Safety is no accident
Kesseler, Ernst

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2008

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
1 Introduction

The theme of this work is achieving safe software. To put this work into context, first some background information on safety is provided, followed by some considerations on software for embedded applications.

For humans, air transport, i.e. flying, is an inherently unsafe activity. During the early years of powered flight flying was an experimental and indeed dangerous activity. By a persistent and dedicated effort during several decades, civil air transport has achieved an enviable safety record which is reflected in a good safety image with the general public. The former is a good reason to include the aerospace domain as the practical application domain for the work. To increase coherency of the research, all presented work is related to a single domain, for which aerospace has been selected. Given the potential similarities between some domains with safety concerns, part of this work may be interesting for such other domains as well. Therefore a chapter is included on the relation between various standards for software safety certification.

After an elaboration on safety in section 1.1, the information technology perspective is provided in section 1.2. Safe software differs in some respects from general domain software, but other general domain trends do hold. Section 1.3 provides a synopsis on air traffic management. From this combined background of safety and information technology in section 1.4 the research questions are derived, which this thesis will address. Subsequently section 1.5 describes the approach to answer these questions. An overview of the thesis is provided in section 1.6. This introduction concludes with section 1.7 listing the related work the author has published. All work relates to software systems for use by humans, so where relevant those human-computer aspects are taken into account.

1.1 Some observations on safety

The following tables provide some information on the risk of fatal accidents, which result when safety is compromised. Table 1-1 gives an overview of the risk to the general public of various classes of fatalities caused by unintentional accidents. In order to improve the intuitive understanding of the data, the yearly odds and life time odds of dying of unintentional accidents are provided. To facilitate comparison with the other tables and literature, also the yearly fatality rate per 100 000 persons
is included. Table 1-1 contains the most recent information available at the time of writing and is derived from the USA (NSC, 2005).

<table>
<thead>
<tr>
<th>Type of accident</th>
<th>One year odds</th>
<th>Yearly fatality rate per 100 000</th>
<th>Life time odds</th>
</tr>
</thead>
<tbody>
<tr>
<td>All causes</td>
<td>1 755</td>
<td>57,0</td>
<td>23</td>
</tr>
<tr>
<td>All transport modes</td>
<td>5 953</td>
<td>16,8</td>
<td>77</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>47 273</td>
<td>2,1</td>
<td>612</td>
</tr>
<tr>
<td>Car</td>
<td>13 645</td>
<td>7,3</td>
<td>177</td>
</tr>
<tr>
<td>Bicycle</td>
<td>375 412</td>
<td>0,27</td>
<td>4 857</td>
</tr>
<tr>
<td>Civil air transport</td>
<td>440 951</td>
<td>0,23</td>
<td>5 704</td>
</tr>
</tbody>
</table>

Table 1-1 shows that transport contributes significantly to the accidental fatality rate for the general public. The general public considers the odds of dying due to civil air transport relatively low, which is supported by the data in Table 1-1. These fatality data also indicate that air transport is relatively safe compared to other transport modes, despite the inherent risks of air travel and the many technical systems and organisational arrangements needed to make civil air travel feasible for current levels of air traffic. The latter contributes to making air transport an interesting application domain for the presented work.

However most people are exposed to the risk associated with being pedestrian and occupying a car on a daily basis, while only occasionally being an aircraft passenger, if at all. As the fatality rates in Table 1-1 are influenced by a person’s behaviour, Table 1-2 provides some information on fatality rates based on selected transport modes. The information is provided both for the distance travelled and for the travel duration. The data are taken from (ETSC, 2003), are valid for the 15 pre-2004 European Union (EU) member states and again are the most recent available at the time of writing.

From Table 1-2 it follows that air transport and rail share a similar fatality rate per distance, which is low when compared to the other transport modes. However there is an eight times difference by time due to the difference in travelling speed. Although car travel is nine times safer then walking for a given distance, on an
hourly basis the fatality rates are similar. Table 1-2 reinforces the perceived safety of air transport per distance travelled, but on an hourly basis the difference with some other transport modes is much less impressive.

Table 1-2. Fatality rates of selected transport modes, per distance and per travel time.

<table>
<thead>
<tr>
<th>Transport mode</th>
<th>Fatalities per 100 million person kilometres</th>
<th>Fatality rate per 100 million person travel hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Road total</td>
<td>0,95</td>
<td>28</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>6,4</td>
<td>25</td>
</tr>
<tr>
<td>Bicycle</td>
<td>5,4</td>
<td>75</td>
</tr>
<tr>
<td>Car</td>
<td>0,7</td>
<td>25</td>
</tr>
<tr>
<td>Civil air transport</td>
<td>0,035</td>
<td>16</td>
</tr>
<tr>
<td>Rail</td>
<td>0,035</td>
<td>2</td>
</tr>
</tbody>
</table>

A plane crash and the resulting fatalities are high profile, generating much media attention. Often such events generate headlines on an international scale. In contrast, cars yearly cause 41 000 fatalities in the 15 pre-2004 EU member states and hardly generate any media attention. Still the general public is aware of the relative risks of road travel with respect to the comparatively safer air transport. Despite the good air travel safety record as well as the good perceived safety by the general public, both the European vision (Argüelles et al., 2001) and the US vision (Walker et al., 2002) aim at a fivefold increase of air transport safety. Note that EU policies also aim to reduce car fatality rates per distance travelled by 50% between 2000 and 2010 (ETSC, 2003).

Apart from the perspective of the general public (Table 1-1) and the traveller (Table 1-2) the last safety perspective presented is from the workplace, i.e. the pilot’s perspective. Table 1-3 provides an overview of the fatality rates for various categories of the workforce. All data relate to 2004 and are obtained from the (DoL, 2005) with the exception of the mining data obtained from (MSHA, 2004). All figures exclude military casualties.
Table 1-3. Fatality rates for selected categories of the workforce per year.

<table>
<thead>
<tr>
<th>Profession</th>
<th>Fatality rate per 100 000 workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilots</td>
<td>92.4</td>
</tr>
<tr>
<td>Roofers</td>
<td>41.2</td>
</tr>
<tr>
<td>Underground mining</td>
<td>33.2</td>
</tr>
<tr>
<td>Truck drivers</td>
<td>29.5</td>
</tr>
<tr>
<td>Construction</td>
<td>13.2</td>
</tr>
<tr>
<td>Total workforce</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Pilots experience the highest fatality rate of any profession in the USA, more than double the fatality of the intuitively unsafe activities of roofers and more than triple the fatality rate of truck drivers. By regulations, pilots are only permitted to fly one hundred hours per month. Truck drivers easily are exposed the double amount of working hours and consequently risks, so the hourly rate for pilots is even more unfavourable. Still, according to (Duke, 1999), only one in twenty pilots experiences a fatal accident in his/her entire career. The objective is to reduce that to one in fifty careers by 2009 (Duke, 1999). So while air transport is safe from the perspective of the general public and from a passenger perspective, it remains risky from a professional point of view. Another consequence of these observations is the need for permanent vigilance during day-to-day operations for an event which is unlikely to occur, even when considering an entire professional career. Well defined and safe software might help.

