
Derivation of Safety Requirements for an Embedded Control System

Peter A. Lindsay
School of Information Technology and Electrical Engineering
University of Queensland

John A. McDermid
Dept of Computer Science
University of York, UK

ABSTRACT
Overlooked and poorly understood requirements are known to be a major cause of software-related failures in system designs. This paper describes the application of a process for deriving software safety requirements to an embedded control system for a defence-related application. The process integrates a number of different hazard analysis techniques in a systematic fashion that enables mission-critical concerns to be balanced against safety concerns. System-level safety requirements are identified and flowed down to software functional requirements based on a Real-Time Network (RTN) description of the software architecture. Along the way, a detailed cause and effect model of component failures, system hazards and mishaps is developed which would form the basis of a safety case for the system.

INTRODUCTION
This paper describes the application of a new process for identifying and ranking safety requirements for software-intensive systems. Overlooked and poorly understood requirements are known to be the major cause of software-related failures in system designs [Luqi 2002]. It is important that critical requirements be identified early in a development project, when it is cheaper and easier to correct them, for example by developing architectures that are robust against failures.

The requirements-identification process illustrated here was developed jointly with John McDermid of the University of York, UK. The process was first presented at SAFECOMP 2000 [Lindsay 2000] and is described in more detail in [Lindsay 1999]. The process is heavily influenced by traditional safety engineering analysis approaches such as ARP 4754 [ARP 4754] and is consistent with modern safety standards for computer-based systems such as IEC 61508 [IEC 61508] and Def(Aust) 5679 [Def(Aust) 5679]. It differs from them in a number of ways (see [Lindsay 2000] for details), primarily in where it places the onus of responsibility for the structure of the safety argument: instead of using a notion of Safety Integrity Levels (which, for system and software design primarily relate to the degree of rigour applied in the development process), our approach derives quantified target safety-integrity requirements which the system developer would be required to show have been met. We argue [Lindsay 2000] that our approach is more flexible and produces more credible results than process-based approaches.

This paper reports the outcomes of a trial application of the requirements-identification part of the process to a defence application: an air-to-air missile being developed by Matra BAE Dynamics. The process was trialled as part of an investigation by the UK Ministry of Defence into ways of improving defence system safety standards. Quantification of safety requirements was not treated in the trial.
OVERVIEW
The process and its theoretical basis are described in detail in [Lindsay 1999].
System-level safety requirements are flowed down to components, by integrating system hazard analysis techniques in a manner that enables functional and quantitative analysis.

Figure 1. Missile schematic

In outline, the steps of the proposed process are:

1. Construction of a System Conceptual Design model which defines the scope of the system and the functions of its components
2. Identification and classification of system hazards
3. Identification of system safety functions and protective measures through consideration of accident sequences
4. Construction of a detailed cause and effect model, to record how faults propagate through the system from component level to system level, and to identify possible common causes
5. Allocation of a budget of safety integrity requirements, using the models from previous steps to justify quantitative targets

Because of space limitations the paper simply outlines the models and techniques used at steps 1–4 of the process, and illustrates some of the results. The design notation used to express the software architecture here is the Real Time Network (RTN) specification language developed by Matra BAE Dynamics [Paynter 2000]. RTN is derived from MASCOT, the Modular Approach to Software Construction, Operation, and Test methodology [Simpson 1986]. The terminology of IEC 61508 is used where possible.

CASE STUDY: AN AIR-TO-AIR MISSILE DESIGN
The example concerns the design of the Electronics Unit (EU) of a simplified, hypothetical Long Range Air-to-Air Missile (LRAAM), illustrated in Figure 1. As the name suggests, LRAAM is launched from an aircraft and propelled by a rocket motor towards an airborne target. The missile incorporates a “Seeker”, which searches for (“acquires”) a target and guides the missile accurately to the target in its final stages. The missile can be mounted on and launched from a variety of positions on an aircraft, including a wing pylon or body recess.

The Electronics Unit on the missile manages the other missile subsystems, including communications with the launch vehicle, and implements the flight control laws (see Figure 2). The EU takes input from inertial movement sensors in a separate Inertial Measurement Unit (IMU), which provides data on missile acceleration, including roll, pitch and yaw. This input is processed within the EU to provide inertial navigation: i.e., to calculate the missile’s velocity and displacement relative to an initial reference point. The EU sends outputs to fin actuators to control flight. The missile is long and slender, and sudden changes in commanded direction can result in loss of control and break-up of the missile; hence correct implementation of control laws is critical.

