Software Design Decision Vulnerability Analysis

P G Avery*, R D Hawkins †

**Thales UK, UK, email: phil.avery@uk.thalesgroup.com, †The University of York, UK, email: richard.hawkins@york.ac.uk*

Keywords: software, safety, design, decision, analysis.

Abstract

Software is a key part of today's increasingly complex safety systems. There are many techniques that are available to identify system hazards and hazardous software behaviour. An identified gap in these techniques is the analysis of decisions performed during design and development that can potentially increase the risk to safety of the system due to vulnerabilities introduced by the chosen solution. In this paper we propose a method to record and justify design decisions, identify the vulnerabilities of each design decision and recommend further targeted analysis and mitigation to control those vulnerabilities.

1 Introduction

Safety critical and safety related software systems are becoming ever more complex. An example of this is the civil aircraft industry where better architectural design allows development of a core product using a set of generic functional components that can be used for several aircraft types. Approaches such as Integrated Modular Avionics (IMA) and Fault Detection and Isolation (FDI) software also improve aircraft availability that enables operators to continue flying the aircraft with a number of faults until the next scheduled maintenance service. Such approaches however add layers of complexity to what are already complex systems that are difficult to understand and analyse with confidence.

In this paper we first consider the sufficiency of current commonly used hazard analysis techniques in identifying hazardous behaviour and failure conditions in complex software systems. Based upon these findings we present a new additional method, Design Decision Vulnerability Analysis (DDVA). DDVA is a systematic method to record and justify design decisions based upon an assessment of the vulnerabilities they introduce into software, and the identification of further requirements and additional analysis that may need to be performed. DDVA is not a replacement for existing hazard analysis techniques, neither is it a full solution to the problem of analysing complex systems, but it is a method that fills an identified gap with existing techniques. It is also a method that can be used to assess where analysis using existing well proven techniques can be best targeted based upon design decisions and system vulnerabilities.

This paper is structured as follows. Section 2 discusses sufficiency of existing techniques for analysing complex software. Section 3 describes the DDVA method. Section 4 describes the application of DDVA in practice within industry on an active development that identified a potential vulnerability of a design decision. Section 5 provides the conclusion and discusses future work.

2 Assessment of Existing Software Analysis Techniques

To supplement the authors' own experiences of analysing complex software systems, we applied a number of techniques to several publically reported incidents and accidents attributed to software [1]. In most cases the public formal accident reports, written by the relevant appointed accident boards, were used as the source data for the case studies because the investigators had access to the original design information, data and engineers. These incidents and accidents included complex systems such as Ariane 5, Mars Polar Lander (MPL), Hatch Nuclear Power Plant, Boeing 777-200 and a Patriot Missile Defence System. The most commonly used analysis techniques including Functional Failure Analysis, Fault Tree Analysis, Event Tree Analysis and HAZOP were chosen. These were applied to assess if they would have been able to identify, during system development, the hazardous behaviour that resulted in the unsafe events.

The loss of Ariane 5 was attributed to a defect with alignment function software that is used to measure horizontal velocity [2]. A decision had been made to re-use this software from Ariane 4 in Ariane 5. The performance of the two vehicles and the different alignment procedures used during and after launch are however different. The greater horizontal velocity value experienced due to the increased performance of Ariane 5 caused a software exception within the re-used function ultimately resulting in catastrophic loss of the vehicle.

The most plausible reason for the loss of the MPL was a premature shutdown of descent engines during the landing phase onto the surface of Mars [3]. There were many conflicting requirements imposed on the developers of the software functions used during the landing phase of the mission that led to decisions based on compromise. These decisions increased the vulnerability of the system to premature shut-down of the descent engines due to erroneous touch-down indications from the deployment of the landing legs from the stowed position at an un-survivable altitude.

The shutdown of the Hatch Nuclear Power Plant, although demonstrating correct fail-safe behaviour, was an expensive error [4]. It was due to the decision to connect a business computer network to the primary control system with no obvious assessment of the potential impact. The vulnerability of the power plant to unnecessary shut-down due to errors with a non-safety related system was increased and proven.

