Article:
Hodge, Victoria J. and Osborne, Matthew (2025) Agile Development for Safety Assurance of Machine Learning in Autonomous Systems (AgileAMLAS). Array. 100482. ISSN: 2590-0056

https://doi.org/10.1016/j.array.2025.100482

Array 27 (2025) 100482

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

Agile Development for Safety Assurance of Machine Learning in Autonomous Systems (AgileAMLAS)

Victoria J. Hodge *, Matt Osborne

Department of Computer Science, University of York, York, YO10 5GH, UK

ARTICLE INFO

Keywords:
Agile development
Autonomous systems
DevOps
Machine learning
Safety assurance
Software engineering
Through-life

ABSTRACT

Recent advances in ML have enabled the development of autonomous cyber–physical systems for a broad range of applications. Using ML, these autonomous systems are able to learn, adapt, and operate with no human intervention. However, this autonomous operation poses a problem when proving that they are acceptably safe. Designers and engineers have traditionally used 'Waterfall' or V-model development lifecycles to develop safe systems, but ML engineering requires iteration and adaptation. Iterative development necessitates enhanced lifecycles, augmented methodologies, and the need to systematically integrate rigorous safety assurance with ML development and operation activities. In this paper, we introduce a novel lifecycle, and comprehensive methodology for safely developing, operating, and assuring autonomous systems which use ML. The lifecycle combines Agile software engineering, ML engineering, and a safety engineering framework using iterative and incremental development. This paper provides systematic step-by-step guidelines for developing and deploying ML for autonomous systems using DevOps and MLOps, and for generating compelling safety cases. We have developed and refined our methodology on a recent set of projects undertaken to develop autonomous robots across a variety of domains.

1. Introduction

Recent developments in machine learning (ML) have taken autonomous systems to an advanced level. A barrier to their wider deployment however, is confidence in their safety [1]. Autonomous systems must model and maintain the model of their operating environment, and be able to continuously moderate and optimise their operation in the presence of uncertainties, resource variability, evolving user needs, attacks and faults. Additionally, software (including that used in machine-learnt functions) can contribute to hazards, and its development must be confidently assured for use in autonomous systems [2]. Hence, the development and safety assurance of these adaptive and safety-critical systems (SCSs) is complex. Many existing safety-critical development processes are also aligned with traditional 'Waterfall' or V-model development lifecycles, even though such lifecycles have been decried in the academic literature since 1988 [3]. Justifying the safety of ML necessitates rigorous assurance processes, but these are often expensive [4]. Recently, researchers have devised methodologies to systematise the assurance process for ML — such as AMLAS [5] (Assuring Machine Learning in Autonomous Systems) which focuses on assuring the ML model development. AMLAS comprises a process for: (1) systematically integrating safety assurance into the development of ML models and (2) generating the evidence base for justifying the acceptable safety of these components when integrated into autonomous system applications. AMLAS does not consider the ML component's software engineering, however.

Autonomous systems dynamically adapt their behaviour without any or with limited human involvement. Key to developing safe ML-based autonomous systems, and building a convincing safety case for the absence of unreasonable risk is integrating software engineering lifecycles with end-to-end and continuous safety assurance [6,7]. To this end, we assimilate AMLAS with an Agile development framework [8] and Agile systems thinking for through-life safety assurance of ML components of autonomous systems in safety critical domains. Building on knowledge gained during our recent projects to develop and assure autonomous robots, we create a methodology and guidelines for designing, developing and deploying assured ML components with compelling safety cases. Our novel safety-assured through-life development framework has been produced for software, ML, and safety engineers to work together to systematically integrate safety assurance into their ML development lifecycle using Agile practices.

The main contributions of this paper are:

* Correspondence to: Centre for Assuring Autonomy, Department of Computer Science, University of York, Heslington, York, YO10 5GH, UK.
E-mail addresses: victoria.hodge@york.ac.uk (V.J. Hodge), matthew.osborne@york.ac.uk (M. Osborne).

https://doi.org/10.1016/j.array.2025.100482
Received 8 April 2025; Received in revised form 27 June 2025; Accepted 28 July 2025
Available online 14 August 2025

- A novel and comprehensive framework for developing safe ML components for autonomous systems. The framework assures the design and planning, the software for ML, system software (the software stack), and the software for deployment, operation and maintenance of the ML model. This end-to-end development and assurance is currently missing from the literature [6].
- Our framework tightly couples safety engineering and software engineering to support the development of a defensible and comprehensive[1] safety case for the ML component where all safety claims are systematically argued over, and each is supported with a body of evidence.
- We decompose through-life system development and assurance into stages, and detail the objectives, and input and output artefacts for each stage.
- Our framework provides a palette of tools deliberately designed to be adapted pertinent to the specifics of individual domains and applications.
- This paper aims to guide designers, developers, and safety practitioners by explaining how to use Agile frameworks, practices, and processes while accommodating both safety and agility across applications and domains.

We found limited evidence of the adoption of Agile methodologies for safety-critical software development of ML in autonomous systems. Iterative, Incremental Development (IID), or 'Agile' lifecycles have their own detractors (see [9] for example) and these lifecycles are often misunderstood or misrepresented. Developers of safety-critical software for autonomous systems have not yet widely embraced Agile lifecycles [10,11] though some are beginning to adopt Agile practices (see Section 2.1).

In Section 2 we explain the Agile process, analyse Agile development for safety assurance in Section 2.1, and analyse Agile development for ML in Section 2.2. Section 3 describes the AMLAS methodology. We then introduce and motivate our AgileAMLAS methodology in Section 4, and explain the AgileAMLAS lifecycle in Section 5. The paper concludes in Section 6.

2. Agile development

Agile development is grounded in the Manifesto for Agile Software Development [8]. The Manifesto encompasses thinking patterns; design principles and values; and processes for individuals and teams from all stakeholder organisations. Agile has changed the way software is developed. It is time-boxed, iterative, and incremental with build-measure-learn development cycles which quickly, continuously, and cost-effectively improve the product's effectiveness, and remove uncertainty. Build-measure-learn development matches ML model development's train, test, and learn cycles, and also aligns with assuring the ML model's safety. Agile has other benefits for SCSs (summarised in Myklebust and Stålhane [12]) including faster delivery, accommodating change, providing traceability and reproducibility, enhancing quality, and reducing costs and risks. We discuss many of these benefits in this paper.

By carefully combining aspects of different Agile methods, we develop an Agile framework for software development of safety-assured ML. However, as we discuss next, Agile has generally been considered incompatible with safety-critical development to date (see for example [13]).

2.1. Agile for Safety-Critical Systems (SCSs)

Previously SCSs had a stable architecture design; upfront hazard analysis with a subsequent specification of safety requirements and the delivery of a safety case; formal modelling; a rigorously controlled requirements' change-request process [4,14,15] to ensure traceability [16]; and guidelines for change impact analysis [17]. At first glance, Agile appears to lack the software engineering rigour necessary for SCSs [18,19]. Two of the Agile values "*working software over comprehensive documentation*" and "*responding to change over following a plan*" have proved particularly challenging for SCSs developers [4,20,21]. The perceived lack of documentation and specification could even undermine the safety certification process [22,23]. Hence, Agile needs to be adapted [24] and the challenges carefully considered, [4].

Anecdotally, many people misinterpret Agile development as representing an absence of documentation and specification. This is a misconception, as the Agile Manifesto explicitly states that documentation should be minimised but used where appropriate. Agile aims to accommodate the whole spectrum of planning and documentation required for each product — just no more than is necessary. The BSI [25] safety standard states that documentation shall be accurate and concise, and be accessible and maintainable. It does not mandate a specific lifecycle model is followed. Agile development offers flexibility with sufficient formality for specification and requirements management as necessary [26], and formality and rigour can be increased as required [24]. For example, formal methods can be incorporated into the verification process of autonomous systems alongside other verification methods [27]. Formal methods can also be used for identifying and verifying safety requirements [28,29], run-time monitoring [30] or code generation from verified requirements [27]. In fact, with careful and minimal adaptation, Agile can be argued to be well-suited to safety critical software development.

