Safety Lifecycle Activities for Autonomous Systems Development

Robert Alexander, Tim Kelly
Department of Computer Science, University of York
Heslington, York, YO10 5DD

Ben Gorry
BAE Systems Military Air Solutions,
Warton Aerodrome, Preston, Lancashire, PR4 1AX

Abstract

Current safety lifecycles do not deal well with the challenges of new and rapidly advancing technologies. As autonomous systems are such a technology, an alternate lifecycle model is required. This paper outlines a lifecycle model based on an iterative, exploratory approach, and identifies a range of evidence sources that can be used at the various stages of the lifecycle.

Keywords : safety, certification, lifecycle, evidence, safety cases

Introduction

In our previous work for the SER011 project ([1-3]), we addressed the AS safety lifecycle as far as top-level objectives and hazard analysis. In this paper we provide guidance for later stages of the lifecycle further through the lifecycle, as far as confirmatory safety analysis and the accompanying gathering of evidence for certification.

There are many existing models of safety lifecycles. However, AS are difficult for a number of reasons, not least that they use novel and rapidly advancing technology. The development of AS is therefore attended by a high degree of uncertainty about what is possible and what can be made safe. In this paper, we present a lifecycle model that makes allowance for this uncertainty, and allows AS developers to manage their commitment in the face of rapid change in methods, technologies, requirements and doctrine.

The next section defines what we mean by a "safety lifecycle" - experienced safety engineers may want to skip this. The section after explains what it is about AS that makes the safety lifecycle difficult. This is followed by a description of our proposed lifecycle. There is then a section on the different ways we can get safety evidence across the different stages of the lifecycle. Finally, we present some conclusions.

Safety Lifecycles in General

Put simply, a safety lifecycle is a description of the safety activities performed on a system over its lifetime. Ideally, this consideration should be cradle-to-grave; from first concept to final decommissioning. In this paper, however, we will restrict ourselves to activities up to release to service; we will exclude those activities that take place after the system is in the field.

A safety lifecycle happens in parallel with the main *development lifecycle* for the system, which is a description of all the other activities carried out to develop the system. To carry out safety lifecycle activities one must relate them to the development lifecycle. In order to keep this paper relevant to range of development lifecycles, we will not overly labour this relationship, but we will observe where the safety process has implications for development.

As engineers progress through the lifecycle, they are working on several fronts. For one, they are moving from a basic understanding of the system to a more sophisticated one. Secondly, they are moving from generating an in-house assessment of whether the system can be made safe to actually creating certification evidence. Finally, they are generally increasing the level of detail they are using their our model of the system; the object of analysis moves concept to architecture to design to actual implementation details. These activities do not have to proceed in lock-step, but for this paper we will assume that they move forward together.

It is worth noting the difference between a safety lifecycle and a safety process – a lifecycle is a high level description of the types of activities involved, whereas a process gives specific prescriptions as to what engineers should do. In this paper we have taken middle road: we describe a high-level lifecycle to which many safety processes could be fitted, and give some specific suggestions for techniques that could be used in that process. Def Stan 00-56 allows developers a wide range of different safety processes, and we have tried to support that flexibility while still giving some guidance.

There are many possible lifecycles, and several lifecycle models have been described, such as the safety variant of the "V" model given by Pumfrey in [4]. This is reproduced in Figure 1.

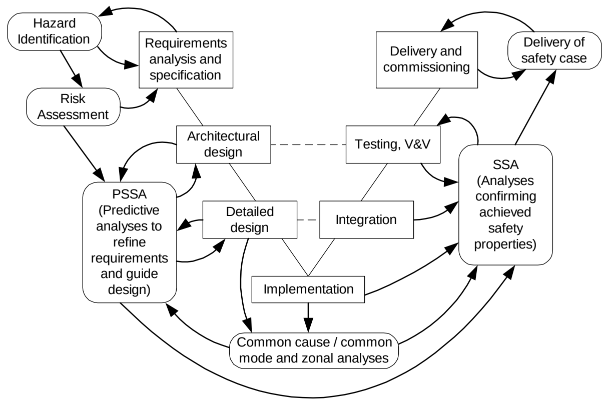
Hazard Identification
Requirements analysis and specification
Risk Assessment
Architectural design
PSSA (Predictive analyses to refine requirements and guide design)
Detailed design
Implementation
Common cause / common mode and zonal analyses
Integration
Testing, V&V
SSA (Analyses confirming achieved safety properties)
Delivery and commissioning
Delivery of safety case

Figure 1 - Pumfrey's 'V' Model (reproduced from [4])

At a minimum, a safety lifecycle must in some way contain the core safety activities that are shown in Table 1. Note that this table is not in any sense a lifecycle description - it makes no attempt to show how the activities interrelate, are positioned in time, or relate to the development lifecycle.