The launch vehicle’s Stores Management
System (SMS) is responsible for igniting the missile’s rocket motor. The EU has no control over the rocket motor once ignited. The EU communicates with its launch vehicle’s Inertial Navigation System (INS) and SMS via a physical connection (“umbilical”) in the pre-launch sequence, to initialise and check missile subsystems and to load an initial target position. Fusing and detonation of the warhead will not be treated here. Constraints on space and power consumption mean there is limited possibility for redundancy in missile sensors and systems, so it is important that the EU software be designed carefully for safe operation.

SAFETY HAZARDS
Here we outline the main safety hazards for such a missile. We consider only safety hazards arising in the operational concept described above (and within the EU’s power to influence): in particular, we ignore handling & testing hazards, flight out of Range, violation of safe lanes, etc.

Dangers to launch vehicle include: fusing at or near launch vehicle; inadvertent launch; break-up near launch vehicle; hitting the launch vehicle as the rocket escapes; awkward/no escape (gouging or rocket motor ignited but missile not released). Dangers to other vehicles include: hitting an accompanying vehicle; and hitting the wrong target (collateral damage). Our process can be used to treat mission-critical aspects (such as unavailability of the missile, or failing to detonate near target) in the same framework, unlike most defence safety approaches.

SYSTEM CONCEPTUAL DESIGN MODEL
The System Conceptual Design model used to identify critical system-level safety requirements was based on a simple decomposition of the missile’s lifecycle into top-level modes (Figure 3), a functional breakdown (Figure 4) and an assignment of system functions to modes (Figure 5). System functions include functions internal to the EU as well as interfaces to external systems.

In outline, the top-level system functions of the EU considered were as follows: IMU takes inputs from the missile’s Inertial Measurement Unit. Navigation calculates the missile’s current position, velocity, and attitude from IMU data (plus initial data from the launch vehicle). Data Transfer
handles data coming into the EU from the umbilical, including aircraft Inertial Navigation System (INS) data, target initial position, missile station (mount position), and launch vehicle manoeuvring data. The latter two data items are used by the Separation Autopilot to calculate the missile’s initial escape trajectory.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Corresponding phase in missile lifecycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerUpCheck</td>
<td>Power-up stage, with built-in hardware checks</td>
</tr>
<tr>
<td>Prelaunch</td>
<td>Initialisation stage, with software system checks</td>
</tr>
<tr>
<td>Launch</td>
<td>Ignition of rocket motor &amp; flight clear of launch vehicle</td>
</tr>
<tr>
<td>Flight</td>
<td>Free flight towards initial target position</td>
</tr>
<tr>
<td>Acquisition</td>
<td>Seeker locks onto target &amp; updates target position</td>
</tr>
<tr>
<td>Terminal</td>
<td>Seeker takes over guidance, warhead gets armed &amp; detonates close to target</td>
</tr>
</tbody>
</table>

Figure 3. Top-level system modes

The Actuators function handles signals to the actuators controlling the missile fins. Status Reporting is used for reporting results of pre-launch checks from the missile to the launch vehicle, including a Built-In Test (BIT) and that the IMU and Navigation functions are working correctly, by checking against aircraft INS data. The launch is blocked by the SMS until all status checks have been confirmed. The Flight Autopilot is an inner control loop that controls flight surfaces in free flight (post-escape). Initial Guidance is an outer control loop that calculates an optimal flight path, with position and alignment updates from Navigation. The target position is initially obtained from the launch vehicle and, once acquired, from the Seeker. Final Guidance (using the Seeker to define a flight path direct to the target) and Fuzing (arming the warhead) will not be discussed here.

**SYSTEM HAZARDS**

The next step in our process is to identify and classify system hazards: i.e., system conditions or states that can result in mishaps. We applied a form of Functional
**Failure Analysis (FFA)**, in which the consequences of failure of each of the top-level system functions were considered in turn, using modes and keywords to guide consideration of functional failures in a systematic, high-level manner. FFA is a

<table>
<thead>
<tr>
<th>Function</th>
<th>Mode</th>
<th>Failure mode</th>
<th>Effect</th>
<th>Sw detection /protection</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMU</td>
<td>Prelaunch</td>
<td>Omission</td>
<td>Missile unavailable</td>
<td>Status report to SMS</td>
<td>Consider firing ballistically as deterrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incorrect – drift</td>
<td>Missile off-target, may hit launch vehicle</td>
<td>Perform error check</td>
<td>Check robustness of Separation autopilot</td>
</tr>
</tbody>
</table>

**Figure 5. Assignment of system functions to modes**

fairly well known technique in safety engineering but is not currently very well written up in the literature: [ARP 4761] has an example. The keywords used for the LRAAM case study were: *omission* (function not provided), *commission* (function provided when not desired) and *incorrect* (incorrect values passed). Figure 6 illustrates part of the FFA table for a failure of the IMU during the pre-launch phase.