Agile development elicits regular stakeholder feedback at each iteration; tightly integrating developers, customers and users. This enables safety assurance to be considered throughout the lifecycle. In fact, academic studies show that non-Agile SCS development often lacks the required input from stakeholders, operators or users [31]. As such, several papers in the literature support the use of Agile development with safety-critical systems — including AgileSafe [19], SafeScrum [32], Scrum for Safety (S4S) [33] and [34]. Heeager and Nielsen [22] provide a literature survey. None of these papers are tailored to the specifics of autonomous systems, however. AI, ML and autonomous technologies introduce their own challenges as we discuss next.

2.2. Agile for ML

The number of ML software engineering projects using Agile development is increasing [35–37]. However, ML has nuances and intricacies that make applying Agile methodologies more challenging [36]. ML is underpinned by data and ML models rather than just programming code [35,37], so this requires enhanced cross-functional teams and augmented lifecycle processes. We discuss our cross-functional AgileAMLAS team setup in Section 4.1. Moreover, technical debt (the cost of rework) can be a particular issue for software development for ML systems if speed of development is prioritised or ML engineers focus on improving ML model accuracy while increasing the algorithm's complexity [38]. The AgileAMLAS team should focus on simplicity, rigour and traceability as also advocated by Myklebust et al. [39].

Many ML models are high dimensional matrices and functions which are opaque. Additionally, ML components have more complex dependencies than traditional software engineering modules [35,37]. Thus, the scope of testing for ML development needs be widened beyond standard software testing to data coverage tests, model performance tests, model fairness tests, model latency tests, model robustness tests [40], defect detection, and on-device and platform testing [41] (as ML models can behave differently on different devices and hardware platforms). ML development requires enhanced traceability and reproducibility. Developing ML systems requires more iterations than 'traditional' systems, but iterating and incremental changes have been

[1] Comprehensive: including or dealing with all or nearly all elements or aspects of something (Oxford English Dictionary https://www.oed.com)

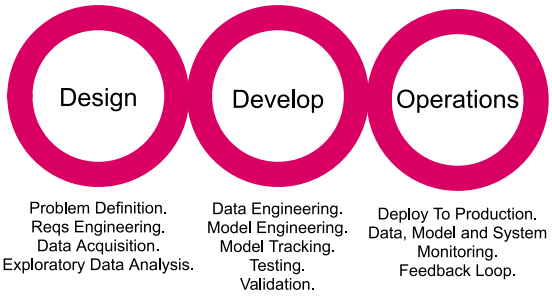

Fig. 1. The 3 loops of the MLOps lifecycle. This is often abridged to 2 loops as shown in Fig. 4 where step 1 Planning of the Develop loop subsumes the Design loop here.

argued by some to invalidate safety requirements (see [23,42,43] for example). However, Nielsen and Heeager [24] show how the flexibility of Agile practices is able to accommodate such incremental changes — not least as it can readily check the backward compatibility of newly developed code using regression testing. Agile for ML necessitates augmenting the development lifecycle with sophisticated testing and versioning (see Sections 5.7 and 5.3.1). Once the model is deployed to production it must be monitored continuously using the tests above and other tests such as out-of-distribution data tests [40] to identify faults, errors, and failures — and their causes. We discuss monitoring in Section 5.12.

We base our framework on **DevOps** (Development Operations) [44] which extends software development to cover operational responsibilities using a core set of practices adopted by cross-disciplinary teams. DevOps emphasises a *plan, code/develop, deploy/monitor* development lifecycle which is often automated using continuous integration, delivery and deployment. All assets within a project are version-controlled for traceability and configuration control. DevOps also aims to deliver and operate software reliably through continuous feedback and monitoring using appropriate metrics. Thus, DevOps is particularly well-aligned with safety assurance and ML model development; being responsive, iterative, experiment-focused, traceable, learning incrementally, and enabling cross-disciplinary teams. Myklebust et al. [39] demonstrate a high-level safety case and DevOps process for the automotive industry which allows the safe deployment of incremental updates. The compatibility with DevOps motivated our desire to integrate Agile (in particular MLOps with design, develop and operations stages shown in Fig. 1) with the AMLAS ML and safety methodology.

Before considering our extension of AMLAS (AgileAMLAS), we first describe the AMLAS methodology itself.

3. AMLAS

AMLAS [5] takes the system safety requirements as inputs, generates a **safety case** and develops an assured ML component in parallel, using a six-stage process. **The safety case identifies the system's safety risks and argues (logically states and convincingly demonstrates) how the risks are being mitigated using the body of evidence gathered.** A safety case is a living artefact that provides a compelling argument as to why a system is safe (in the context of the system, its intended use, and in a specified operating environment). Some safety cases can span 100–300 pages of notations and text and often have several thousand or more pages of accompanying evidence, support, context and assumptions [45] so incorporating a systematic assurance process such as AMLAS is vital. Each AMLAS stage shows the contribution to system safety of arguments and evidence, and highlight assumptions, trade-offs and uncertainties. AMLAS includes argument patterns which figuratively illustrate how the artefacts interrelate.

In AMLAS, 'artefact' specifically refers to the items (indexed by letters) that are consumed and/or produced by each stage [5]. In Agile, there is no consensus on artefact's definition so, for software and ML development, we define an artefact as "an item consumed/produced during the development process".[2] This aligns with the AMLAS definition and can be software (code), data, an ML model, a prototype, workflow diagram, a design document, a configuration script, or an output log, among others.

The six AMLAS stages, shown in Fig. 2, are interdependent, e.g., an activity in a stage might use artefacts produced by another activity in a previous stage.

Stage 1: Safety assurance development and safety case scoping for the system.
Stage 2: ML Safety requirements development and validation. Create a structured argument, and gather initial evidence to justify ML safety requirements.
Stage 3: Data requirements development and data management. Create a structured argument and gather evidence to justify data requirements.
Stage 4: Model learning and testing. Create a structured argument, and gather evidence to justify that ML model meets the safety requirements
Stage 5: Model verification. Create a structured argument, and gather evidence to demonstrate ML meets safety requirements.
Stage 6: Model deployment. Create a structured argument, and gather evidence to demonstrate ML model continues to meet the safety requirements.

Each stage is linked to a 'Feedback' loop which enables iterative and incremental development. We further align ML, agile, systems, and safety thinking next in our description of AgileAMLAS.

4. Agile for AMLAS - AgileAMLAS

Systems thinking [48] focuses on understanding systems and components, their behaviour, and how they interact. For ML, this iterates through hypothesis[3] testing, experimentation, and measurement — and aims to improve system performance. Agile thinking aims to make systems and components more adaptable to uncertainty and thus safer. Using these as our basis, we develop an 'AMLAS thinking' approach that focuses on systems, components, and their safety. We take further inspiration from Cleland-Huang and Vierhauser [20],Myklebust et al. [49],Vuori [50] and Atwal [51] who have already adapted Agile for SCSs and data processing respectively, and from Liu et al. [41]'s deep learning software lifecycle. Vuori [50] relates the Agile lifecycle to BSI [25]. ISO [52] safety standard provides guidance on design, validation, and verification, and applies to ML components [53] along with iso [7].

Integrating Agile with AMLAS brings flexibility and adaptability while maintaining the ability to use traditional safety engineering practices. It emphasises iteration, traceability and adaptation. Our integrated methodology allows AgileAMLAS teams to develop and assure ML components, and to combine software development with data and ML engineering, and with safety engineering.

4.1. AgileAMLAS team structure

The AgileAMLAS Team has a quality and safety-focused culture and the team members work together to deliver safe ML using AgileAMLAS. Dey and Lee [16] observe that delivering safe ML is a multi-disciplinary challenge requiring collective intelligence and coordinating many domain areas. To achieve such 'collective knowledge engineering', we have refined the AgileAMLAS team composition during our recent projects to comprise the scientists, engineers and

[2] adapted from https://www.leanix.net/en/wiki/trm/software-artifacts (accessed 23 Jun 2025)

[3] An ML hypothesis is a set of parameters characterising the behaviour of the model

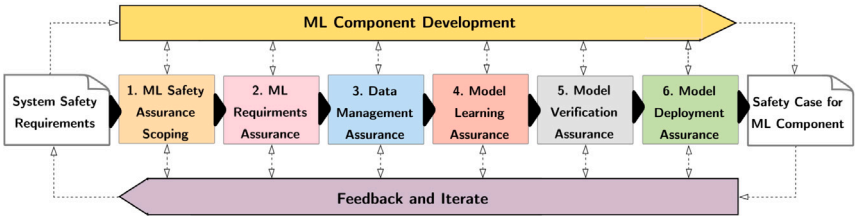

Fig. 2. Overview of the AMLAS Process from Hawkins et al. [5].