Table 1 – Minimum Lifecycle Activities

Activity	**Role**	**Inputs**
Hazard Analysis (FHA)	Derive hazards of conceptual system	Function/capability model of AS
Preliminary System Safety Analysis (PSSA)	Early on, evaluate potential for system to be made safe	An architectural design (an allocation of functions to components)
System Safety Analysis (SSA)	Get confirmatory evidence of safety	A detailed design (*how* the architecture will be implemented)
Particular Risk Analysis (PRA) and Common Cause Analysis (CCA)	Handle risks coming out of implementation details (*ongoing in parallel with PSSA and SSA*)	(*as each parallel stage*)
Safety Case	Produce a	Detailed design,

Development	safety case	analysis plan, and confirmatory safety evidence

Safety Challenges Posed by AS

There are several safety challenges for AS that are particularly relevant to the safety lifecycle.

First, engineers rely on getting precise, complete specifications for the systems they are working with. AS are problematic in that we are often forced to start with a *partial specification*. Realistically, we will only reach sufficiently developed specifications through prototyping and trials.

Second, safety relies heavily on predicting system behaviour, but AS make it hard to do this. Partly, this problem stems from novel technologies (see below) but it is also caused by reliance on emergent behaviour and the system architectures that that leads to. This reliance on emergent behaviour means that we will stop decomposing functions at a much higher level than we have traditionally. The resulting architecture will allow many components and a large amount of software to impact any given requirement at that level.

Third, AS are inherently very sensitive to their operating environment. In inhabited vehicles and human-operated system, much of the interpretation of the environment has been done by the human – they observe the objects in the environment (either directly or via artificial sensors) and make decisions about how to respond. AS will have to do much of this on their own.

Many current safety processes make little reference to environment modelling. Partly, this stems from the expectation of a well understood, well regulated and relatively predictable environment (e.g. the public roads for MISRA [5] and international controlled airspace for ARP 4754 [6]).

Fourth, introducing a new AS will inevitably have an impact on the humans who have to interact with it. This is particularly true when the AS replaces a traditionally human role, as in many UAV deployments. Many current safety processes pay little attention to modelling the human side of the system and its environment. It is either ignored entirely (e.g. in [7]) or only mentioned in passing (e.g. in ARP 4754, where it is relegated to a single paragraph (7.5.2.a) under "Validation of Requirements").

Fifth, AS certification requires high levels of assurance. In many AS deployments, either the problem, or the solution, or both, is unfamiliar. According to the McDermid Square from Def Stan 00-56 [8], it is this case (unfamiliar solutions to unfamiliar problems) that requires the highest assurance.

In those cases where the AS replaces an existing human-operated system (i.e. the problem is familiar) developers and operators may still need to give very high assurance of safety. This is because AS control systems may be held to the high standards required of machines, rather than the (often lower, often "grandfathered in") standards required of humans.

Finally, AS often depend on novel technologies. We discussed this in our earlier papers [1-3], and noted that any given new technology may not be amenable to the analyses we need for safety. It may be that we can make the technology work in a functional sense, but cannot be confident that it will remain safe. Novel, in this case, may include technologies that have been widely used in other domains but have never seen safety-critical use before.

One aspect that we have not mentioned before is the use of existing technologies in new roles. For example, we may want to take a sensor unit that has been used on inhabited aircraft and attach it to a UAV. If the role is the same (e.g. it only provides a surveillance video feed to a remote observer) then this may be easy to argue. However, on the UAV it may have a different role (e.g. the remote operator will use it to help their navigation of the UAV) or a different user (e.g. the video will be used directly by the UAV for autonomous navigation). In these different use cases, the technology may have new requirements that are hard to meet.

Proposed Lifecycle

Project risk in safety stems primarily from certification risk. This, in term, stems from several sub-types of risk:

- The risk that we won't be able to create a safe system
- The risk that we won't be able to provide a compelling safety case even though our system is safe
- The risk that our specific regulator will reject our safety case even though it is compelling to a general safety engineering audience

When we start out on a development project, we can't assess these risks accurately. We therefore need to take steps to refine our knowledge before we commit to a project. This management of commitment is at the heart of Boehm's Incremental Commitment model [9], a systems engineering process model that emphasises managing developer commitment before the risks of the project are understood. Adapting this idea from the ICM, we can propose a core safety process that lets us manage our commitment.