In order to facilitate putting Tables 1-2 for the travelling public and 1-3 for the workplace into perspective, the fatality risk for road travel is forty times that of working and twelve times the risk of staying at home (ETSA, 2003). Even rail travel, the safest transport mode per hour, is still twice as risky as working. Road fatalities represent about one quarter of all work related fatalities (DoL, 2005). All of this illustrates that making transport safer is relevant, also for air transport, so the subject of this research has societal relevance.

Air transport consists of many interacting systems, both airborne and on the ground. The resulting complexity makes the work challenging on top of being relevant. At the introduction into service in 1995, the Boeing 777 already contained certified software in 76 systems, and the number of systems has grown since. Another consequence of the safety perspectives provided above is, that even though the occurrence of a single failure in a single system might be quite rare, the overall
occurrence for the entire civil aircraft fleet still is considered to be too high. Given
the above, observing the behaviour of a system in its intended environment for a
limited amount of time is unlikely to provide sufficient confidence in its safe
behaviour. This explains the emphasis safety certification requirements put on
compliance with procedures of above measuring safe performance for the more strict
safety criticality levels.

When analysing accidents, in many cases accidents happen because of an
unfortunate combination of events. Already in 1931 over half a million workplace
related accidents have been studied with the aim of prevention (Heinrich, 1931). The
results are depicted in Figure 1-1.a. Note that the definition of the lower layers, i.e.
the more frequent events, includes the more serious events of the upper layers. The
US National Safety Council reflects this in their definitions of accident and incident.
An accident is an act not planned, nor wanted that resulted in personal injury or
property damage. An incident is an event not planned or wanted that adversely
affects the completion of a task. The incident may cause personal harm or damage
(NSC, 2001), which implies that the incident definition includes all more serious
accidents.

It was found that for each workplace related fatality, there were 29 accidents with
minor injuries or property damage and 330 incidents. The one highly visible fatality
can be depicted as the top of a safety iceberg. A general strategy resulting from this
finding is that by reducing the causes of the more readily observable incidents and
minor accidents, also the fatal accidents can be reduced. Due to the higher frequency
of incidents, they are considered a useful safety indicator. This is referred to as
common causation.
In 1969 a similar safety analysis has been performed by Bird for 1 753 498 incidents in 297 organisations (Bird, 1969). The results are that for each serious injury there are 10 minor injuries, 29 cases with property losses and 600 incidents, as shown in Figure 1-1b. The concept of the safety iceberg remains, but the proportions between the layers differ, as do the definition of the layers.

In 1992 (Salminen et al., 1992) performed a similar analysis of 105 371 minor injuries, which once again confirmed the iceberg concept still holds for more contemporary circumstances. Again different definitions were used, and different proportions per layer were obtained. The results are given in Figure 1-1c. Given the long period between these three analyses, the iceberg concept may be considered valid. (Salminen et al., 1992) also tested the common causation, by comparing 14 classes of accidents factors. The common causation was confirmed. Similarly (Wright, van der Schaaf, 2000) provide quantified support for the common causation in their analysis from the railway domain.

![Figure 1-2. UK civil aviation authority reporting schema, based on Heinrich safety iceberg.](image)

According to (Duke, 1999), air transport experiences 300 incidents per accident, confirming that applicability of the safety iceberg concept is valid for the domain. Consequently the UK Civil Aviation Authorities base their mandatory occurrence reporting system (Wright, Lyons, 2001) on the Heinrich safety iceberg, as depicted in Figure 1-2. Safety management systems ensure that the air transport sector does continuously learn from incidents. According to the latest analyses (Flight 2006) some categories like controlled flight into terrain are causing multiple accidents. Programmes are being renewed to reduce the occurrence of the deviations leading to such incidents and ultimately, in some unfortunate cases, to accidents. Such
programmes include crew training as well as technical solutions. The latter may involve safe software. The data also indicate that the effort on reducing approach and landing accidents seems to be paying off. Again this includes the use of software, confirming the relevance of studying safe software.

Figure 1-3 provides another way of depicting the safety iceberg for a system with multiple layers of defence. Of the many deviations, most are caught by the system. Some slip through the first layer of defence and lead to incidents. Similarly in some unfortunate circumstances some incidents cause injuries with even fewer resulting in fatalities. (Salminen et al., 1992) report an average of twelve contributing factors per accident. As the general concepts of the safety iceberg and common causation hold for air transport, knowledge to achieve safe software in this domain, as presented in this work, might be relevant for similar safety conscious domains.

The above mentioned observations on safety reveal that software is relevant for safety in several ways. Software can fail, resulting in a deviation which may contribute to an accident. On the other hand, information of various sources can be combined to detect deviations or incidents. In such cases warning to pilots or air traffic controllers could provide a layer of defence to prevent deviations to deteriorate into accidents. Software can also help to monitor daily events to provide advance warning for the rare case which might turn dangerous. Such software analyses normal and abnormal variations for unknown combinations with the potential for hazards.
To summarise, to study certifiable safe software, the air transport domain provides a good case. This work can be relevant for other domains, as safety concepts like the safety iceberg and common causation hold for various domains. Work is included to relate air transport software certification with other certification standards. There is a need to improve domain safety, which implies a need for more safe software e.g. to provide additional safety layers or improve existing safety layers. As software safety extends beyond the implementation phase, the work will include integration, human machine interface and specification issues. This work will concentrate on the more critical class of software whose anomalous behaviour might cause danger, not on software which might inadvertently fail to detect danger.

So after providing some context on safety, the next section will explore safe software for embedded applications, the subject of this work.
1.2 Some observations on software

1.2.1 Software size growth

The hardware capabilities which support the information technology grow continuously. Based on a few years of initial experience with integrated circuits, Moore’s law predicted that hardware capabilities, i.e. the number of transistors, would grow exponentially by doubling in a fixed period. This doubling period is observed to be approximately 2 years. Amazingly Moore’s law was published in 1965 (Moore, 1965) and still holds today, more than forty years later. According to the author, Moore’s law is predicted to hold for another decade (Moore, 2003).

To make use of all these hardware capabilities software is needed. For consumer electronics, the size of the software is also observed to grow exponentially. The software doubling period has a similar size of around 2 years. (Ommering 2004) has shown this to be true for high-end televisions, the first consumer electronics item with a significant amount of software. It is also observed to hold for video recorders (Ommering 2002), as shown in Figure 1-4. This exponential software growth also expected to hold for other consumer items, in which software was introduced at a later date (Ommering 2002). For mobile phones the available, more scarce, data suggest that a similar trend holds, despite mobile phones facing additional challenges in physical size and power consumption to limit exponential software growth. Such challenges do not apply to stationary home equipment powered by the net like TV’s and video recorders. As depicted in Figure 1-4 the doubling period for mobile phones might be slightly longer. The production volume of all these consumer electronic items runs into the millions. Consequently the software investment can be amortised over a large number of units, resulting in acceptable unit costs despite the exponentially growing software size.