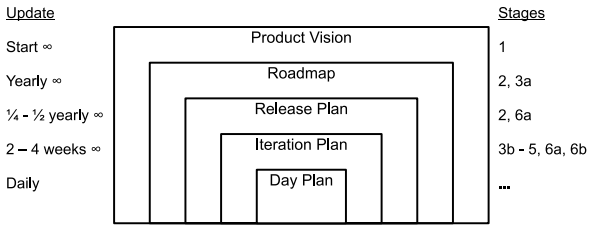

Fig. 3. The Five Cycles of Agile Planning aligned with AgileAMLAS. ∞ indicates continually updated and versioned at the relevant time-scale. Product Vision is updated as needed. Note: 'Daily' is indicative, with the frequency varying between daily scrum [46], through 3 times per week, to weekly meetups [47] according to what suits the team and application.

Safety Requirement 1

The autonomous aerial vehicle must not land when an object is present in the landing zone (with a target frequency of no more than once per 1000 landings when an object **is** present)

Safety Requirement 2

The autonomous aerial vehicle must identify that an object **is** present in the landing zone with a probability of at least 99.9%

developers who take the product requirements and transform them into Agile items to be designed, built, tested, deployed and operated. Our team needs software developers and testers; the operations team, data engineers, ML engineers, ML safety specialists; and safety verifiers, validators and assessors. We propose that verification and validation is performed by independent parties rather than the original developers who may suffer from confirmation bias [54]. To ensure safety requirements are met, we recommend dedicated safety assurance and reliability roles: one or more safety analysts and testers, and a reliability, availability, maintainability and safety (RAMS) engineer to verify safety compliance [32].

We only consider a single team project here but Agile can accommodate cross-team collaborations for large projects and products.

4.2. AgileAMLAS product

For AgileAMLAS, we need to consider if the ML model will be used for real-time inference or for off-line learning. AMLAS focuses on off-line learning as this is currently the predominant application of ML in autonomous systems [5]. However, other types of ML such as continuous learning: both batch updating [55] and online (real-time) learning [56], and reinforcement learning (RL) [57,58] may also benefit from this guidance, particularly with regard to data, model and platform management, and incorporating safety requirements.

The product generated by the AgileAMLAS methodology comprises:

- data,
- the ML model,
- any associated hardware or software for integrating the ML component into the overall system,
- traceability and versioning information (see Section 5.3.1), and
- the safety case justifying how the safety requirements are met.

4.2.1. Safety requirements

Safety requirements are requirements defined for the purpose of risk reduction, and such requirements aim to ensure the system does not cause harm. Safety requirements are elicited to mitigate identified hazards, and are further derived at increasing levels of granularity as the design progresses. Each requirement is integrated into the system's lifecycle, from design to decommissioning. Two examples are:

5. AgileAMLAS lifecycle

We can align the DevOps lifecycle (design, develop and operations) with MLOps and SafetyOps, see Table 1. We derived the steps for SafetyOps ourselves from experience with assuring ML and autonomous robots. To align and integrate AMLAS with SafetyOps, MLOps and DevOps, we incorporate new stages and objectives, and add new input and output artefacts to AMLAS giving 8 stages. Myklebust et al. [39] have 10 stages in their safety case DevOps approach. They have separate "release" and "approve" stages compared to our combined "release and approve" stage as we felt these happen together. They also include "infosec" (information security) - we focus on safety in this paper and do not consider security which needs to integrate its own systematic development process. Security failures can be causal factors to a hazard, however.

The MLOps lifecycle decomposes into five nested cycles of Agile planning: Product Vision, Roadmap, Release Plan, Iteration Plan, and Day Plan, (see Fig. 3). Each cycle has a different time span as shown and relates to AgileAMLAS stages in the following sections. The Iteration Plan specifies the product development iterations. For example, in the Scrum framework [46], each iteration is called a sprint and lasts 2–4 weeks.

AMLAS artefacts are needed for the safety case. The safety case documentation should be kept simple and be traceable; all claims should be linked to justifications and evidence, as advocated by Hanssen et al. [32]. Links should be included in the safety case documentation to the software code, ML data version, ML model version, ML configuration version, output logs, software documentation and ML documentation for cross-referencing, forward and backward traceability, and to prevent duplication of effort [45].

The following sections are organised hierarchically with a top level of design, develop and operations and subdividing these to align with the 8 steps in the DevOps, MLOps and SafetyOps lifecycles in Table 1 and Fig. 4. We also integrate the 5 nested Agile planning cycles in Fig. 3.

5.1. DevOps overview — design, develop, operations

- **Design** encompasses the **product vision, roadmap** and **release plan** for the ML component. These are refined at the appropriate

Table 1
How the DevOps, MLOps and SafetyOps lifecycle steps align with AgileAMLAS stages (see also Fig. 4). Relating to the design, develop and operations phases of DevOps/MLOps/SafetyOps: Step 1 is design, steps 2-4 are develop and operations covers steps 5-8. EDA is exploratory data analysis. Inference is the process of using a trained ML model to make predictions or decisions on new, unseen data.

Stage	DevOps	MLOps	SafetyOps	AgileAMLAS Stage
1	Plan	EDA & Plan	Analyse & Plan	A1, A2, A3a
2	Code	Data Prep	Develop	A3b
3	Build	(Re)Train	Build	A4
4	Test	Test & Validate	Verify & Validate	A5
5	Release	Package	Review & Approve	A5
6	Deploy	Deploy	Commission	A6a
7	Operate	Inference	Operate	A6a
8	Monitor	Monitor	Monitor & Maintain	A6b

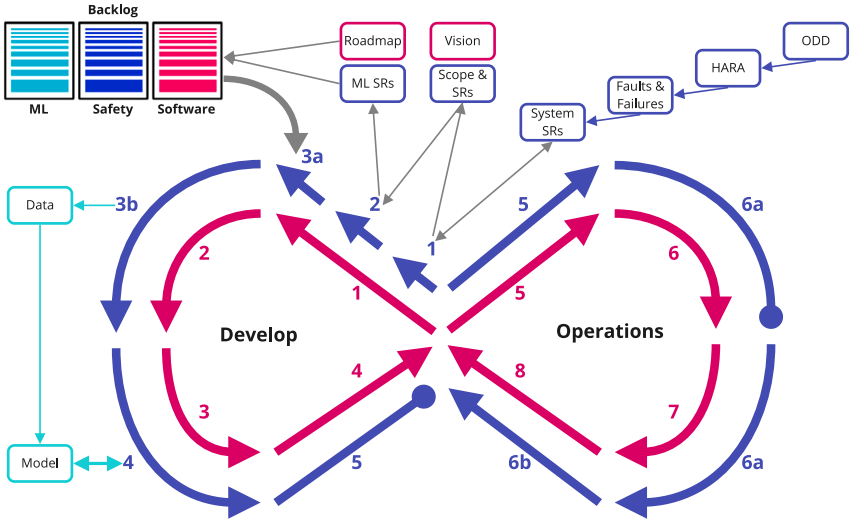

Fig. 4. AgileAMLAS Lifecycle: the 8 DevOps steps in red, AgileAMLAS stages in dark blue, and the key AgileAMLAS artefacts in boxes (with ML artefacts in cyan,). The blue boxes are the main inputs for safety assurance (see Section 5.1) which feed into DevOps step 1. The roadmap, product backlog and safety requirements feed into AgileAMLAS stages 3a–6b. The main backlog is three sub-backlogs comprising ML, safety and software tasks, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

time scale in Fig. 3. Safety assurance processes vary but typically produce the operational design domain (ODD) of the autonomous system, hazard analysis and risk assessment (HARA), and a faults and failures (causal) analysis to identify causes of hazards and risks. Safety assurance methods are then used [58,59] to derive the safety requirements (goals) to mitigate the risks and prevent harm, see Fig. 4. AMLAS assumes this step has completed [6].