Figure 2 illustrates our process graphically in terms of the standard activities from Table 1. Starting with an idea, we conduct an initial hazard analysis and then proceed to PSSA. We iterate PSSA, changing our system concept, our system design and our analysis techniques each time, until we reach a point where we are satisfied that we can manage all risks and create appropriate certification evidence. At that point, we fully commit to the project and attempt to build and certify our product.

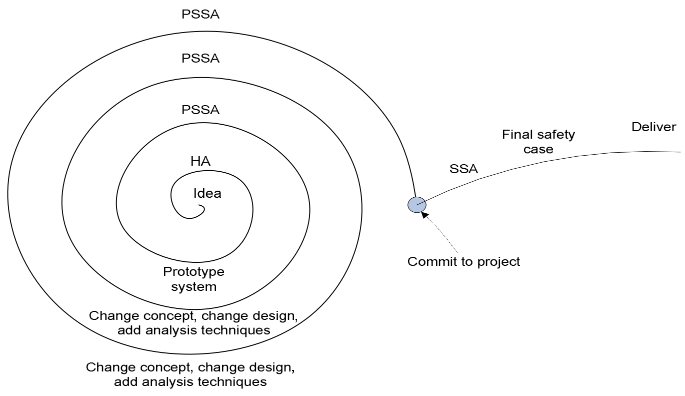
PSSA
PSSA
PSSA
HA
Idea
Prototype system
Change concept, change design, add analysis techniques
Change concept, change design, add analysis techniques
Final safety case
Deliver
SSA
Commit to project

Figure 2 – Summary of Our Approach

At the start of each iteration, the developer needs to determine what information is most important to acquire in this iteration. Information may concern design, safety risks, potential for analysis, or any other factor of concern. In the initial iteration this will be a pure judgement call; in later iterations it should follow from analysis of the nascent safety case. In every case, the information needs should be prioritised in terms of what will move you closer to a commit/no-commit decision.

Some of the changes to information needs will come from changes in requirements. As noted earlier, AS development will often start with a very partial specification – the developer will have quite hazy ideas of what the actual requirements are. It is likely to take prototyping to achieve any degree of clarity in these. Any change in system requirements will introduce new information requirements.

Once information needs are clear, development can proceed to a model or prototype. This is simplified, of course – in a real system, especially early on, there will be many models in parallel. These diverse models will be at different tiers and will focus on different components. For example, we may start with be a whole-system model represented as a MODAF OV-1, some informal sketches of the architecture of an AV and notes on the capabilities required of some sensor. Later, we may have a low-fidelity simulation model of the AV in context and a highly-detailed software-in-the-loop simulation for the sensor. In practice, models and prototypes need not be created anew each cycle, and will in most cases be revisions of models from earlier iterations.

Once we have at least one model, we can analyse it for safety evidence and use that to build a preliminary safety case. The safety case on the first iteration may be no more than a general strategy attached to a list of hazards, whereas later cases may be plausibly complete cases that just lack sufficient confirmatory evidence.

The preliminary safety case provides a means to assess the state of the project. The case will identify safety risks and assurance needs, and the developer should assess these and identify what needs to be done: they should identify unmanaged safety risks and extant assurance deficits. It is not necessary at this stage to have completed the management and to have all the assurance evidence, but it is important to identify what *can* be done. Similarly, the developer needs to identify any risk mitigation or assurance that they *cannot* see a way to perform.

When a developer assesses the risks they don't know how to mitigate and assurance deficits they don't know how to resolve, they get an indication of their certification risk and hence their overall project risk. At the end of the iteration, the developer can use this to decide whether and how to proceed.

At the end of each spiral, if we are happy with the residual safety risk we may proceed as described above. Alternatively, we can undertake risk-management strategies:

- We may "rebaseline" - change our project timescale and cost estimates
- We may change the project itself - the overall concept, the project scope, or the usage limits we require
- We may abandon the project (in AS projects, there are likely to be novel technologies and analysis methods that we can "cannibalise" for future projects)

A similar step is identified in the ICM [9] and in the project management guidance in the recent statement of best practice for software under Def Stan 00-56 [10].

Producing Evidence Throughout the Lifecycle

Def Stan 00-56 has a hierarchy of evidence types. Analytical evidence is favoured most highly, with testing evidence being less desirably and qualitative evidence (process, personnel etc) evidence being favoured least of all. This is a very simple presentation, however – there are many variables that need to be considered and all evidence types have potential value.