An in-flight upset on a Boeing 777-200 was caused by the failure of the FDI system to correctly re-configure the set of accelerometers to use after a second sensor failure [5]. The first failure of an accelerometer had occurred four years previously and this sensor was correctly isolated as part of the fault masking design, but following the failure of a second accelerometer the re-configuration allowed the previously failed accelerometer back into the sensor chain. One of the causes was the failure of the FDI system to read the sensor health status following a power off/on cycle. The failed re-introduced sensor, along with a second latent software error that allowed the sensor input to be used without adequate validation checking, caused disengagement of the auto-pilot and simultaneous and conflicting over speed limit and stall alerts to the crew. The decision to use such a fault masking tactic made the system vulnerable to latent errors that triggered hazardous behaviour.

The failure of the Patriot system to track, identify and intercept the Scud missile that hit a US Army barracks killing 28 soldiers and injuring 98 was due to a software problem in the Weapon Control Computer [6]. The method used to measure time caused inaccuracies in the target tracking function that increased over time since the system was last reset. The system was originally “mobile” that was frequently reset or power cycled and was very effective. The decision to make it a static “permanent” form of protection for this installation increased the length of time that the system was likely to be operational between power cycles such that the error became large enough that Scud missiles were no longer identified and were not intercepted.

Our case studies concluded that current techniques are adequate for hazard analysis of complex systems although there is some room for improvement. The main gap identified by the case studies was that design decisions are not adequately recorded or justified and the impact and vulnerabilities that they may introduce that can cause hazardous behaviour, are often not analysed. The accident reports studied often cited the lack of analysis and justification of design decisions as a cause that in some cases directly contributed to the incident or accident. Critical design decisions had been taken, but there was no record of why those decisions were taken or if the vulnerabilities were considered.

3 The Design Decision Vulnerability Analysis Method

DDVA provides a method to record and justify design decisions. For each design decision the vulnerabilities are identified and assessed and the results recorded. The assessment of each vulnerability may identify that further controls or mitigation are needed. These new requirements are recorded. It may not be possible to immediately identify new requirements or mitigations because further detailed analysis is required. Such additional analysis should also be recorded as an action for a further activity. The output of the DDVA process is a living log or table of design decisions that record the iterative identification of vulnerabilities, their analysis and derivation of requirements, mitigations and or actions. The output of the process is a table that has the following columns:

Design decision: A design decision is an informed choice made from an analysis of available solutions that is required to enable system design to progress to the next stage of development.

The process of requirements analysis and hazard analysis is likely to identify conflicting requirements and these have to be resolved before moving onto the architectural design phase. Decisions to select the tactics and design patterns to use will be required at the architectural design phase. Only when this is complete can the detailed design progress where further lower level decisions will be required including choice of software language, operating system and software design methodology. This method is not generally intended to record the decisions made at the implementation level because there will be too many of these performed on a daily basis during production of the code that would make the recording process too onerous. However if such decisions do have vulnerabilities that could impact previous analysis or, a general policy decision is required to ensure consistency with the handling of, for instance, divide by zero conditions or software exceptions or errors, then these should be recorded.

Justification of design decision: The rationale behind the design decision. This should be a summary and does not need to list the options that were assessed before the final design decision was made. In a safety case it is such justifications that should be challenged, therefore this has to be considered when providing the rationale. A reference to the outputs of other methods used should be provided if appropriate.

Vulnerabilities: The design decision may increase the vulnerability of the system to threats or hazards. The decision may also impact and reduce the effectiveness of mitigations identified by previous analysis. In assessing potential vulnerabilities the impact of the design decision on system function, data flow, concepts of execution, use of resources, timing, performance, architectural modularity, partitioning, compatibility and re-use should all be considered. The vulnerabilities are therefore not necessarily failure modes or failure conditions but may be a side effect of the design decision. The vulnerabilities are a potential trigger for further hazard analysis. Every vulnerability for the design decision should be listed and assessed.

Additional requirement(s) or justification if none: For each vulnerability all additional requirements or mitigations that can reduce its risk should be identified. Assumptions should also be included. The principles of As Low As Reasonably Practicable (ALARP) should be considered. It is possible that there are no practicable mitigations for the identified vulnerability. This should be stated here and justified. Justification may mean an action to assess any increased residual risk to the system.

It may not be possible to complete this until additional analysis and further design decisions have been performed.

Actions: This column is used to identify further activities or analysis that may be required including the verification of assumptions identified. Actions may include the recommendation to perform specific hazard analysis techniques to parts of the system or to review and update previously performed analysis.