  - DevOps 1: Plan and MLOps 1: EDA and Plan

- **Develop**. Development of the ML component covers DevOps/MLOps stages 2–4 (listed below) which align with AgileAMLAS stages 3b - 5 as in Fig. 4. The first iteration lays the foundation for development. It is when the software architecture is planned (including the data engineering, model training pipeline and the functional and safety aspects of the architecture). This iteration plans which product features to prioritise to give maximum customer value and to prioritise safety. Hence, it should identify the relevant safety requirements, setup the development environment, and define the delivery/deployment/operating environment. This iteration is also when the data scientists identify the candidate ML algorithms most suitable for the data types, safety requirements, the business, and its customers. Subsequent development iterations refine and improve.

  - DevOps 2: Code and MLOps 2: Data Preparation
  - DevOps 3: Build and MLOps 3: (Re)Train Model
  - DevOps 4: Test and MLOps 4: Test & Validate

- **Operations**. AMLAS **Stage 6** deploys the model to production in one final step. DevOps/MLOps use stages 5–8 (see below). AgileAMLAS aligns **stages 5, 6a and 6b** which are incremental and iterative. AgileAMLAS Stage 5 tested and now releases the product. **6a** deploys and operates an initial minimum viable product to production as soon as practical and then incrementally adds features. Each iteration also refines, reviews, and approves the system safety case. Vuori [50] identified this incremental release as a benefit of Agile development of SCSs as it increases safety testing, evidence gathering opportunities, and promotes increased customer involvement. **6b** adds system monitoring to AMLAS to analyse and assure the product during operations.

  - DevOps 5: Release and MLOps 5: Package
  - DevOps 6: Deploy and MLOps 6: Deploy
  - DevOps 7: Operate and MLOps 7: Inference
  - DevOps 8: Monitor and MLOps 8: Monitor

We now detail the stages for AgileAMLAS aligned with DevOps and MLOps stages for context. AMLAS denotes artefacts as [A^+] (one or two upper-case letters). We add new artefacts which are denoted with 3 upper-case letters.

5.2. DevOps 1: Plan - AgileAMLAS stage 1: Planning & scoping

Stage 1 provides a baseline for development, determines the safety scope and context; and the safety and reliability levels which influence the performance, quality and safety assurance of the product. It should establish clear objectives and plan the software and safety case development to mitigate the identified hazards, and meet the relevant safety standards, requirements and guidelines for the domain and application as advocated in Myklebust et al. [45].

For systems underpinned by ML, this stage frequently involves a review of the relevant literature to guide development and provide baselines for comparison.

Stage 1 of AgileAMLAS adds Objective 4 and the *[AAA]* product vision to AMLAS. These artefacts identify the requirements of the product: its purpose, features and functionality and guide development.

AgileAMLAS Stage 1 takes in the high level *[A]* system safety requirements and *[C]* system description; identifies the *[B]* system's operational design domain (ODD); develops an initial (HARA) hazard analysis with system risks and their causes (failures and faults); and plans how safety is assured using the *[F]* scope of the component's safety assurance, i.e., how the safety requirements will be met and how they will contribute to the safety case. Lindvall et al. [60] argue that Agile methods are ideal for developing these. Stage 1 outputs the *[AAA]* vision, the *[E]* allocated safety requirements (those relevant to the ML component) and *[G]* safety assurance scoping arguments for the ML.

Stage 1: ML Planning and Scoping

1. Define the scope of the safety assurance,
2. Define the scope of the safety case,
3. Specify the relevant contextual information.
4. Identify the product's purpose, features and functionality.

Inputs	Outputs
[A] System Safety Reqs,	[AAA] Product Vision^{v}
[B] System ODD,	[E] Allocated Safety Reqs
[C] System Description,	[G] Safety Scoping Arg
[D] ML Component Description,	
[F] Scoping Arg Pattern	

Artefact^{v} denotes artefacts that are versioned; each version has a unique ID.

5.3. DevOps 1: Plan - AgileAMLAS stage 2: Roadmap

The *[AAA]* product vision leads to a **roadmap** which aligns with **Stage 2** of AMLAS. The roadmap focuses on the requirements, objectives and justification. It does not provide details of specific components or technologies, but states how the product will be developed and tested (test plan), and provides a long-term schedule of major milestones (release plan).

Thus, we adapt AMLAS **Stage 2** adding new **Objectives 4, 5 and 6** for defining the testing scope and goals, testing framework, and testing priorities in the test plan, two new inputs, and three new outputs. It now takes in *[E]* allocated safety requirements and their *[I]* argument patterns plus *[BBB]* system testing requirements which include KPIs and measurable acceptance criteria that the product must meet. It outputs *[H]* ML safety requirements derived from the *[A]* system safety requirements which have been *[E]* allocated to the ML component being developed, their *[J]* validation results and *[K]* evidence that safety requirements are met — achieved through a *[CCC]* test plan, *[DDD]* release plan and *[EEE]* product backlog.

The *[CCC]* test plan comprises general software engineering analyses, such as KPIs, testing, acceptance criteria and review, combined with updated hazard/risk analyses and safety assessment. This enables analysis of the product's achieved levels of safety, and an initial identification of what additional safety controls may be required. The milestones in the *[DDD]* release plan reflect when new product versions are released to customers. For SCSs, *[DDD]* also identifies where and when verification and validation are needed for safety assurance of each product release as this is time-consuming and needs to be planned ahead. The roadmap is updated as in Fig. 3; allowing product changes to be accommodated during development [26].

Stage 2: Roadmap (including requirements and objectives)

1. Develop the ML safety requirements,
2. Validate the ML safety requirements,
3. Create an assurance argument,
4. Define the test scope and test goals.
5. Determine the test environment, methods and tools.
6. Set testing priorities considering safety, customer, and user requirements.

Inputs	Outputs
[E] Allocated Safety Reqs,	[H] ML Safety Reqs,
[I] ML Safety Reqs Arg Pattern,	[J] ML Safety Reqs Validation Results,
[AAA] Product Vision^{v}	[K] ML Safety Reqs Arg,
[BBB] System Testing Reqs,	[CCC] Roadmap: Test Plan^{v},
	[DDD] Roadmap: Release Plan^{v},
	[EEE] Product Backlog^{vs},

Artefact^{s} denotes artefacts updated sequentially at least once per iteration.

5.3.1. Reproducibility and versioning

Reproducibility is critical for software development, but this becomes challenging when developing safety-critical ML systems with respect to the traceability of safety artefacts, data, models and platforms [16]. Simply storing versions of the data and models does not provide the whole story. Throughout development, we must version the AgileAMLAS artefacts as well as the software, data, the data schema, models, the algorithm and parameters of the model (ML hypothesis), the resources needed to process the data and train the model, the log files, and the environment, along with detailing and versioning all design decisions and rationales. Importantly, versioning gathers evidence for safety assurance that each risk is mitigated with safety constraints for this version and provides evidence that the system meets its safety requirements.

5.3.2. Product backlog, epics and PBIs

The *[EEE]* **Product Backlog** is a prioritised list of required development **items** and is the main knowledge base for Agile development. For AgileAMLAS, we advocate one backlog with three sub-backlogs: **ML, safety assurance and software** as shown in Fig. 4. These sub-backlogs have different development requirements and provide clarity for the team. This separation aligns with other safe Agile advocates, e.g., [12]. We refer to the sub-backlogs collectively as the **backlog** - they function as a single unit.

Items can be **epics** - large requirements (1–3 months work) that will be subdivided into items when the necessary knowledge is available. Epics have acceptance criteria that define their requirements including safety requirements from *[A],[E]* and *[H]*. In conventional Agile software development, an epic will be developed until it meets its acceptance criteria. For MLOps and safety assurance, an epic can have a positive or negative outcome depending whether the ML model meets its acceptance criteria. For example in Safety Requirement 2 in Section 4.2.1, if the ML component must have an accuracy of 99.9%, then a ML model with inference accuracy of 95% will not be deployed to production.