As well as identifying the system failures that could result in mishaps, the FFA clarified the part played by the EU in detecting and protecting against such failures and external failures, and yielded derived system safety requirements. The latter included for example the need for the pilot to be able to override the SMS and to launch a missile despite failed pre-launch checks, under certain circumstances. (This illustrates the advantage of being able to treat mission-criticality and safety-criticality in the one framework.) FFA is a relatively quick and lightweight way of identifying system hazards, well suited to workshops with multiple stakeholders.

**SYSTEM SAFETY FUNCTIONS**

The next step in our process is to conduct a more detailed analysis of the system’s design, to identify system safety functions and protective measures through consideration of accident sequences. This
was done by Event Tree Analysis (ETA) [Leveson 1995], starting from two different kinds of initiating event: individual failures of system functions (which thus covers external failures and interface failures); and external events which may be hazardous in any or all phases of operation, and which may be common cause failures, such as

![Event Tree Analysis Diagram]

Figure 7. Event tree for value failure of the Navigation function

power loss. (For systems involving human actions, the analysis would also cover initiating events associated with operators or other personnel which require the triggering of a protection mechanism to avoid a hazard.)

Figure 7 gives the event tree for a value failure of the Navigation function. The analysis revealed of the pre-launch check that the IMU and Navigation functions are working properly: if not, then the Separation Autopilot won’t work effectively and there is a danger of collision with the launch vehicle.

Because IMU failure is quite credible, the analysis reveals that it is critical that the EU be able to abort the launch. In the design described above, this is done by relying on the SMS to monitor missile status reports. This was considered too critical a requirement to trust to software, and the recommendation was made that the system design be modified to include a Firing Interlock (a latched relay which can be triggered by the EU to disable the firing sequence). As a result, the event tree in Figure 7 was modified to replace “software detects & SMS blocks firing?” by “software detects and triggers firing interlock”. The event tree has the same shape, but now the likelihood of the firing branch occurring will be far less. Note that there is now also the need to add a new mode "Abort launch" which triggers the Interlock handler.

FAILURE PROPAGATION

The next step in the process was to look deeper into the system (software) architectural design to identify more precisely how faults propagate through the design, and to identify possible common causes (i.e., failures that would cause one or more of the protective measures to fail). The purpose is to identify the critical safety requirements at component level, and
software safety requirements in particular. In the full method, this analysis is also needed in order to determine the conditional probabilities of events in the detailed cause-and-effect model, so that quantitative targets can be assigned, for expressing the criticality of the requirement (roughly, the tolerance of the overall system design to failures of that requirement).

Figure 8. Software architecture (part) for the EU

The method used was Fault Tree Analysis (FTA) [Roberts 1981] applied to a software architectural model. The latter was defined in the RTN notation for real-time networks [Paynter 2000], which evolved out of the MASCOT design notation widely used in the UK MOD [Simpson 1986].

Figure 8 presents part of the EU software architecture, showing the interfaces with the IMU, the Aircraft, the Actuators and the Firing Interlock, and the primary functional chains in the pre-launch mode. For example, the functional chain from the IMU input to the Actuators output takes the form of a sequence of activities connected by signals that propagate a trigger along the functional chain in “real-time”.

Figure 9 shows the fault tree for the “software fails to trigger firing interlock” event in the (modified) event tree described above. FTA is a laborious process requiring a high degree of domain expertise and close familiarity with the design being analysed.

We found that this process could be simplified significantly by first systematically generating a list of possible component failure modes as follows: for RTN activity components, simply note that
the activity can fail; for each of the other RTN component types (pool, channel, clock, etc) use a HAZOP-like procedure, applying keywords to each component type [MOD 00-58]. For example, for a pool, the following failure modes would apply: 

- omission (value does not get changed); 
- value/detectable (infeasible value change); 
- early (feasible but incorrect value change); 
- late (delay before value gets changed).