The tabular output of the method, described above, should record all design decisions made during the life-cycle of a product. The table or log would grow as development progressed. Like a hazard log it can be updated throughout the life-cycle of the product and should also be understandable to all stakeholders. Following the DDVA process would ensure the history and reasoning behind the design decisions and mitigating requirements would not be lost. Re-use of software that has been subject to DDVA enables the vulnerabilities to be assessed prior to their re-use because the rationale can be challenged by the attributes of the new system. It can therefore be used as evidence in safety arguments that the re-use of components has been analysed. It provides the story of how the design was derived and will answer many questions raised towards the end of a project such as “why was it done that way?”.

4 Practical Examples of using DDVA

This section provides examples of the use of the method to analyse design decisions that would have been required for the Boeing 777 and Ariane 5 case studies and a practical example from current aircraft development. The method would have been equally useful in analysing the design decision to connect a business system to the primary control system of the Hatch Nuclear Power Plant, the Patriot system for its chosen method of measuring of time and change in operational requirement and to analyse the use of leg sensors known to produce erroneous touch-down indications on the MPL.

The design decision relevant to the Boeing 777 incident was how to manage and record a failed sensor or component for potentially a long period of time until a scheduled maintenance activity. An example of using DDVA to record and analyse the vulnerabilities of this design decision is provided in Table 1. This example demonstrates that concentrating on a specific design decision allows better assessment of the vulnerabilities of that decision. In this case power cycling or equipment updates could cause a loss of sensor serviceability status, a cause of the Boeing 777 incident. Table 1 also provides an example of how DDVA can be used to record new requirements to control or mitigate the vulnerabilities and also the recording of actions that identify further analysis that should be performed. Those actions should be managed to closure before the design decision can be ratified. Closure of actions can be recorded within the table by reference to the action activity output evidence.

Ariane 5 had development cost and schedule pressures and so the re-use of software from a previous system was a design decision. Table 2 is the initial analysis of vulnerabilities of the decision to use the alignment function from Ariane 4. It demonstrates that in the early stages of development the method can be used to identify potential vulnerabilities of a specific decision and to recommend actions for further analysis. The additional requirements and mitigations are still to be defined in this example and, after completion of the actions, these can be identified and the justification for the design decision improved. An outcome of this design decision may be that the amount of analysis required, potential additional mitigations impacting other existing or new software that interface to the alignment function and lack of credible service history may need more effort than writing a new alignment function.

These two examples of using DDVA are based on case studies of reported incidents and accidents where the causes were known and well reported. We recognised that the DDVA should also be evaluated on a current development to ensure that it adds value in practice.

The system used for this evaluation is a Flight Program function for an Unmanned Air Vehicle (UAV) that was in development during the evaluation of DDVA and is now in-service. The Flight Program function compares the current position of the UAV against that required by a flight plan, defined by a set of way-points, and determines corrective commands for the flight surfaces via actuators to ensure that the vehicle follows the flight plan. The Flight Program is executed by a Real Time Operating System (RTOS) via an infrastructure layer. This layer provides independence between the Flight Program and the hardware.

The RTOS includes a Board Support Package (BSP) that boots-up the processor, starts the drivers and system features and starts the real time scheduler. However for this application the Flight Program requires a Transmission Control Protocol/Internet Protocol (TCP/IP) interface to enable it to communicate with other aircraft systems, an interface not provided by the certified RTOS chosen. A design decision was to use a certified version of the RTOS, that has the advantage of providing a deterministic reduced subset of the standard version, but required the addition of the interface for TCP/IP provided by drivers from the standard version of the RTOS. The effect of this is that the BSP needs modification to start the TCP/IP drivers on start-up.

A previous system level design decision for the overall flight control system is that it shall be deterministic. To achieve this, a requirement was defined that in the event of particular failures that cause software exceptions to be raised the processor shall be reset. Justification for this is that an exception is likely to be due to bad data and a reset will re-start the Flight Program with fresh data. During the reset a reversionary function running on a different processor, with much reduced functionality than the Flight Program, takes over control until the Flight Program has re-started. This requires a quick start-up time so that the vehicle is in a reversionary mode for a short period of time. The BSP, as supplied, takes longer than the quick start-up time required and therefore needed further performance improvements in addition to the requirement to enable the TCP/IP drivers.