We do not prescribe a format for backlog items as teams tend to have their own format of choice. We instead provide general guidelines. The following example epics and items are for a safe ML component to land an autonomous aerial vehicle using the vehicle's sensor data and ML to detect objects and localise [61].

ML Epic

Identify the object detection algorithm that most accurately detects objects in the AAV's landing zone

Safety Assurance Epic

To achieve safe landing, there must be redundancy in the AAV's object detector in case of failure, to ensure 99.9% detection.

Software Epic

Develop a processing pipeline for an object detector in image data with 99.99% availability (up-time).

When the team has sufficient knowledge, epics are broken down into **Product Backlog Items (PBIs)**. In Agile software development (and our software sub-backlog), these are user stories (requirements), written from the user's perspective to describe how each product feature will provide value to the customer. However, these are subjective so may not be sufficient for SCSs where safety requirements have defined design and implementation requirements. Hence, ML and safety assurance PBIs provide detailed testing, verification and validation specifications including acceptance criteria so it is clear what to develop and how to assess it (including safety requirements from *[H]*) as advocated by [12,20]. The following example PBIs are for a safe ML component to land an autonomous aerial vehicle using the vehicle's sensor data (LiDAR in this case) for extero-perception to provide object detection.

ML PBI

Implement most accurate algorithm to process LiDAR scan data and geolocate objects.

Safety Assurance PBI

To argue safe landing, collect collision distances to objects calculated using LiDAR data so we can assure error is no more than 5 cm of x,y,z.

Software PBI

Develop data processor to ingest the LiDAR scan data or output an error log in case of data error, for 100% of LiDAR scans.

Before and during each development iteration, the team meet to refine, re-prioritise and re-estimate the backlog. This is based on feedback from recent development plus customer feedback and new requirements (including new safety requirements [62]). This refinement can be small adjustments through to large changes. For example, in **Stage 4**, the team can initiate another research and development iteration to optimise the current ML model, or abandon it and train a new model if it is not meeting expectations. In **Stage 6b**, monitoring of the deployment may trigger maintenance iterations varying from small data quality investigations to full model redevelopment.

5.4. DevOps 1: Plan and MLOps 1: EDA and plan AgileAMLAS stage 3a: Development planning

The **roadmap** describes data acquisition and pre-processing. It incorporates the first part of AMLAS Stage 3. Thus, we split AMLAS **Stage 3** into AgileAMLAS **Stage 3a and 3b**. **Stage 3a** is initial planning and elicits the data requirements (design). MLOps uses multiple datasets and configurations to ensure data coverage [36]. This is vital for safety assurance to assure against errors at run-time. AMLAS only considers a single dataset, whereas we outline where to use multiple data splits in **Stage 3b** (see Section 5.5).

AgileAMLAS **Stage 3a** maps the *[H]* ML component's safety requirements plus its *[R]* data argument pattern (data schema) and the *[CCC]* test plan to *[L]* data requirements which are listed in a *[M]* justification report. Note, having the justification coupled with each requirement conforms to the Agile Manifesto edict of only documenting what is necessary; we are not simply documenting a list of requirements but why they are needed — rationale which are auditable and configuration-controlled. Stage 3a also elicits and prioritises *[FFF]* ML hypotheses and the *[HHH]* ML experiment specifications needed to achieve those goals (described next). It updates the *[EEE]* product backlog and produces a *[III]* iteration plan for the next development iteration.

Stage 3a: Development Planning

1. Develop data requirements.

Inputs	Outputs
[H] ML Safety Reqs,	[L] Data Reqs^{v},
[R] ML Data Arg Pattern,	[M] Data Reqs Justification Report^{v},
[CCC] Test Plan^{vs},	[FFF] ML Hypotheses^{s},
	[HHH] ML Experiments Specs^{vs},
	[EEE] Product Backlog^{vs},
	[III] Iteration Plan^{vs},

5.5. DevOps 2: Code and MLOps 2: Data preparation AgileAMLAS stage 3b: Data development

Stage 3b in AgileAMLAS has an additional objective to develop a baseline ML model that satisfies the safety requirements and provides comparison for model development in **Stage 4** whereas AMLAS only considers a single model that has been pre-selected. In AgileAMLAS, we iteratively develop the model using multiple data configurations, multiple model configurations (parameter settings) and even multiple algorithms if necessary.

The first part of Stage 3b is data refinement to assure the data. The team collaborate to produce and test a data processing pipeline. The data engineers (in conjunction with safety engineers) determine if the product specification is achievable with any existing data; and what data are needed to meet the acceptance criteria of the ML KPIs in *[CCC]*, assure the data meets its *[L]* requirements, and produce *[N]* train, *[O]* test and *[P]* verification datasets. This data generation is logged in *[Q]* to provide evidence that the data are sufficient (complete, accurate and correct). The data are then validated to ensure the dataset meets the *[L]* data requirements with the validation results contained in *[S]*. The *[T]* data arguments justify that the data used in the development and verification of the model are sufficient.

N.B., ML engineers often use 10-fold cross-validation to train and test their models. This splits the data into 10 subsets, using 9 for training and 1 for testing in rotation. This can still be accomplished here but there would be 10 instances of [N] and [P] (paired), with each pair associated with a separate instance of [O] and carefully versioned for reproducibility.

Stage 3b also updates the *[EEE]* backlog, plus *[HHH]* specifications for ML experiments and produces our *[GGG]* baseline ML model which meets the *[H]* ML safety requirements and the KPI acceptance criteria in *[CCC]*.

Stage 3b: Data Development and Management

1. Generate data sets, providing a rationale,
2. Analyse the data sets for sufficiency,
3. Develop baseline model using [N] that meets the KPI acceptance criteria in [CCC] and ML safety requirements [H].
4. Create an assurance argument,

Inputs	Outputs
[L] Data Reqs^{U}	[N] Development Data^{VS},
[JJJ] Iteration Backlog,	[EEE] Product Backlog^{VS},
[KKK] Iteration Experiments Specs,	[O] Internal Test Data^{VS},
[H] ML Safety Reqs,	[P] Verification Data^{VS},
[CCC] Test Plan^{VS},	[Q] Data Generation Log^{VS},
	[S] ML Data Validation Results^{VS},
	[T] ML Data Arg^{VS},
	[HHH] ML Experiments Specs^{VS},
	[GGG] ML Baseline^{U},

Stage 3b of AgileAMLAS is iterative, see Fig. 4, so each iteration outputs *[N, O, P, Q, S, T, EEE, GGG, HHH]* i.e., the data and its provenance information with a unique and linked (version) identifier. Versioning must be tightly integrated so the model can be reproduced precisely. Thus the provenance is used for creating an audit trail for production during Stages 3b, 4, 5, 6a and 6b, for understanding the behaviour of the model, its environment (context) and for providing safety assurance evidence.

5.6. DevOps 3: Build and MLOps 3: (Re)train model AgileAMLAS stage 4: Model development

Once the data pipeline is ready, the ML engineers train models in **Stage 4**. The ML engineers use the *[N]* training data to generate a set of models. Each *[V]* model is tested against the *[CCC]* test plan KPIs and *[H]* safety requirements using *[O]* internal test data to identify the "best"[4] model. The *[W]* ML learning argument pattern details what needs to be demonstrated to prove the ML model is sufficient to meet the *[H]* ML safety requirements. This model development is comprehensively described in *[U]* including all version control information. Model testing is logged in detail in *[X]* for reproducibility. The *[Y]* ML learning argument justifies that the *[V]* model meets its *[H]* safety requirements and explains trade-offs, assumptions and uncertainties. Each model developed in **Stage 4** should be considered a configuration item. In ML engineering, it is not necessarily the final model configuration that yields the "best" model (to be used in production) so previous iterations of **Stages 3b, 4 and 5** must be fully versioned and reproducible, see Section 5.3.1.