**Figure 9. Fault tree for “software fails to trigger firing interlock”**

Each of the possible causes gives rise to a software safety requirement, such as verifying that the interlock handler gets scheduled and verifying correctness of the interlock handler algorithm. Perhaps unsurprisingly, the analysis revealed that mode control is a critical common-mode failure in the design. Consideration of early transition to launch mode reveals that starting the Separation Autopilot when the missile is in the aircraft body cavity may cause gouging of the body cavity or the
missile to jam in the cavity due to missile fin movement. This revealed a safety requirement in that the fins must not be activated, and the mode not changed to Launch, before the missile is clear of the body cavity.

**SUMMARY AND CONCLUSIONS**

In summary, the paper illustrates the application of a process for deriving software safety requirements to an embedded control system for a defence-related application. The process developed progressively more detailed descriptions of the design of the system, with different hazard analysis techniques applied at different stages.

At the system concept level of design the design descriptions and analysis techniques were simple, which made them well suited to workshopping with multiple stakeholders simultaneously, and gave good feedback on the desired balance between functional safety and mission effectiveness. At the software architectural level of design the techniques were more tedious and required good familiarity with the design. The lower level analysis was aided significantly by on-the-fly development of a HAZOP-like procedure for identifying possible functional failures by RTN component type. Most of the analysis reported here was achieved in a 3-day workshop with system designers and stakeholders (writing it up took longer!). Each step in the process revealed or clarified safety requirements that had not been apparent to the system designers and resulted in modifications to the design.

The project stopped short of deriving quantified safety requirements, which was the purpose of the full process described in [Lindsay 2000]. We believe that quantification would be possible for this example, but would require tool support before it is practicable.

The trial illustrated that the process enables closer integration of safety analysis into design early in system development, even as early as system concept. This in turn enables design-level safety requirements to be revealed early, at a stage when trade-offs can be discussed and assessed by stakeholders, before detailed design begins. This is a particularly good way of reducing programmatic risk associated with certification, since certifiers can be brought into the loop earlier. Partly as a result of this trial, the MOD is now undertaking a complete rewrite of its system safety standard [MOD 00-56].

The models developed during the process improve traceability of safety requirements. This makes them an excellent basis for the system safety case and for deriving system safety testing requirements. They are particularly valuable for through-life reassessment of risk, for example if components or subsystems fail to meet their original integrity requirements, or when engineering changes are planned. Being product-based, the approach is a very useful complement of the process-based approaches to development of safety cases described in standards.

**ACKNOWLEDGEMENTS**

The authors gratefully acknowledge the assistance of Bob Born and Steve Paynter of Matra BAE Dynamics in carrying out the case study, and to Colin O’Halloran of DERA (now Qinetiq) for organising the trial. Bob and Steve provided the missile schematic and the RTN diagram. The first author’s part in the trials was financially supported by the UK Defence Evaluation and Research Agency (DERA), Malvern while the author was visiting the University of York as part of a UK EPSRC funded project led by the second author.

**REFERENCES**

ARP 4754. Certification Considerations for Highly-Integrated or Complex Aircraft Systems, Society of Automotive Engineers, November 1996


Def(Aust) 5679, Australian Dept of Defence Standard: The Procurement of
Luqi and Broy, M. (editors), Special Issue on Engineering Automation for Computer Based Systems, Science of Computer Programming 42(1), 2002
MOD 00-56. UK Ministry of Defence Standard 00-56: Safety Management Requirements for Defence Systems. 1996

ABOUT THE AUTHORS

Peter Lindsay is Boeing Professor of Systems Engineering in the School of IT&EE at the University of Queensland. He has held academic and research positions at the University of New South Wales, the University of Manchester, UK and the University of Illinois at Urbana-Champaign, USA. He has more than sixteen years experience in formal aspects of software and systems engineering. He is a member, and past chair, of the Australian Computer Society's National Technical Committee on Safety Critical Systems. His current research interests include techniques for the analysis, development and assurance of trusted software-intensive systems.

Contact details: c/o School of IT&EE, University of Queensland, Qld 4072.
Email: Peter.Lindsay@itee.uq.edu.au
Web: http://www.itee.uq.edu.au/~pal/

John McDermid is Director of the Rolls-Royce University Technology Centre at the University of York in the UK, and co-Director of the BAE Systems Dependable Computing Systems Centre. The work in these two centres was a significant factor in the award of the 1996 Queen's Anniversary Prize for Higher and Further Education to the University. He is author or editor of 6 books, and has published about 250 papers.

Contact details: c/o Dept of Computer Science, University of York, Heslington, York YO1 5DD, UK.
Email: John.McDermid@cs.york.ac.uk
Web: http://www-users.cs.york.ac.uk/~jam/