A more sophisticated structure for *software* evidence types can be found in [11]. In that, Hawkins identifies a range of ways in which evidence types and individual instances of evidence types can differ. When we are looking at evidence for the safety of an AS (which includes software), we can form a similar model.

Field Trials and Formal Methods

The first major type is field trials – operating of the AS under realistic conditions (although not actual operational ones). Although these are not the most favoured by 00-56, they are extremely powerful *in theory* (see Alexander and

Kelly in [12] for some expansion of this point). In particular, because they combine the whole AS product with its real world embodiment in a real-world environment they have excellent potential to reveal unwanted emergent behaviours. The quality of the evidence they provide is exceeded only by real operational use, although because of the limitations of the physical world it is sometimes difficult to acquire that information (e.g. what was that computer board doing just before it failed?)

A major problem with field trials, however, is their extreme expense. They require huge amounts of time and resources. A second problem is risk (because the embodied equipment is being used) and a third is the practicality of actually doing the trials at all (e.g. in the UK there is inadequate range space for many UAV trials). Field trials therefore have limited value in practice because we cannot do anywhere near as many as we would like. They are necessary, but they cannot be used for all our evidence. We must therefore explore alternatives that can give us information about whole-CAS behaviour in context ("mission-level behaviour") at less expense.

Similarly, formal methods for software provide excellent confidence in their results – they provide a strong answer to the question “does this software meet its specification?”. They are the quite different to field trials in that they can rarely detect that the specification is wrong in the first place. We must therefore supplement them with field trials or other methods that help us to get to a specification that we have confidence in.

There are a range of alternative evidence sources. Here, we have concentrated on simulation; for other sources, the reader is referred to our SEAS DTC report [13].

There are three ways that alternative sources can be used. One way is to have them provide evidence directly (for example, using formal methods to demonstrate that a particular software failure cannot occur). Another is to use them to direct and focus our limited capability to do field trials (for example, using a constructive simulation to find hazardous scenarios that should be explored in field trials). A third way is to create evidence using an alternative method and use field trials to test and challenge the evidence it produces, thereby creating composite evidence that is more powerful than either. This approach can provide evidence that is more credible the alternative could provide alone and far broader in scope than could be afforded using field trials entirely.

Simulation-Based Methods

By “simulation”, we refer to all computer models that derive predictions of a system’s behaviour by progressing the model through time. There is some overlap with the formal models discussed in the previous section, but not all simulation models have formal semantics or are amenable to model checking.

There are many different scopes and abstractions of simulation that can be useful here. Perhaps the most valuable scope is *mission-level simulation* – a computer model that describes the behaviour of the AS in an operating context that includes simulated peers, outside entities and (where applicable) hostile entities. This can be carried out using several distinct abstractions, for example:

- A very abstract simulation of the data and command links between the members of an AS group, which are subjected to stimuli from pre-programmed test sets (since this simulation doesn’t have ongoing human interaction, it is a *constructive simulation*)

- A medium-fidelity simulation of AS behaviour, including movement within a 3D space and explicit physical communications (given terrain and electromagnetic restrictions), which runs simulations of various plausible simulations using a server farm (using a Monte Carlo approach to generate random variations).
- An interactive simulator in which a large number of personnel can take the roles of the humans in the scenario and interact with realistic models of the AS (such an interactive simulation is often described as a *synthetic environment*).

Within the scope of our current work, there are two key roles for simulations – prototyping and evidence gathering.

The use of simulation for prototyping is well established. It is popular because it doesn't require strong commitment to simulation validity – as with many PSSA activities, a problematic lack of fidelity is a project risk, not a safety risk. Prototyping for AS may be particularly rewarding because of problems of partial specification and a general uncertainty about the best operating concept. It may only be at the prototyping stage that the interactions of the AS's diverse capabilities are fully understood.

Once we reach the architecture stage, we can exploit our increased knowledge to explore the implications of different architectures. Simulation has a key role here in bringing together our concepts for individual AS with our model of the context that they occupy.

At the very least, we can build a simulation context that represents our knowledge of communication and responsibility as captured by human role analysis (based on techniques such as that described by Thoms [14]) We can also exploit any knowledge we have about networks between AS, and between AS and humans.