AgileAMLAS **Stage 4** has an additional objective to re-engineer the data produced in **Stages 3a and 3b** to improve model performance, such as adding features or more data points to ensure relevance, balance, completeness and accuracy [40]. Thus, data versioning is vital for traceability.

The knowledge gained from Stage 4 updates the *[EEE]* product backlog.

[4] In safety assurance, the best model is the safest as defined by a set of metrics derived from *[W]* not just the most accurate.

Stage 4: Model Development

1. Develop the ML model so ML safety requirements are satisfied,
2. Use internal test data to assess the ML model against KPIs,
3. Create an assurance argument,
4. Re-engineer the data from Stage 3 to improve model performance, e.g., data cleaning, feature engineering or bias reduction,
5. Version the data and model to create an audit trail for production.

Inputs	Outputs
[H] ML Safety Reqs,	[U] Model Development Log^{VS}
[JJJ] Iteration Backlog,	[EEE] Product Backlog^{VS},
[N] Development Data^{VS},	[V] ML Model^{VS},
[O] Internal Test Data^{VS},	[X] Internal Test Results^{VS},
[W] ML Learning Arg Pattern^{VS},	[Y] ML Learning Arg^{VS},
[CCC] Test Plan^{VS},	
[LLL] Iteration ML Hypotheses,	[FFF] ML Hypotheses^{U},
[KKK] Iteration Experiment Specs,	
[VVV] ML Baseline^{U},	

5.7. DevOps 4: Test - AgileAMLAS stage 5: Verification and validation

The team must test rigorously throughout the AgileAMLAS lifecycle to identify uncertainty and risk [63]. Testing should be at the functional, structural and statistical levels incorporating hazard and risk analysis and safety assessment to mitigate product/business/technological risks. Testing provides feedback, identifies hidden assumptions, provides evidence regarding the state of the product and can enforce an update (new version) of the test plan or safety documentation if necessary. Testing should use defect tracing to track each defect found, its source, when it was detected, when it was resolved and how it was resolved — outputting this in AgileAMLAS stage 3b artefact *[Q]*, stage 4 *[X]*, stage 5 *[AA]*, stages 6a and 6b *[DD][FF]*. This captures evidence for the safety case and that the product meets the safety requirements in *[A][E][H]*. SCSs have tended to postpone testing until the later stages of development [33] which delays analysis and feedback and may lead to changing large portions of code.

AgileAMLAS testing is detailed in the *[CCC]* test plan and covers:

(1) Unit testing (each software module is tested in isolation by developers).
(2) Integration testing (related modules are grouped and tested by developers and testers). It tests functional requirements and interfaces.
    (a) This should include regression (change) testing which is vital for safety assurance as it ensures any software changes have not adversely affected the existing software.
(3) System testing uses functional and non-functional testing in a production environment. System and safety testing may be one iteration behind development as it is testing the released system not the latest development.
(4) Validation: acceptance testing releases alpha and beta versions of the product for users to evaluate and provide feedback. In SCSs, this has to be managed very carefully, the released product must meet its safety requirements and be compliant with relevant standards.

Testing takes in the *[CCC]* test plan, the *[FFF]* ML hypotheses and *[H]* safety requirements, *[HHH]* experiment specs and *[BB]* verification arguments and applies *[P]* verification data to the *[V]* model. The *[Z]* inference verification results are output and the process is logged in *[AA]*. The *[CC]* development safety justification demonstrates the relationship between the testing results and the *[H]* safety requirements and explains any trade-offs, assumptions or uncertainties. Stage 5 updates the *[EEE]* product backlog.

Stage 5 links the safety requirements, and KPIs and acceptance criteria in the test plan to this version of the *[V]* model for version control and reproducibility. If the verification data are re-engineered in an iteration of Stage 3b (for example, new data features or new data points added) then it should be fully versioned and changes logged in *[AA]* and explained in the *[CC]* safety justification.

Stage 5: (Model) Verification and Validation; and Release

1. Demonstrate that the model will meet the ML safety requirements,
2. Create an assurance argument - safety case.
3. Safety case submitted, reviewed and approved.

Inputs	Outputs
[FFF] ML Hypotheses^{v},	[EEE] Product Backlog^{vs},
[H] ML Safety Reqs,	[Z] ML Verification Results^{vs},
[P] Verification Data^{v},	[AA] Verification Log^{vs},
[V] ML Model^{vs},	[CC] ML Verification Arg^{vs}
[BB] ML Verification Arg Pattern^{vs},	
[CCC] Test Plan^{vs},	
[HHH] ML Experiments Specs^{vs},	

5.8. DevOps 5: Release - AgileAMLAS stage 5: Release

This stage includes final quality assurance, reviewing and approving the safety case, release planning, and scheduling. Releases can be internal (released within the organisation for quality assurance) or external releases to customers. External releases must meet the specific requirements for the domain, application and even the geographical region of operation, so the release plan *[DDD]* developed in Stage 2 must take these requirements into consideration. *[DDD]* must ensure that the ML component and safety case comply with the relevant safety standards such as [52] and any pertinent legislation. In certain industries the safety case may need regulatory approval before the ML component is signed-off and released. Hence, we have added a new objective to have the safety case approved. The AgileAMLAS team should also document in *[CC]*: a list of any restrictions for the ML component's use; the permitted application domain(s) and any limits, and software and hardware compatibility.

Once fully approved, the ML component has passed all software engineering, ML and safety assurance tests and is now prepared for production.

An extra AgileAMLAS check is to verify that the dataset used in production is the same version used for model training and verification. The production inference model outputs (*[FF]* in **Stages 6a and 6b**) must be regression tested against the outputs of the trained model (*[V]* in **Stage 4**) and the outputs of model verification (*[Z]* in **Stage 5**) to make sure the ML component works as expected using the same test dataset and meets the safety requirements.

5.8.1. CI/CD

AgileAMLAS development can use continuous integration, delivery and deployment (CI/CD) [37]. CI/CD automates the system lifecycle through a continuous build and test process.

It automatically integrates and tests new code in combination with existing code and rebuilds the product every time. Continuous delivery automatically validates and verifies new versions of the code, ML model and associated processes and deploys them to a testing environment for Stages 4 and 5. Continuous deployment automatically deploys new versions to production (Stage 6a) and monitors them (Stage 6b). For safety assurance, CI/CD should incorporate regression testing to ensure that new changes have not adversely affected the existing code's functionality and technical quality. Automated and continuous testing ensures the ML component continues to meet requirements and safety standards [43].

5.9. DevOps 6: Deploy - AgileAMLAS stage 6a: Deploy

AgileAMLAS stage 6a (which encompasses DevOps stages 6–7) is planned in the *[CCC]* test plan and *[DDD]* release plan produced in Stage 2 and Fig. 5 shows where it fits into the AgileAMLAS lifecycle.

The initial deployment of the product containing a minimum viable product (MVP) of the ML component is usually only a partial deployment to a subset of users. It provides user review and code testing in test platforms with an operating system, packages, databases, or tools that match the final production platform. A partial deployment reduces the operational risk of the rollout. It allows experimentation through making changes and observing outcomes and enables the calculation of benefit measurement (KPIs) comparing users exposed to the MVP and those who are not. Once the data and model pipelines meet the requirements for the MVP and successfully pass tests, they are ready to be deployed in a production environment.

5.10. DevOps 7: Operate - AgileAMLAS stage 6a: Operate

The operation step is where customers use the product. Here, the team must collect feedback: verifying using system testing in *[DD]* and *[FF]*, and validating against the user requirements in *[DD]*. These combine automatic logging and user feedback to identify potential product and safety improvements. Any feedback is integrated into future development iterations as improvements.

5.11. AgileAMLAS stages 6a and 6b

Stage 6a covers DevOps stages 6–7 and **stage 6b** covers DevOps stage 8. Stages 6a and 6b consider the *[C]* system, *[A]* safety requirements, *[B]* product environment, *[EE]* operation scenarios and *[GG]* ML deployment argument pattern. Stage 6a: **Objective 1** deploys and operates the ML component coupled with **Objective 2** to produce the *[HH]* operations safety justification (which also explains any trade-offs, assumptions or uncertainties). **Objective 4** (Stages 6b) requires that we perform validation and verification (monitoring) through-life, as shown in Figs. 4 and 5. Both 6a and 6b output *[DD]* error logs and collect *[FF]* test results with 6b collecting them throughout *Operations*.

