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.