The growing software is used to add features to the product as well as to increase user value for the consumer. The latter intends to make it easier for the consumer to satisfy his needs in stead of offering a technical capability and putting the burden of operating the capability on the user. A typical example of the latter is the video recorder. The first video recorders had lots of features, but were notoriously difficult for the general public to operate, let alone exploit all features offered. Later generations provided a user interface designed for improved operating convenience realised by significant amounts of software. For none of these systems safety is an
issue, which makes software production more affordable than for safety critical software, allowing for more rapid software growth.

For professional systems the publicly available data is unfortunately more scarce. Figure 1-5 provides available data for a complex non-safety critical product (Brugman 2003) (a wafer-scanner which is officially referred to as a micro lithography manufacturing system) and a complex safety critical product, a large civil aircraft from one of the major manufacturers (Pehrson 1996). Data of one of the consumer electronic products, mobile phones, is included in Figure 1-5 to allow comparison with the data for consumer electronics depicted in Figure 1-4. For the professional systems all data is for the first systems of a new product generation. The available data for the professional systems do not seem to confirm Moore’s law. In order to be interoperable, part of the mobile phone software like the communication stack needs to be certified. Still mobile phones are not authorised for use in safety critical environments. Repeated failures may harm company image and hence sales, effectively limiting the acceptable amount of software deviations for the entire software configuration. Wafer-scanners have to increase their reliability for commercial reasons, so reliability is expected to become more important for such systems. Apart from being reliable, for economic reasons, of course aircraft need to be demonstrably safe, as stated by their safety certificate i.e. the airworthiness
certification. This certification requires substantial effort, which impedes exponential software growth.

![Figure 1-5. Software growth for safety critical and non-safety critical professional systems (respectively waver-scanner, aircraft) compared with one selected consumer product (mobile phone).](image)

To analyse the nature of the software growth for non-safety-critical professional systems, some information for waver-scanners is provided. The amount of sensors and actuators increased from 50 sensors plus 40 actuators for the first generation, via 150 sensors plus 100 actuators for the second generation to 400 sensors plus 300 actuators for the latest generation (Brugman 2003). Also the amount of subsystems, measured by the number of processors, increased from eight via twenty to fifty (Brugman 2003), which suggests the hardware and the number of subsystems grow exponentially. Exponential hardware growth and slower non-exponential software growth may suggest an opportunity for increased software productivity. As expected software growth in the safety critical area (illustrated by the aircraft data) is lagging the general domain (illustrated by the waver scanner data). This work takes these observations into account.

Already ten year ago, the large civil aircraft for which the data is shown in Figure 1-5, comprised 76 systems, and that number is increasing. Note that for aircraft the number of processors is not an appropriate measure of the number of subsystems. Due to availability requirements caused by the need to safely complete the flight, the more critical systems are implemented on multiplied hardware. More current data
for civil aircraft then provided in Figure 1-5 is not publicly available, as both major suppliers are in the process of developing their next generation aircraft. The military Joint Strike Fighter, with the first deliveries planned in 2008 (JSF 2006), will contain 6 million lines of code (Costlow 2006). This confirms the linear software size growth shown in Figure 1-5. It should be noted that military aircraft are exempt from civil certification, but in order to fly through civil airspace need to carry compliant civil equipment. To complement these scarce data for professional safety critical systems, the next paragraph will provide some additional pieces of information on safety critical professional systems to allow comparison with general domain software growth trends.

For a specific aircraft engine, the size of safety critical software in a controller only increased by five percent in ten years. Like for the entire aircraft, the number of engine systems which include software is observed grow substantially. Typically software is added to subsystems to add features and to add or improve safety layers. This observation supports the view that feature growth is also a driver of the observed growth of safety critical software. Similarly for helicopters, the software size per subsystem remains rather constant during a subsystem generation, but the number of subsystems containing software increases, contributing to an overall growth of safety critical software. A special characteristic of certifiable safe systems is that any modification of the certified software obviously requires re-certification to the same standards with equal rigour as for the original certification (DO-178B 1992). The additional effort required effectively counteracts the “natural” trend for features driven growth and reduces the software update rate with respect to non critical software. Another result of the strict certification requirements is that certified software is made configurable through data files. Subsequent tailoring of the software for a specific aircraft is done through such data files. Another impediment to software growth in certified systems is that the hardware itself needs to be certified. In the general domain, due to Moore’s law, increasing the software size beyond the current hardware limit can be accommodated affordably by choosing a larger chip. In a certifiable safe environment this is not possible as it requires a hardware modification which needs to be certified as well. This puts an incentive on limiting the software size to the available resources within a generation of systems. It also explains why introducing a new function usually means a new dedicated hardware unit is added. Only the new generation of civil aircraft, currently being developed, is going to deploy multiple software applications per hardware unit. This might reduce the influence of hardware limitations on growth of safety critical software.
Software growth facilitates new features. In safety critical environments such features can be used for various purposes. Additional software can monitor hardware systems to provide early warning for maintenance, thereby increasing service reliability. Alternatively, additional software can connect systems allowing more consistency checking between systems to be performed, thereby adding safety layers, or increasing the efficiency of existing safety layers.

Similar to the trend in the general domain, apart from adding features, software growth usually allows a new product generation to provide more user value. For example, in the safety critical air transport domain this translates in commonality between aircraft cockpits (i.e. shielding the aircraft characteristics from the pilots so they can use their same trained capabilities and their license to fly different aircraft) (Airbus 2006), (Adams, 2003).

Civil aircraft production is a small market in terms of the number of units produced. Typically a few hundred aircraft are delivered per year for each of the two main suppliers of large civil aircraft. Aircraft last long, often for decades, so the total amount of civil transport aircraft in use from all manufacturers just passed the twenty-five thousand mark (Kingsley-Jones 2005). This limited amount of delivered products, compared to the general domain, on the one hand limits the amount of product versions. On the other hand, the long in-service life compared to the general domain means the products will experience a significant evolution during their lifetime, increasing the number of variants. The small market size in terms of units also implies that software is a substantial part of the unit costs, limiting software growth.

Another observation from Figures 1-4 and 1-5 is that the size of software for high-end consumer electronics is approaching that of professional systems, like the two presented above. This might imply similar levels of complexity.

1.2.2 Safety certification of software

Already since the start of information technology, some software failures have caused accidents. Back in 1994 (Les Hatton, 1994) already listed many high profile software failures spanning decades. The amount of software is observed to grow and by connecting systems the complexity grows even faster. If nothing changes in the software failure rates, software growth will imply more software deviations. Through common causation software failures may be expected to become more
frequent. The good safety record of air transport, which also uses a large and growing amount of software, may provide lessons for other domains. Conversely the safe software domain may benefit from lessons learned in the general domain like re-use to increase affordability and reduce time-to-market e.g. to enable the additional safety functions or improved usability mentioned above.

The seminal book by (Hatton, 1994), and the many subsequent years of courses based on the same material, provide guidance to improve the writing of safe software based on the combined practice of many. Recently for Java similar guidelines have been produced in (ESA, 2005). Chapters four and five extend this by providing additional information on the software realisation process. According to (Leveson, 1995) incorrect specifications have caused more accidents than software bugs have. Consequently this work extends the mentioned state-of-the-art by addressing subsystem interaction and human design issues in chapters two and three to improve such specifications.

The relatively slow growth of safety critical software size means such software is lacking capabilities which are becoming wide spread in the general domain. One accident mentioned chapter three provides some insight in this. Satellite navigation is widely available and affordable for the general public, especially in cars. Such equipment could have helped improve the situational awareness of one pilot whose loss of it contributed to the Linate accident. However the important message from chapter three is that adding features without designing the systems for the humans who have to use them is not the way to go. This example illustrates the need for more affordable safe software development, based on good human-centred design combined with Commercial Off-the-Shelf (COTS) functions. This is why chapter seven extends the current state-of-the-art by performing an experiment on more affordable software development and COTS for medium safety criticality levels to complete the safety critical work suite proposed in this work.

The following data further justify the need for more affordable and swifter safe software development and hence the relevance of the experiment on COTS in chapter seven. Usually in aircraft each new function is implemented on dedicated hardware in its own subsystem. According to an industrial source (Teksci 2002), the average amount of time needed to realise the custom real time operating system functions for such hardware is 127 man months. A lot of these real time operating system functions are standard, so in principle suited for a COTS approach. In fact many such functions are already based on re-use. The productivity for safe software
is relatively low compared with general domain software. On average, US companies with ample experience with air transport software certification, require 16 man-years of effort for ten thousand lines of code of the second most critical software level. Comparing this productivity with the total amount of software in aircraft already 10 years ago, Figure 1-5, provide an idea of the investment in airborne software. This investment explains the long product lifespan of airborne software, typically a decade (Teksci 2002) as well as the low update frequency, typically 3 to 4 years (Teksci, 2002).

The first certification documents for airborne software already date from 1982, so around 25 years of experience has been accrued in this domain. Combined with the domain’s good safety record, this suggests the domain as suitable for study and experimentation on achieving safe software. To improve the cohesion of this work, all presented work is taken from this domain. Chapter six compares air transport certification with the certification documents of other domains. The next paragraphs provide a very brief introduction into air transport software certification.

The airworthiness certification of aircraft relates to the whole system i.e. the aircraft type, not just to the incorporated software. The airworthiness certificate is provided by a third party (independent from the supplier and the customer) demonstrating that the specified aircraft type is safe to use. Due to the nature of software, a special document (DO-178B/ED12B, 1992) describes what is needed to get the software systems certified. First the software needs to be classified, according to the (worst case) impact of a potential software failure on the safety of the entire system i.e. the aircraft. The aircraft level FAR/JAR AC-25-1309 provides the safety classification into five categories, with a verbatim copy of their wording provided below. The European version of the certification documents is denoted as Joint Aviation Requirements (JAR), while the US version is referred to as Federal Aviation Requirements (FAR). FAR/JAR-25 uses the general principle of an inverse relationship between the probability of a failure condition and the degree of hazard to the aircraft or its occupants. As DO-178B considers qualitative demonstration of software compliance to such high reliability to be beyond the current software technology, the FAR/JAR-25 failure probability in flight hours is provided for information only.

- **Level A:** Catastrophic failure: failure conditions which would prevent continued safe flight and landing, FAR/JAR-25 extremely improbable i.e. $< 1 \times 10^{-9}$, i.e. these failure conditions are so unlikely that they are not anticipated to occur during the entire life all aircraft of one type. Examples of DO-78B level A
software include software to display critical flight information to the pilot and parts of the flight management system to control the aircraft.

- **Level B**: Hazardous/Severe-Major failure: Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:
  
  - A large reduction in safety margins or functional capabilities;
  
  - Physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely;
  
  - Adverse effect on occupants including serious or potentially fatal injuries to a small number of those occupants.

FAR/JAR-25 extremely remote, $1 \times 10^{-9} < \text{hazardous failure} < 1 \times 10^{-7}$. Examples of DO-178B level B include the most critical parts of future European satellite positioning system Galileo.

- **Level C**: Major failure: Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example:
  
  - A significant reduction in safety margins or functional capabilities;
  
  - A significant increase in crew workload or in conditions impairing crew efficiency or;
  
  - Discomfort to occupants, possibly including injuries;

FAR/JAR-25 remote, $1 \times 10^{-7} < \text{major failure} < 1 \times 10^{-5}$. The most critical parts of the aeronautical telecommunication network (for automatic communication between pilots and ground based air traffic controllers) is classified at DO-178B level C.

- **Level D**: Minor failure: Failure conditions which would not significantly reduce aircraft safety and which would involve crew actions that are well within their capabilities. Minor failure conditions may include for example:
  
  - A slight reduction in safety margins or functional capabilities;
• A slight increase in crew workload, such as, routine flight plan changes, or some inconvenience to occupants.

FAR/JAR-25 probable, minor failure > $1 \times 10^{-5}$. A system to record engine cycle data, required to determine maintenance intervals, is an example of a system classified at DO-178B level D.

• Level E: No Effect: Failure conditions which do not affect the operational capability of the aircraft or increase crew workload. A passenger entertainment system falls into this category, as long as it is not connected to more critical aircraft systems.

Both classical (Butler, Finelli 1993) and Bayesian approaches (Rushby 1993) show that for a median failure time of $n$ hours, approximately $n$ hours of failure free operation need to be observed. A year contains 8760 hours, so for level A software $10^9$ failure free hours of operation under realistic test conditions need to be observed, which is more than 114 000 years. This makes observing safe software behaviour for the higher criticality levels impractical. The general domain safety-critical software standard (IEC-61508 1998) confirms this by stating that for a for their lowest criticality level, 95% confidence in correct functioning requires 300 hours of relevant service experience. For a system classified at their highest criticality level, 99.5 % confidence requires 690 000 years of relevant service experience, which is clearly impractical. Consequently DO-178B contains 66 detailed requirements for the software development process, with all requirements applicable for DO-178B level A software. Demonstration of a fully DO-178B compliant software development process is required including verification of each software process. As specification is the first DO-178B software development process, formal methods which require a specification to start with, can provide only part of the evidence required by airborne software certification. In case formal methods are used for automatic code generation, either the code generator needs to be certified at the same level as the generated code or the generated code needs the same verification, including thorough testing, as manually generated code. For well defined functions, like aircraft control, formal methods are used. This thesis focuses on systems interacting with humans for which formal methods have not been applied.

Apart from DO-178B several additional software certification standards are being applied in the air transport domain. In the US for ground-based air traffic management software DO-278 has been derived from DO-178. For this purpose in
Europe the independent Recommendation for Air Navigation Services software (Eurocontrol 2003) has been produced. For the European satellite navigation system Galileo, designed to provide position information with sufficient reliability to be used in civil air transport, a proprietary DO-178 derived standard is being used. In modern cockpits there is a need for a laptop like device, call the electronic flight bag, for which an additional software classification AC-120-76A is derived from DO-178B. Table 1.4 provides an overview of all mentioned software safety standards.