Stages 6a and 6b continue to refine the *[EEE]* product backlog to keep development, safety, risk and hazard knowledge up-to-date as recommended by [11,64].

Stage 6a: Release; Deploy; Operations.
Stage 6b: Monitoring

1. 6a - Integrate the ML component into the target system,
2. 6a & 6b - Create assurance argument for continued usage - operations safety case,
3. 6a - Operations safety case submitted, reviewed and approved.
4. 6b - Verify that safety requirements are satisfied during operation.

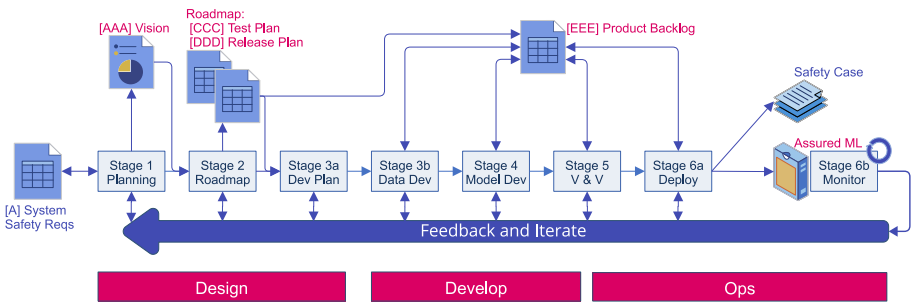

Fig. 5. Schematic of AgileAMLAS, showing the main processes, inputs and outputs of our methodology and highlighting where these fit with design, develop and operations in DevOps. DevOps and the DevOps artefacts are labelled in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Inputs	Outputs
[CCC] Test Plan^{v},	[DD] Erroneous Behaviour Log^{vs},
[DDD] Release Plan^{v},	[FF] Integration Testing Results^{vs},
[A] System Safety Reqs,	[HH] ML Deployment Arg^{vs}
[B] Environment Description,	
[C] System Description,	
[JJJ] Iteration Backlog,	[EEE] Product Backlog^{vs},
[V] ML Model^{vs},	
[EE] Operational scenarios.	
[GG] ML Deploy Arg Pattern,	

The complete through-life safety case comprises the set of safety artefacts [A] to [HH] produced in the AgileAMLAS process and covers design, development and operations. A design safety case is approved in stage 5 and the operations safety case is approved in stage 6a (Objective 3).

5.12. DevOps 8: Monitor - AgileAMLAS stage 6b: Monitoring

Monitoring uses Objective 4, plus the inputs and outputs above. The activities include monitoring the data, model and safety compliance (whether it meets its safety requirements defined in *[A],[E],[H]*). The monitoring uses KPIs detailed in the *[CCC]* test plan applied to this version of the *[V]* model and linked data. It uses metrics (with acceptance criteria) such as processing time; data quality; error counts; data ingestion correctness and coverage; ML component latency; end-to-end latency; and data, model and prediction availability (as up-time is important for SCSs). These are applied during integration testing of the whole processing pipeline (including its *[C]* platform and *[EE]* operating scenarios) not just the software development. Stages 6a and 6b integration testing is different from the software integration testing in Section 5.7 which analyses software module integration rather than full system integration. Vuori [50] recommends exploratory testing to learn model behaviour, systematic testing via test cases specified using the *[V]* model and its behaviour, then more detailed explorative testing to find any defects. Regular user testing is also important to elicit feedback. This can be automated by building feedback loops into the product to monitor user acceptance of the model's inference (e.g., whether users override an autonomous vehicle's autopilot).

The team also monitor the distribution of the data flowing through the model. Changing data distributions can decrease model performance. Detecting this is important for assuring safety [40]. The data distribution may shift (data drift) or the relationship between new input and output data may change (concept drift) [34]. There are algorithms to detect drift and data preparation methods can overcome it, see Lu et al. [65].

The testing and monitoring results are documented in *[FF]* with any issues or errors logged in *[DD]* using informative alerts that explain the issue or map issues to their effect on the model so developers and engineers can diagnose and resolve the issue [66]. The *[EEE]* backlog may be updated as new knowledge arises from testing and monitoring and the safety case may be updated including the *[A]* system requirements.

5.13. Decommissioning

At some point, it will be necessary for the product to enter the retirement phase to decommission the model and pipelines. This is not covered here.

6. Conclusion

In this paper, we integrated an Agile methodology and Agile thinking with an AMLAS [5] methodology for assuring the safety of ML components in autonomous systems through-life. End-to-end approaches across lifecycle stages are missing from the literature [6]. AgileAMLAS develops the software, ML components and their associated safety cases in parallel and aligns with standards such as ISO PAS 8800. We guide the design and development of ML components in safety-critical domains and detail the context, roles, processes and workflow of the development process.

We are not advocating Agile development as a one-size fits-all solution for safety assurance of ML. There is no single solution and all solutions require adaptation and careful consideration. For example, Agile is best suited to complex products. Hence, our approach is a palette of tools that can be used and adapted by teams to suit: their requirements, the product's requirements and the business's requirements while meeting the customer's and user requirements, and relevant safety regulations and standards.

In a recent project to safety assure an autonomous mobile robot [67], we found that: software and hardware development suited Scrum sprints [46] while preparing the associated safety artefacts suited Kanban [68] - breaking document writing into tasks of 2 weeks or less did not work for the project. We used a single product backlog with PBIs for software, hardware, ML and safety assurance which fed two task boards: a Scrum board for hardware, software and ML; and a Kanban board for developing safety assurance artefacts. The latter still had deadlines and was reviewed as part of the bi-weekly sprint review ceremony.

In future work, we will apply AgileAMLAS in more real-world projects and investigate how it fits into whole system development and system safety assurance. This may well require an industry partner to change their development practices thus, introducing uncertainty and potentially resistance to change. They may need to accommodate the flexibility, iteration and dynamism of Agile practices where they may have previously used strict waterfall approaches with all safety requirements fixed and pre-specified. This will need to be independently assessed to objectively assess AgileAMLAS's utility.

CRediT authorship contribution statement

Victoria J. Hodge: Writing – original draft, Validation, Methodology, Conceptualization. **Matt Osborne:** Writing – review & editing, Validation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the Centre for Assuring Autonomy, a partnership between Lloyd's Register Foundation and the University of York (https://www.york.ac.uk/assuring-autonomy/).

Data availability

No data was used for the research described in the article.

References

[1] Perez-Cerrolaza J, Abella J, Borg M, et al. Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey. ACM Comput Surv 2024;56(7). http://dx.doi.org/10.1145/3626314.

[2] Hawkins R, Habli I, Kelly T. The principles of software safety assurance. In: 31st international system safety conference. 2013.

[3] Boehm BW. A spiral model of software development and enhancement. IEEE Comput 1988;21(5):61–72. http://dx.doi.org/10.1109/2.59.

[4] Pedersen Notander J, Höst M, Runeson P. Challenges in flexible safety-critical software development–an industrial qualitative survey. In: International conference on product focused software process improvement. Springer; 2013, p. 283–97.

[5] Hawkins R, Paterson C, Picardi C, Jia Y, Calinescu R, Habli I. Guidance on the assurance of machine learning in autonomous systems (AMLAS). 2021, CoRR, arXiv:2102.01564. URL https://arxiv.org/abs/2102.01564.

[6] Kuehnert B, Kim RM, Forlizzi J, Heidari H. The "who", "what", and "how" of responsible AI governance: A systematic review and meta-analysis of (actor, stage)-specific tools. 2025, arXiv preprint arXiv:2502.13294.

[7] ISO PAS 8800:2024. Road vehicles—Safety and artificial intelligence. 2024, URL https://www.iso.org/standard/83303.html (Accessed 23 June 2025).

[8] Beck K, Beedle M, Van Bennekum A, et al. Manifesto for agile software development. Snowbird, UT; 2001.