Table 3 demonstrates the use of DDVA in evaluating the design decision to modify the BSP to include the TCP/IP drivers and reduce boot-up time. One of the vulnerabilities of this decision, shown in Table 3, is that too much could be removed from the BSP in achieving the target boot-up time leading to a loss of vital initialisation activity. In this case the original BSP requirements were poorly defined and the requirement to reset memory was missed.

The three examples demonstrate the use of DDVA in order to identify vulnerabilities that are not just failure modes. Most design decisions are sensible and technically sound. However such decisions can introduce vulnerabilities. The stakeholders need to know the risk that is being taken due to design decisions, many of which will be a compromise between the available solutions. The use of the method challenges the design decision and therefore improves the final solution and ultimately the safety argument. DDVA, by identifying vulnerabilities rather than only failure modes, may even result in reversing or changing the original design decision. The DDVA method resolves the gap identified by the case studies following evaluation of the existing hazard analysis techniques.

5 Conclusions and Future Work

The conclusion of the case studies identified that the incidents were all due to the consequences of design decisions creating new hazards or undermining previous safety analysis and reducing the effectiveness of previous identified mitigation. In many cases design decisions made during development, a change in system functionality or operation, were not analysed for their potential impact to increase the vulnerability of the system to hazards. This paper has presented DDVA as a method to record design decisions, justify those decisions, to identify the vulnerabilities and to recommend further mitigation and/or hazard analysis.

The paper demonstrates the use of DDVA on examples of reported incidents and accidents and a practical example performed during the development of a new aircraft.

DDVA does not add complexity to the safety process because the recording of design decisions is already a requirement of most standards and is required evidence in safety cases. The method is an enabler of concentrating the mind of the analyst and integrating design decisions with safety analysis providing a safety case compliant method of recording the decisions, linking those decisions to analysis already performed and identifying where further analysis is required. Complex systems cannot be fully analysed but DDVA can be be used as an aid to better target further analysis.

DDVA requires no specialist tools other than standard PC Microsoft Word or Excel type tools. The guidance on the use of the method does not require any specific training for those with an awareness of hazard analysis techniques. It is not dependent on specific system design, architectures or hazard analysis techniques but should be used in addition to those existing techniques.

DDVA is considered to be suitable for other disciplines, not just software, and at all stages of system development. It is also considered to have potential for use in other non-safety related development such as security.

DDVA is a new method and future work will include practice of its use at Thales UK for future projects.

Acknowledgements

The authors acknowledge the support provided by Thales UK and the University of York.

References

[1] Phil Avery, “Software Safety and Hazard Analysis Techniques”, Project submitted for the MSc in Safety Critical Systems Engineering, University of York, September 2013.

[2] Prof. J.L.Lions, “Ariane 5 - Flight 501 Failure – Report by the Inquiry Board”, European Space Agency.

[3] “Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions”, Jet Propulsion Laboratory (JPL) Special Review Board, JPL, California Institute of Technology. 22 March 2000.

[4] Brian Krebs (June 5, 2008), Cyber Incident Blamed for Nuclear Power Plant Shutdown [On-Line], Washington Post. Accessed on 05/07/2014. Available at http://www.washingtonpost.com/wp-dyn/content/article/2008/06/05/AR2008060501958.html

[5] Australian Transport Safety Bureau (ATSB) Transport Safety Investigation Report, “In-flight upset event 240 km north-west of Perth, WA, Boeing Company 777-200, 9M-MRG, 1 August 2005”, Aviation Occurrence Report 200503722 – Final.

[6] The Honorable Howard Wolpe, Geneal Acounting Office (GAO) Report on Patriot Missile Software Problem, February 4 1992, Accessed on 05/07/2014. Available at: http://www.fas.org/spp/starwars/gao/im92026.htm