### Table 1-4. Air transport software safety certification documents and classifications.

<table>
<thead>
<tr>
<th>Document</th>
<th>Classification</th>
<th>Based on</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO-178B</td>
<td>A Catastrophic</td>
<td>JAR25</td>
</tr>
<tr>
<td></td>
<td>B Hazardous</td>
<td>DO-178B</td>
</tr>
<tr>
<td></td>
<td>C Major</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D Minor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E No effect</td>
<td></td>
</tr>
<tr>
<td>DO-278</td>
<td>AL1 AL2 AL3</td>
<td>DO-178B</td>
</tr>
<tr>
<td></td>
<td>AL4 AL5 AL6</td>
<td></td>
</tr>
<tr>
<td>Galileo</td>
<td>B C D E</td>
<td>DO-178B &amp; ECSS</td>
</tr>
<tr>
<td>AC-120-76A</td>
<td>Type C = apply DO-178B</td>
<td>B A DO-178B</td>
</tr>
<tr>
<td>EUROCONTROL</td>
<td>1a Accident</td>
<td></td>
</tr>
<tr>
<td>ANS</td>
<td>1b Hazardous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(DO-178B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Serious Incident</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Major Incident</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 Significant Incident</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 No immediate effect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESARR 4</td>
</tr>
</tbody>
</table>

### 1.2.3 Formal methods

One approach to safe software development is to deploy formal methods. Some work in which the author participated (Fokkink et al., 2002), provided interesting results. The term formal methods refers to the use of techniques from formal logic and discrete mathematics in the specification, design, and construction of computer systems and software (NASA 1998). A goal of application of the formal methods to system development is to replace the subjectivity of the reviews with a repeatable process of mechanically verifiable calculation (NASA 1997). Despite all progress begin made in formal methods (Berry et al, 2003) have suggested to use the term automatic bug detection in place of formal verification to illustrate that for nontrivial systems a conclusive proof of correctness remains hard. Rushby classified formal methods usage from level 0 (no use of formal methods), via level 1 (use of concepts and notations from discrete mathematics) and level 2 (use of formalised specification language with some mechanised support) to level 3 (use of full formal specification languages with comprehensive support environments including theorem proving or proof checking) (Rushby 1993). The level of formal methods
used should reflect the confidence required in the specification and the level of abstraction needed by the formal model. Recently (Vaandrager 2006) updated this classification. His first class is formed by invisible formal methods, followed by model checking, automated abstraction and theorem proving respectively.

The basic idea of formal methods is to use symbolic calculation. By using symbolic computation, formal tools can search huge state spaces efficiently. As a result these formal tools can find rare (classes of) errors as well as verify their absence for the specified properties of the model (Vaandrager 2006). All formal methods need a model, specifying some properties of the system in such a way that mathematical reasoning can be applied.

For theorem provers both the system and the specification have to be modelled as mathematical entities. For verification of the system with respect to the specified properties automated proof assistants have been developed. The Prototype Verification System (PVS) is an often used tool. The advantage of theorem proving is that once the proof is obtained, it will always hold for the class of systems modelled. The disadvantage is the considerable effort of specialised experts needed and the amount of abstraction needed to produce a tractable model.

In model checking the system specifications are expressed as (temporal) logic formulas. Efficient symbolic algorithms are used to traverse the defined model. Examples of tools are uCRL and Uppaal. An advantage of model checking over theorem proving is that analysis of the model and its properties are automatic. Another advantage is the ability to produce a counter-example in case the property does not hold. A disadvantage compared to theorem provers is the reduced expressivity of the modelling language. Despite the automatic analysis, model checking still requires substantial effort of trained experts.

As systems with a complexity found in real-life remain difficult to handle for current model checkers, the automated abstraction approach has been used. On purpose the system modelling is started at high abstraction, i.e. an integer is zero, positive or negative. If the property can be proven verification is completed. In case of a counterexample, either the counterexample is valid and a bug in the real system has been found, or the counterexample can not occur in the system. In the latter case a refinement of the abstraction is needed. Tools like Slam and Blast have been used successfully for device drivers of substantial size (Gates 2003).
The class of invisible formal methods includes for example the concept of variable types. Efficient algorithms check their correctness at very low cost to the user. This principle that the user provides some additional specifications in commonly used tools, like UML, which then automatically perform some checks, is used in the Omega project. This European project aims at correct development of real-time embedded systems (Omega, 2005) by adding some property information in the model using Object Constraint Language (OCL) or life sequence charts. Various (semi)automated tools will check timing properties at small cost to the users.

To conclude this part on formal methods, the relevant text of the airborne software certification document is copied below. Formal methods could produce an implementation whose operational behaviour is known with confidence to be within a defined domain. In their most thorough application, formal methods could be equivalent to exhaustive analysis of a system with respect to its requirements. The goal of applying formal methods is to prevent and eliminate requirements, design and code errors throughout the software development processes. Thus, formal methods are complementary to testing. Testing shows that functional requirements are satisfied and detects errors, and formal methods could be used to increase confidence that anomalous behaviour will not occur (for inputs that are out of range) or unlikely to occur. (DO-178B 1992). As the work in this thesis includes obtaining the specification, formal methods have not been used.

1.2.4 Human-centred design

All work in this thesis relates to (sub)systems with human operators. These professionals are retained as not all circumstances can be foreseen preventing full automation (ICAO 2005). This is typical for systems in safety critical applications. (Vicente 1999) characterised the resulting complex systems as those in which many people with different perspectives are working together in a dynamic environment with uncertain data, unanticipated disturbances and with computer mediated actions. Historically a technology-centred approach (provide what is technically possible without paying proper attention to the remaining human’s task) is used. It is well known that humans are the most important contributing factor to aircraft accidents (NASA 2004), (Kebabjian 2004). Human-centred design is an alternative design method for complex systems that addresses such problems by focusing mainly on the user. Norman (1998) defines it thus: “It’s a process of product development that starts with users and their needs rather than with technology. The goal is a
technology that serves the user, where the technology fits the task and the complexity is that of the task, not of the tool.’’ The foundation of a human-centred design is a structured analysis of the users’ tasks. Based on a thorough task analysis, the design process includes activities to ensure its focus on the human, like usability engineering, iterative design and prototyping. The often used ISO standard 13407 (ISO-13407 1999) describes human-centred design as a multidisciplinary activity, which incorporates human factors and ergonomics knowledge and techniques to enhance effectiveness and productivity, while improving human working conditions. Additional guidance is offered by ISO TR 18529 (ISO-18529 2000), which contains a more detailed list of activities and 44 base practices. For domains without safety concerns, like consumer application, such an approach is already well established (Koskela, Väänänen Vainio Mattila 2004).

In aeronautics human-centred design was first introduced for pilots at equipment level. Subsequently human-centred design spread to include the entire pilot work suite, i.e. the flight-deck, culminating in the proposed flight deck certification by FAR/CS 25.1302 (FAR 2004). For the flight-deck FAR/CS 25.1302 certification will enforce the following: provision of an intuitive visual layout, consistent behaviour, provision of feedback and minimisation of mode effects. In the USA the NASA human-centred systems lab performed a lot of work at equipment level, which in recent years evolved into application of human-centred design at equipment integration level, which level is also referred to as higher level or system level (NASA, 2007). In Europe EURISCO is performing a similar role (EURISCO, 2007). Recent contributions for pilots which confirm the importance of the cited requirements of FAR/CS 25.1302 include (Hooey, et al., 2002) and (Hooey, et al., 2000). The need for consistent behaviour is the main theme of (Boy, Bradshaw, 2006). Human-centred design work for air traffic controllers, on which this thesis focuses, is lagging behind that for pilots.

1.2.5 Real-time simulation

In air transport use of real-time simulators is common. For training purposes the military employ synthetic environments comprising real-time simulators for the subsystems involved. As these simulators can be quite substantial and consequently can not be relocated conveniently for a combined simulation, networked simulation technology has been developed to combine geographically dispersed real-time simulators into a single networked simulation. The resulting High Level
Architecture (HLA) standard series (IEEE-1516 2000), (IEEE-1516.1 2000), (IEEE-1516.2 2000) supports component-based simulation. Each real-time simulator is considered a component. HLA allows each participating simulator to define the set of attributes of (selected) objects it will make available to the other networked simulators and which objects will remain private and hence inaccessible for the other simulators. Similarly each simulator defines which attributes are needed from the other participating simulators. In the synthetic environment the simulated (sub) system behaviour can be observed whilst interacting with the other systems of the intended target environment.

For those readers which are not familiar with air traffic management, the next section contains a synopsis of the field, to facilitate their understanding of some of the performed research in the next chapters. The synopsis relates to advanced air traffic management in busy European airspace.

1.3 ATM synopsis

Pilots are responsible for safely flying aircraft. However when flying in Instrument Flight Rules conditions, like flying through clouds, pilots can not apply the see-and-avoid principles from the Visual Flight Rules so pilots need assistance from (ground-based) air traffic management to ensure safe separation from other aircraft. In such cases, the pilot and the air traffic controller are responsible for maintaining safe separation of their aircraft from other aircraft or of their aircraft from obstructions in case the aircraft is manoeuvring on the airport. To know the current position of all aircraft under his control, the air traffic controller needs surveillance data. Surveillance data can be independent from the aircraft, in which case it is usually provided by radar observation, or provided by the aircraft (dependant surveillance). For dependant surveillance the aircraft downlinks its position to the ground using a (dedicated aeronautical) data link. Dependant surveillance is used where radar coverage is not available, for instance over large water areas or sparsely populated areas. In areas with low traffic density surveillance may not be available at all. Such cases are not considered in this thesis. Prior to departure pilots have to file a flight plan indicating the flight’s preferred route and the estimated arrival time at certain points along the route. Ground-based air traffic management systems combine surveillance information (either independent or dependant) and flight plan information to obtain a permanently updated air traffic status picture. Based on this air traffic status an advanced ground-based air traffic management system performs
two functions to increase safety, mid term conflict detection and short term conflict alert. These functions act as separate safety nets, as depicted in Figure 1-3. Mid term conflict detection works up to 20 minutes in advance, while the short term conflict alert functions up to 2 minutes ahead. Such air traffic controller tools are discussed in the chapters on human-centred design and networked real-time simulation. On top of this a third independent safety layer is active. The aircraft based Traffic Collision Avoidance System, which can provide traffic advisories 40 seconds before reaching minimum separation and (mandatory) resolution advisories 25 seconds before reaching minimum separation. Such Traffic Collision Avoidance System advisories should not be activated during normal operations.

The International Civil Aviation Organisation (ICAO) is a United Nations based organisation with nearly all nations as members. The task of ICAO is to ensure uniformity in systems and procedures, which is of critical importance in international civil aviation (Schaik 2005). The main document is the “Chicago convention” originally signed in 1944 with currently 18 annexes, all of which are periodically updated. For the work presented in this thesis the most relevant are Annex 8, airworthiness of aircraft and Annex 11, air traffic services. The aircraft level FAR/JAR25 falls under Annex 8. In addition to its world-wide standards, ICAO recognises regional arrangements, as civil air traffic characteristics differ in different regions. Regions with heavy traffic and consequently advanced air traffic management and commensurate local arrangements include North America and Europe. The latter two regions also co-operated in the airborne software safety document DO-178B/ED-12B, an elaboration of FAR/JAR-25.

ICAO has developed a global air traffic management operational concept with a planning horizon up to and beyond 2025 (ICAO 2005). The ICAO vision is “to achieve an interoperable global air traffic management system, for all users during all phases of flight, which meets agreed levels of safety, provides for optimum economic operations, is environmentally sustainable and meets national security requirements”. Air traffic management is “the dynamic, integrated management of air traffic and airspace (safely, economically and efficiently) through the provision of facilities and seamless services in collaboration with all parties”. The first two of their six guiding principles address safety and humans. As expected “the attainment of a safe system is the highest priority in air traffic management, and a comprehensive process for safety management is implemented that enables the ATM community to achieve efficient and effective outcomes”. The ICAO concept contains a conflict management function which comprises three independent layers,
in accordance with the current practise described in the first paragraph of this section and in line with the principles depicted in Figure 1-3.

The guiding principle addressing humans is worded as “humans will play an essential and, where necessary, central role in the global ATM system. Humans are responsible for managing the system, monitoring its performance and intervening, when necessary, to ensure the desired system outcome. Due consideration to human factors must be given in all aspects of the system”. The guiding principle on humans implies that, for the foreseeable future, two humans will remain involved in air transport, pilots flying aircraft and ground-based controllers performing air traffic management. Due to this observation, this thesis focuses on safe software systems with human involvement. As a consequence of this principle, Eurocontrol even requires the primary mode of operation of Unmanned Aerial Vehicles (UAV) to entail oversight by a pilot-in-command when operating in civil airspace (Eurocontrol 2006). Only in the event of loss of the control data link, the autonomous flight mode of the unmanned aerial vehicle is reverted to, so studying safe software systems with human involvement is also relevant for this type of vehicles. More information on air traffic management can be found in for example (van Schaik 2005) and (Nolan 2003).

To summarise, the work extends the state-of-the-art of safety-critical software development by addressing specifications of system of systems, by applying human-centred design to a new domain, by addressing the software realisation phases, and by performing an experiment to contain costs by deploying modern software techniques like Java and COTS for software with medium safety criticality levels. The next section focuses the presented research by developing a number of research questions, which this work will elaborate in the subsequent chapters.

1.4 Research questions

The research theme of achieving safe software, has been broken down into the following research questions. Each research question is elaborated in several auxiliary questions.

*How to increase the confidence in software specifications prior to embarking on an elaborate DO-178B compliant safety critical software realisation process:*
• Can a real time simulation of the software specifications be combined in a cost effective way with existing simulations of systems connected to the specified system;

• Given the essential role allocated to the human in the aeronautical domain, what is human-centred design;

• Can human-centred design be applied to airborne safety critical software development.