Concentrating on the human aspect, it may be possible to estimate operator workload over time through modelling at this stage. It may reveal how intended usage patterns give rise to spikes in demands on the operator. This will allow some scoping for the number of AS per operator. Similarly, we can use our knowledge of data networks to find possible spikes in bandwidth needs.

The use of mission-level simulation for evidence gathering is more unusual. In this role, simulation is used to produce confirmatory evidence of safety, much like conventional testing or operating experience is used. It is difficult to use in this role because of concerns about the validity of any given simulation – the less valid the simulation, then the less compelling the evidence. It may, however, be the only option to resolve many assurance deficits about how an AS will actually behave when deployed – the equivalent volume of field-testing will be impossible.

Like the previous use of simulation for prototyping, this use will help to solve the environment sensitivity problem discussed earlier. The simulation can embody the accumulated knowledge about the environment, and the simulation model of the AS itself can show the AS's response to that environment.

The set of environmental factors to be modelled, and the set of runs to perform within the parameter ranges of the factors that are implemented, must be derived from something. One approach would be to use an accident database such as that described in [15] to derive the scenarios and the parameters of concern, by picking on the features that were implicated in the accident sequence and modelling those. For example, an accident sequence can be repeated in an interactive simulator using an AS in place of an inhabited vehicle.

Conclusions

We can conclude that the lifecycle needed for an autonomous system is not radically different than for a conventional system. However, many *activities* are different, and we need a different *emphasis*.

An iterative approach to AS safety based on extensive exploration through prototyping is of value and will make the challenges of AS development more tractable. Allowing evidence to be generated through non-traditional model-based activities will allow us to get the necessary assurance even though we have great problems in knowing whether the (partial) specification we are using is a good one.

For this to be practical however, extensive further work is needed on validating formal mission-level models, on deriving evidence from non-formal simulation models, and on continuously validating both kinds of model using field trials and operational experience.

References

[1] R. D. Alexander, M. Hall-May, T. P. Kelly, "Certification of Autonomous Systems," in *Proceedings Of Proceedings of the 2nd SEAS DTC Technical Conference*, Edinburgh, (2007).

[2] R. Alexander, N. Herbert, T. Kelly, "Structuring Safety Cases for Autonomous Systems," in *Proceedings Of the 3rd IET System Safety Conference*, (2008).

[3] R. D. Alexander, N. J. Herbert, T. P. Kelly, "Deriving Safety Requirements for Autonomous Systems," in *Proceedings Of Proceedings of the 4th SEAS DTC Technical Conference*, Edinburgh, (2009).

[4] D. J. Pumfrey, "The Principled Design of Computer System Safety Analyses," DPhil Thesis, University of York, (1999).

[5] "Guidelines for Safety Analysis of Vehicle Based Programmable Systems," The Motor Industry Software Reliability Association, (2007).

[6] SAE, "Aerospace Recommended Practice 4754 — Certification Considerations for Highly-Integrated or Complex Aircraft Systems," Society of Automotive Engineers, (1994).

[7] Y. Papadopoulos, J. A. McDermid, "The potential for a generic approach to certification of Safety-critical systems in the transportation sector," *Reliability Engineering and System Safety*, vol. 63, pp. 47-66, (1999).

[8] "MoD Interim Defence Standard 00-56 Issue 4 - Safety Management Requirements for Defence Systems," Ministry of Defence, (2007).

[9] B. Boehm, J. A. Lane, "Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering," University of Southern California usc-csse-2007-715, (2007).

[10] C. Menon, R. Hawkins, J. McDermid, "Interim Standard of Best Practice on Software in the Context of DS 00-56 Issue 4," Software Systems Engineering Initiative SSEI-BP-000001, (2009).

[11] R. Hawkins, "Software Safety Evidence Selection and Assurance," Software Systems Engineering Initiative SSEI-TR-0000041, (2009).

[12] R. Alexander, T. Kelly, "Simulation and Prediction in Safety Case Evidence," in *Proceedings Of the 26th International System Safety Conference (ISSC '08)*, (2008).

[13] R. Alexander, B. Gorry, T. Kelly, "Safety Lifecycle Activities for Autonomous Systems Development," University of York SEAS/TR/2009/2, (2009).

[14] J. Thoms, "Understanding the impact of Machine Technologies on Human Team Cognition," in *Proceedings Of the 4th SEAS DTC Technical Conference*, (2009).

[15] N. Storey, A.-M. Little, "An Internet-Based Searchable Database of Air

Accidents," in *Proceedings Of the 19th Systems Safety Conference*, (2001).

Acknowledgements

The work reported in this paper was funded by the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence.