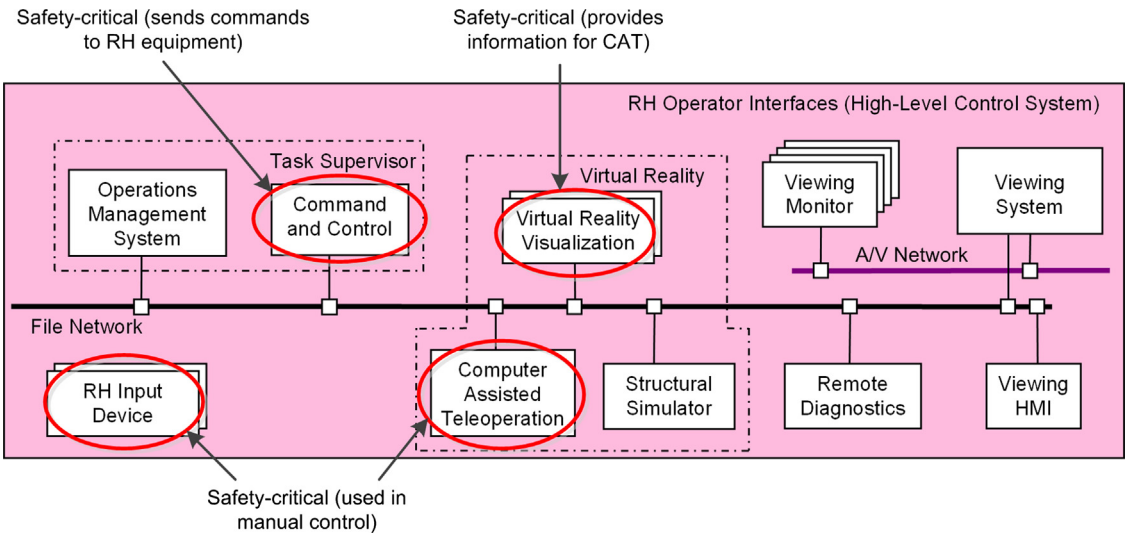

Fig. 1. High-level control system [15], showing safety-critical subsystems as identified in this paper.

- Virtual reality which provides information for CAT (dependencies are not shown in the figure).
- Command and control which is used for executing commands in automatic operations.

Structural simulator, remote diagnostics and viewing system are not considered safety-critical in this example because they do not directly impact hazardous operations, but are used as supporting information sources and for data analysis. However, now the system has a combination of safety-critical and non-safety-critical subsystems. Without decoupled system architecture and error detection, there is a danger of fault propagation. Therefore the system will need requirements and constraints like example 2 in the next section to reduce common-mode failures.

4.4. Example requirements

4.4.1. Example 1

Fast controllers (in a plant system) may run Linux as the operating system and have safety-critical functionality, but Linux is not validated for safety-critical use. However, extensively used software may reach reliability levels suitable for SIL1 or even SIL2 [16]. Recommendation is to restrict allowed versions for kernel and distribution to specific, well-tested versions, and gather reliability-related evidence for building a supporting dependability case. Constraint could be e.g.

[CO-1] Linux kernel version must be 2.6.30.8.

4.4.2. Example 2

To prevent fault propagation, add requirements to support modularity and decoupling between subsystems:

[MA-1] No direct inter-module references (function calls, class references etc.) to other software modules are allowed between subsystems.

4.4.3. Example 3

Specify dependability requirements explicitly for different properties. E.g. incorrect “stop” is significantly less dangerous than incorrect “go” [11].

[RE-1] No more than one out of 100 RH operations using collision detection and virtual force functionality for guiding telemanipulation operations shall be incorrectly stopped.

[RE-2] No more than one out of 10 000 RH operations using collision detection and virtual force functionality for guiding telemanipulation operations shall cause significant damage to ITER components.

5. Conclusions

Requirement specification presents a major source of defects for control systems, and thus has major impact on reliability. However, risk management processes and codes often neglect software reliability, even though reliability requirements need to be managed as a system measure that accounts for both hardware and software, and their different characteristics taken into account. Especially proving the reliability of software can be problematic.

Remote handling systems have safety-critical functionality like stability and anti-collision systems implemented with software, whereas actual safety functions are usually implemented with hardware. Software development is therefore guided by dependability requirements instead of strict safety standards, as risks are economical ones. Cost-efficient development needs to take into account that different subsystems therefore need different levels of reliability (e.g. diagnostics is not as critical as command & control). Systematically developed requirements can be used to form a dependability case for the system under development, where requirements, architectural solutions, verification etc. is provided to give sufficient confidence in the reliability of software.

Acknowledgments

This work was carried out under the EFDA Goal Oriented Training Programme (WP10-GOT-GOTRH) and financial support of TEKES, which are greatly acknowledged. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

References

[1] J. Gray, A census of Tandem system availability between 1985 and 1990, IEEE Transactions. on Reliability (1990) 409–418.
[2] K. Wiegers, Software Requirements, Microsoft, Redmond, 2003.
[3] C. Jones, Software quality in 2011: a survey of the state of the art, 31 August 2011 (accessed 14 August 2012), http://sqgne.org/presentations/2011-12/Jones-Sep-2011.pdf
[4] M. Lyu, Handbook of Software Reliability Engineering, IEEE Computer Society Press and McGraw-Hill, 1995.
[5] K. Chan, et al., Development of a RAMI program for LANSCE upgrade, Particle Accelerator Conference (1995) 822–824.
[6] I. Rona, RAMI, or thinking ahead, 27 Oct. 2008 (accessed 9 August 2012) http://www.iter.org/newsline/55/1193
[7] A. Avizienis, et al., Basic Concepts and Taxonomy of Dependable and Secure Computing, Transactions. on Dependable and Secure Computing 1 (1) (2004) 11–33.
[8] M. Hecht, K. Owens, J. Tagami, Reliability-Related Requirements in Software-Intensive Systems, Reliability and Maintainability Symposium (2007) 155–160.
[9] D. Jackson, A Direct Path to Dependable Software, Communications of the ACM 4 (52) (2009) 78–88.
[10] NASA Software Safety Guidebook, NASA, 2004.
[11] D. Jackson, M. Thomas, L. Millett, Software for Dependable Systems: Sufficient Evidence? National Academy Press, 2007.
[12] ITER, Plant control design handbook, 27LH2V, 2011.
[13] T. Malm, et al., Safety-critical software in machinery applications, VTT, 2011.
[14] L. Scibile, et al., The ITER safety control systems – status and plans Fusion Engineering and Design 85 (2010) 540–544.
[15] D. Hamilton, System Requirement Document SRD-23-07 Remote Handling Control System, ITER, 2008.
[16] P. Bishop, R. Bloomfield, P. Froome, Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications, CRR 336/2001, UK Health and Safety Executive Contract Research Report, 2001.