*Can an organisation without prior exposure to the specific details of the software considerations in the airborne systems and equipment certification document DO-178B, create a compliant software development process in a real world setting:*

• What is addressed by the airborne software certification document DO-178B;

• How can a DO-178B compliant software development process be realised;

• How much effort does verification require for software classified at the highest criticality level relative to other software development processes;

• Can the relative amount of verification effort be reduced by automation;

• Can requirements evolution be accommodated for certifiable software classified at the highest airborne criticality level of DO-178B;

• What is the relevance of airborne software certification processes with respect to the processes required by other software safety certification domains.

*Can COTS be used to obtain certifiable safe software more responsively and more affordably for software classified at the lower airborne criticality levels:*

• Can COTS software of unknown pedigree be certified for lower airborne criticality levels;

• Is a COTS–based approach more responsive then a custom-made approach;

• Is a COTS–based approach more affordable then a custom-made approach.
1.5 Research approach

This work has been performed with the objective of being relevant to practitioners. This means the work having relevance for developers aiming to achieve safe software for application in areas were the resulting products need to be demonstrably fit-for-purpose. Usually such confidence in safe software behaviour is achieved through certification at a safety level appropriate for the intended use. All presented work is based on the aeronautical software safety certification, (DO-178B 1992) and its derivates.

In their often cited paper (Benbasset, Zmud 1999) describe three characteristics for research to become relevant for practitioners. Such research has to be interesting, applicable and current. Interest can be achieved by selecting topics with interest to stakeholders and by looking to practise to select topics. The previous paragraphs illustrate that the topics of this thesis are selected accordingly. Applicable is elaborated by (Benbasset, Zmud 1999) as focussing on the outcome of the research e.g. a usable result which can be implemented in practical situations. All research has been selected accordingly. Current relates to research which focuses on issues and technologies which are in actual use by the stakeholders for the intended application domain. All research in the presented work complies with this recommendation as well. Taken together the research achieves relevance to practitioners.

Case studies are mentioned by (Benbasset, Zmud 1999) as a way to produce results with relevance to practise. Consequently five out of the six contributions in this thesis are case studies contained in the chapters 2, 3, 4, 5 and 7. Another option for achieving relevance, mentioned by (Benbasset, Zmud 1999) is work that synthesizes an existing body of research, which is done in the remaining contribution, chapter 6.

Contrasting with the positivist approach of the preceding paragraphs, the interpretative approach attempts to extract meaning from observations. This approach is often used in software engineering. In practise many variables influence the results of software engineering, which makes it difficult to perform controlled experiments by changing only one variable at a time. Using a concept from biological and medical science, (Torii 1999) refers to this approach as “in vitro” or laboratory setting. Grounding theory (Orlikowski, Wanda 1993) is used in fields including social sciences and business sciences to derive meaning from actual data for cases sharing many similarities but also differing in many other aspects. (Torii, 1999) refers to this as the “in vivo” or naturalistic setting. All presented research
uses the “in vivo” approach. The result is a set of methods or theory applicable for
the defined application area from which it is derived. Chapter 3 uses grounded
theory partially to analyse the results of the first human-centred design case study
and obtain guidance for the second case study. It is also used to substantiate some of
the findings related to using the human-centred approach. Chapter 5 uses some
elements of grounded theory to analyse the observations. In chapter 7 grounded
theory is used extensively in order to extract meaning from the experimental results.

This approach of making cycles through observations or performing experiments,
extracting meaning into a model, verifying (or refuting) the model and learning or
reflection is described by (Basili 1996). This approach is akin to what is done in
other sciences like physics with the first phases performed by experimentalists and
the later phases by theorists. During the reflection phase several models could be
incorporated into a more general theory. However for cases where the original
model holds, the predictions of the new theory should (within the model’s accuracy)
be compatible with the predictions of the original models. In physics already a
century ago the development of quantum theory was guided by the correspondence
principle, first formulated by N. Bohr (Bohr 1923). The correspondence principle
addresses the case when a substantial amount of observations validate a theory and
new observations can not be explained by the existing theory. It states that the new
theory, which explains the new observations, must approach the existing theory for
those cases correctly explained by the existing theory. In the case of quantum theory
and classical theory, the quantum theory must approach the classical theory
asymptotically in the limit of large quantum numbers (Messiah 1974). All presented
research tests the predictions of existing theory with the aim of extending the range
of observations for which the theory holds. In the COTS-study the predicted results
were not observed and grounded theory is used to extend the model.

The correspondence principle is also used in other sciences like human factors,
albeit with different wording. For research in the human factors discipline Sträter
(Sträter 2005) states three scientific rules. One rule states any new model should
include the explanatory power of the preceding ones. A second rule requires any
new model to have more explanatory power then the preceding ones. Taken together
these rules capture the idea of the correspondence principle. Sträter’s last rule
articulates any model to obey the rules of scientific logic, i.e. new observations can
never fully verify any model but any model can always be falsified by new
observations. However the trust in a model increases when it is observed to hold for
more and more diverse observations. The latter rule is implicitly used in fields like physics.

Based on an extensive body of software engineering research, (Basili et al., 2002) provide two lessons learned to increase research efficiency. The first recommends a symbiotic relationship between research and practise, with both activities gaining from the interaction. The second lesson learned advises a close proximity of researcher and developer which is aiding both. Both recommendations have been implemented in the presented research, especially the work including software metrics. These recommendations have been found beneficial.

Summarising the research results are relevant due to the way the topics are selected. By using the “in vivo” approach and deriving theory from practical cases through grounded theory, the used approach results in a valid suite of methods for various aspects of achieving certifiable safe software.

In several of the case studies measurements have been performed. To guide the measurement process the Goal/Question/Metric method (Bassili, Weis 1984) has been applied. This method has been used extensively in the software process improvement field. The Goal/Question/Metric method comprises several phases related to the measurement. The definition phase documents the measurement, i.e. it defines the goal, question(s) and metric(s). During the data collection phase the data are gathered. During the interpretation phase the measurement data are processed to answer the question and evaluate the goal. In all presented results the measurements (or metrics) are objective, i.e. the same value will be determined independent of the observer performing the measurement. For many measurements, like e.g. software size and document volume, dedicated software has been written. This software has been verified before use. The strict configuration control required for certifiable software provides a strict definition of what is included in each release. For other metrics, like for effort, the financially audited administration of the participating organisation has been used, with the project manager responsible for the correct accounting of the effort acting as verification. For yet other metrics, like the verification method used, the certification requirements prescribe their registration so they could be obtained from the software process artefacts. Complying with the Goal/Question/Metric method, members of the team are consulted to verify the correct interpretation of the measurements while addressing the question and the goal. Taken together this process aims to ensure that other observers of the same case study would have obtained the same result. Describing the measurements and
their conclusions separately provides for traceability, and allows the same experiment set-up to be repeated on other similar case studies.