Design Decision	**Justification of Design Decision**	**Vulnerabilities**	**Additional Requirement or Justification if None.**	**Actions**
Sensors could exhibit erroneous or erratic failure conditions. Therefore when a sensor is detected as having failed it should be latched as failed and not used for the navigation solution until a maintenance activity resolves the failure.	Latching a sensor as 'out of service' ensures that erroneous or erratic failure behaviour of sensors are not used in the navigation solution to generate erroneous or erratic flight commands. Such erroneous behaviour could be due to the changing environment that an aircraft flies through and re-using a sensor with known erroneous or erratic behaviour is considered to unnecessarily increase risk of hazardous flight control. Vulnerabilities are mitigated by ensuring that the 'out of service' data previously stored is checked for corruption before use. Further mitigation is provided by processes followed by the maintainer for resolution of sensor failures and software updates.	Erroneous or erratic common mode failure that causes two or more sensor to fail will latch multiple sensors as 'out of service' and fail the Air Data Inertial Reference Unit (ADIRU).	No further functional mitigation considered practicable. <u>Assumption:</u> Redundant ADIRUs that cannot exhibit common mode failure.	Perform common mode failure analysis for sensors. Verify assumption.
		ADIRU fails to read 'out of service' sensor data.	The 'out of service' sensor data shall be read by the ADIRU following a reset or power up.	A facility for the maintainer to reset the sensor 'out of service' status is required.
		The 'out of service' sensor data is corrupted.	The 'out of service' sensor data shall be protected by checksum. On reading the 'out of service' data the checksum calculated from the data shall be compared to the stored checksum and if they do not match the ADIRU shall be failed.	
		Maintainer incorrectly resets 'out of service' status of failed sensor.	No further functional mitigation considered practicable.	Ensure that the maintainer process for diagnosing and resolving sensor failure is adequate.
		Software or hardware updates within ADIRU could cause 'out of service' data to be lost.	Update of software shall not overwrite or corrupt 'out of service' sensor data. Software shall be backward compatible with respect to recorded data on the ADIRU. <u>Assumption:</u> ADIRU software can be updated while installed in aircraft.	System level design decision required to confirm assumption.

Table 1 – Example of using the DDVA method as part of the Boeing 777 case study.

Design Decision	Justification of Design Decision	Vulnerabilities	Additional Requirement or Justification if None.	Actions
Re-use alignment function from Ariane 4.	Ariane 4 alignment function has good pedigree, service history and is a cost effective solution for Ariane 5.	Ariane 5 has different behaviour including trajectory and performance. The alignment function may have incompatible data types and units.	The alignment function shall adhere to the Ariane 5 interfaces and parameter value ranges, precision, data type and units.	Compare the interfaces and required data ranges and units between Ariane 4 and 5. Perform HAZOP for all data relating to the alignment function within the context of Ariane 5.
		Ariane 5 has different alignment procedure therefore scheduling of function may require changing.	The alignment function shall not execute post launch.	Compare the alignment procedures between Ariane 4 and 5.
		Re-used software not compatible with target platform.	The alignment function shall be rebuild using the target platform environment including operating system and compiler.	Compare target platforms between Ariane 4 and 5 including operating system and software compilers and versions of those items to ensure compatibility or identify updates required.
		Service history and pedigree are only relevant to Ariane 4 and due to differences with Ariane 5 may not provide adequate re-use safety argument in safety case.	The re-used alignment function shall be reverse-engineered to the Ariane 5 development processes.	The safety case is required to validate any claims of service history and pedigree by identifying the differences in execution environment between Ariane 4 and 5 to assess if argument is credible.

Table 2 – Example of using the DDVA method as part of the Ariane 5 case study.

Design Decision	Justification of Design Decision	Vulnerabilities	Additional Requirement or Justification if None.	Actions
The BSP provided with Operating System requires modification to enable the TCP/IP driver and to reduce boot-up time.	Boot-up is required to be completed in < 10 seconds. TCP/IP drivers require enabling on start-up. Modification of the BSP is lower risk than creating a new BSP.	Too much is removed from BSP resulting in loss of initialisation, loss of RTOS features that are required and incomplete or non-deterministic initialisation of resources and drivers.	The BSP shall: Initialize the processor Initialize the bus Initialize the interrupt controller Initialize the clock Initialize the Random Access Memory (RAM) settings Reset RAM Enable TCP/IP driver Enable bus driver Enable timer driver Enable serial driver Enable non-volatile RAM driver Configure the segments Load and run bootloader from flash	Perform analysis of what is the minimum required initialisation by the BSP. Use Functional Failure Analysis (FFA) for omission of function and HAZOP for loss or incorrect data flow.

Table 3 – Example of using the DDVA method as part of the UAV Flight Program case study.