1.6 Overview of the thesis

This thesis consists of three journal papers plus three conference papers. A verbatim copy of all papers is included. To provide a harmonised lay-out in this work, in two cases (section 3.2 and 6.2.2) some duplicated information has been replaced by a reference to the information in the introduction.

The first research question is addressed by the first part of the work, containing chapters two and three. Based on the work performed between 1999 and 2001, chapter two discusses how connecting available real-time simulators can help to increase the confidence in the behaviour of the specified system, as it will operate in its intended environment. The HLA standard being used was only officially released in 2000, so the work on which this paper from 2002 is based was innovative at that time. The included paper is:


Chapter three addresses an innovative way to design systems for well-trained human operators. The research on the first case study was performed from 1994 until 2000. The subsequent case study was completed in 2003. Applying human-centred design for air traffic controllers in combined air – ground collaboration tasks, as was done in the first case study, was innovative. The results are reflected in the paper:


The second part of the presented work is dedicated to the second research question and consists of three papers. Chapter four documents how a novice organisation can proceed to develop software which is certifiable as safe for use in aircraft, as described in the paper:

Chapter five analyses the processes at the end of the 1996 – 1999 realisation period, after successfully certifying the first release of the software. The analysis includes the subsequent certification of a significantly extended second release of the software. The combination of software processes used was innovative when the work started. Four years after the release of DO-178B not much was published on compliant processes which successfully made it through certification at the highest safety criticality level. Even now, eight years after the paper’s publication, available data is still scarce. In (Reifer 2004) data is published, including data for airborne military software. However no information is provided on the (mix of) safety criticality levels. The data provided on effort distribution is not detailed for military airborne systems. As a last observation to those data, in many European countries military aircraft are exempt from certification as is mandated for civil aircraft. Summarising this makes the provided work innovative at the time of publication. These results are contained in:


Chapter six, contemplates the relevance of the safety requirements and the corresponding evidence needed for certification in the airborne domain with respect to the standards applicable for other domains with safety concerns based on work performed from 1998 until 2003. Previous comparisons of software safety documents existing at the time of publication, like (Papadopoulis, McDermid, 1999), considered only transport standards. This work is more comprehensive as more standards also from outside the transport domain are considered. The standards were selected based with an actual application in mind. This also led to special attention being paid to COTS issues, for COTS software of unknown pedigree. At the time of research security certification was becoming more important, so some relevant
security standards were considered as well. Consequently the resulting work was original at the time of publication. The result is expressed in the following paper:


In the general and military domain the COTS software paradigm provides much benefit with respect to the custom-made approach based on dedicated proprietary or military standards and specifications. This paradigm might also be relevant for the specialised software of the certifiable safe software niche, which currently mainly builds on proprietary and custom-made software, the subject of the third research question. The supporting research has been performed between 2000 and 2004, at which time it was relevant and innovative for the stakeholders involved. It also provides public data to substantiate some of the opinions in the interview of (McDermid 1998). The arguments to address this research question are considered in the chapter 7, containing the paper:

1.7 Author’s related work

The author has performed related work, the refereed part of which is included below. Some non-refereed publications have been listed separately. To improve the readability of the overview, below the related work is organised in themes, with, for each theme, the most recent work listed first.

Realisation processes for certifiable safe software:


- E. Kesseler, Safety, an organised approach, proceedings 4th annual CNS/ATM seminar, Part 1, session 3, March 9 - 10, 1999, Taipei, published by Institute of Information industry, Taiwan;


- G.J. Dekker and E. Kesseler, Development procedures of the on-board Attitude control software for the SAX satellite, Automazione e strumentazione, no 6, page 93 – 99, Anno XLV, Giugno 1997, published by Associazione Nationale Italiana per l’Automazione;

Applying formal methods to real-world cases:


Software safety certification:


Human machine interface:


Information technology for air traffic management:

E. Kesseler, Improving air transport collaboration, the TALIS experience, proceedings 9th annual CNS/ATM seminar, session 4, 1 – 2 March 2004, Taipei, CD-ROM published by Institute of Information industry, Taiwan;

E. Kesseler, Air transport, From Privilege to Commodity, the World Congress Aviation in the XXIst century, page 2.21 – 2.27, 14-16 September 2003, Kyiv. Conference by AIAA and ICAO and published by National Academy of Sciences of Ukraine;


E. Kesseler, Cheaper / faster / better and safer? proceedings 5th annual CNS/ATM seminar, session 2, March 1 - 2, 2000, Taipei, CD-ROM published by Institute of Information industry, Taiwan;


Information technology for co-operation and multidisciplinary work

E. Kesseler, W. Lammen, J. Weser, P. Guellec, A case study of aeronautic product life cycle management in the (conceptual) design phase, Product Data


- W.J. Vankan, E. Kesseler, M. Laban, Multi-objective optimisation of aircraft range and fuel consumption, First CEAS European Air and Space Conference, 10 – 13 September 2007, Berlin, CD-ROM published by Council of the European Aerospace Societies (CEAS) comprising members from 8 nations;


- E. Kesseler, M.H. van Houten, Multidisciplinary optimisation of a turbine disc in a virtual engine environment, 2nd European Conference for Aerospace Sciences EUCASS 2007, 1 – 6 July 2007, Brussel, CD-Rom published by Universite Libre de Brussels, Symposium 1, Session 5, 1_05_06;

- E. Kesseler, W.J. Vankan, Multidisciplinary design analysis and multi-objective optimisation applied to aircraft wing, WSEAS transactions on systems and control, issue 2, volume 1, page 221-227, December 2006, published by WSEAS press;


- E. Kesseler, Optimisation and multidisciplinary design: advancing the state-of-the-art in the European project VIVACE, ECCOMAS CFD, European Community on Computational Methods in Applied Sciences Computational Fluid Dynamics, 5-8 September 2006, Egmond aan Zee, CD-ROM published by Delft University of Technology;


- E. Kesseler W. J. Vankan, Taking Collaborative Engineering to the Sky, European Conference for Aerospace Sciences EUCASS 2005, 4-7 July 2005, Moscow, CD-ROM published by Russian Academy of Sciences, Symposium 1, section 1.3, 1.03.06;


- E. Kesseler, A. A. ten Dam, R. van Sterkenburg, Divide and control, Making distributed real-time simulations work, 6th International workshop SESP 2000, 10 – 12 October 2000, Noordwijk, published by ESA;

In addition to the refereed publications, the work of the author has resulted in the following non refereed publications:


- E. Kesseler, Improving air transport collaboration, seminar at Korea Transport Institute (KOTI), 6 February 2004, Seoul;


- Aeronautical Telecommunication Network course, authored the following lectures:
  - part European ATM perspective;
  - part ATN life cycle;
  - part ATN certification.

  ATN Technology Transfer, October 1998, Institute for Information Industry, Taipei;

- E. Kesseler, Safety critical software development, an aerospace example, invited presentation on EU European Systems and Software Initiative (ESSI) EUNET workshop at KEMA, September 10, 1999, Arnhem;