[9] Redmill F. Agile methods for developing safety-related software? Safety-Critical Syst Club Newsl 2015;24(2):1–24.

[10] Meyer B. Agile: The good, the hype and the ugly. Springer International Publishing; 2014, URL https://books.google.co.uk/books?id=ELLBBAAAQBAJ.

[11] Kasauli R, Knauss E, Kanagwa B, Nilsson A, Calikli G. Safety-critical systems and agile development: A mapping study. In: 2018 44th euromicro conference on software engineering and advanced applications. 2018, p. 470–7. http://dx.doi.org/10.1109/SEAA.2018.00082.

[12] Myklebust T, Stålhane T. Functional safety and proof of compliance: Agile practices. Springer; 2021, p. 25–58. http://dx.doi.org/10.1007/978-3-030-86152-0_2.

[13] Boehm B. Get ready for agile methods, with care. Computer 2002;35(1):64–9. http://dx.doi.org/10.1109/2.976920.

[14] Doss O, Kelly T. The 4+1 principles of software safety assurance and their implications for scrum. In: Sharp H, Hall T, editors. Agile processes, in software engineering, and extreme programming. Cham: Springer International Publishing; 2016, p. 286–90.

[15] Stettina CJ, Heijstek W. Necessary and neglected? An empirical study of internal documentation in agile software development teams. In: Proceedings of the 29th ACM international conference on design of communication. 2011, p. 159–66. http://dx.doi.org/10.1145/2038476.2038509.

[16] Dey S, Lee S-W. Multilayered review of safety approaches for machine learning-based systems in the days of AI. J Syst Softw 2021;176:110941. http://dx.doi.org/10.1016/j.jss.2021.110941.

[17] Stålhane T, Hanssen GK, Myklebust T, Haugset B. Agile change impact analysis of safety critical software. In: International conference on computer safety, reliability, and security. Springer; 2014, p. 444–54.

[18] Coplien JO, Bjørnvig G. Lean architecture: for agile software development. John Wiley & Sons; 2011.

[19] Łukasiewicz K, Górski J. Introducing agile practices into development processes of safety critical software. In: Proceedings of the 19th international conference on agile software development: companion. ACM; 2018, p. 1–8. http://dx.doi.org/10.1145/3234152.3234174.

[20] Cleland-Huang J, Vierhauser M. Discovering, analyzing, and managing safety stories in agile projects. In: 2018 IEEE 26th international requirements engineering conference. 2018, p. 262–73. http://dx.doi.org/10.1109/RE.2018.00034.

[21] Islam G, Storer T. A case study of agile software development for safety-critical systems projects. Reliab Eng Syst Saf 2020;200. http://dx.doi.org/10.1016/j.ress.2020.106954.

[22] Heeager LT, Nielsen PA. A conceptual model of agile software development in a safety-critical context: A systematic literature review. Inf Softw Technol 2018;103:22–39. http://dx.doi.org/10.1016/j.infsof.2018.06.004.

[23] Paige RF, Charalambous R, Ge X, Brooke PJ. Towards agile engineering of high-integrity systems. In: International conference on computer safety, reliability, and security. Springer; 2008, p. 30–43.

[24] Nielsen PA, Heeager LT. The dynamics of agile practices for safety-critical software development. In: Proceedings of the XP2017 scientific workshops. ACM; 2017, p. 1–6. http://dx.doi.org/10.1145/3120459.3120481.

[25] BSI/IEC 61508. Functional safety of electrical/electronic/programmable electronic safety related systems. Int Electrotech Comm 2010;Parts 1–7.

[26] Gary K, Enquobahrie A, Ibanez L, et al. Agile methods for open source safety-critical software. Softw: Pr Exp 2011;41(9):945–62. http://dx.doi.org/10.1002/spe.1075.

[27] Luckcuck M. Using formal methods for autonomous systems: Five recipes for formal verification. Proc Inst Mech Eng Part O: J Risk Reliab 2023;237(2):278–92. http://dx.doi.org/10.1177/1748006X211034970.

[28] Bhattacharyya S, Davis J, Gupta A, Narayan N, Matessa M. Assuring increasingly autonomous systems in human-machine teams: An urban air mobility case study. 2021, arXiv preprint arXiv:2110.12591.

[29] Narayan N, Ganeriwala P, Jones RM, Matessa M, Bhattacharyya S, Davis J, Purohit H, Rollini SF. Assuring learning-enabled increasingly autonomous systems*. In: 2023 IEEE international systems conference. 2023, p. 1–7. http://dx.doi.org/10.1109/SysCon53073.2023.10131227.

[30] Cofer D, Amundson I, Sattigeri R, Passi A, Boggs C, Smith E, Gilham L, Byun T, Rayadurgam S. Run-time assurance for learning-enabled systems. In: Lee R, Jha S, Mavridou A, Giannakopoulou D, editors. NASA formal methods. Cham: Springer International Publishing; 2020, p. 361–8.

[31] Oldfield M, McMonies M, Haig E. The future of condition based monitoring: Risks of operator removal on complex platforms. AI & SOCIETY 2022. http://dx.doi.org/10.1007/s00146-022-01521-z.

[32] Hanssen GK, Stålhane T, Myklebust T. SafeScrum®-agile development of safety-critical software. Springer; 2018.

[33] Barbareschi M, Barone S, Carbone R, Casola V. Scrum for safety: an agile methodology for safety-critical software systems. Softw Qual J 2022;30:1067–88. http://dx.doi.org/10.1007/s11219-022-09593-2.

[34] Zeller M, Waschulzik T, Schmid R, Bahlmann C. Towards a safe MLOps process for the continuous development and safety assurance of ML-based systems in the railway domain. 2023, arXiv:2307.02867.

[35] Giray G. A software engineering perspective on engineering machine learning systems: State of the art and challenges. J Syst Softw 2021;180:111031. http://dx.doi.org/10.1016/j.jss.2021.111031.

[36] Shankar S, Garcia R, Hellerstein JM, Parameswaran AG. Operationalizing machine learning: An interview study. 2022, http://dx.doi.org/10.48550/ARXIV.2209.09125, arXiv Pre-Print.

[37] Steidl M, Felderer M, Ramler R. The pipeline for the continuous development of artificial intelligence models—Current state of research and practice. J Syst Softw 2023;199:111615. http://dx.doi.org/10.1016/j.jss.2023.111615.

[38] Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M, Crespo J-F, Dennison D. Hidden technical debt in machine learning systems. In: Proceedings of the 29th international conference on neural information processing systems - volume 2. Cambridge, MA, USA: MIT Press; 2015, p. 2503–11.

[39] Myklebust T, Stålhane T, Hanssen G. Agile safety case and DevOps for the automotive industry. In: E-proceedings of the 30th european safety and reliability conference and 15th probabilistic safety assessment and management conference (ESREL2020 PSAM15). 2020, p. 4652–7. http://dx.doi.org/10.3850/978-981-14-8593-0_3495-cd.

[40] Ashmore R, Calinescu R, Paterson C. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. ACM Comput Surv 2021;54(5):1–39. http://dx.doi.org/10.1145/3453444.

[41] Liu Y, Ma L, Zhao J. Secure deep learning engineering: A road towards quality assurance of intelligent systems. In: Ait-Ameur Y, Qin S, editors. Formal methods and software engineering. Cham: Springer International Publishing; 2019, p. 3–15.

[42] Ge X, Paige RF, McDermid JA. An iterative approach for development of safety-critical software and safety arguments. In: 2010 agile conference. IEEE; 2010, p. 35–43.

[43] Zeller M, Ratiu D, Rothfelder M, Buschmann F. An industrial roadmap for continuous delivery of software for safety-critical systems. In: 39th international conference on computer safety, reliability and security (SAFECOMP), Position Paper. Lisbon, Portugal; 2020, p. 1–4.
[44] Kim G, Debois P, Willis J, Humble J. The DevOps handbook: how to create world-class agility, reliability, and security in technology organizations. IT Revolution Press; 2016.
[45] Myklebust TS, Vatn T, Kristin DM. AI act and the agile safety plan. Springer; 2025.
[46] Schwaber K, Sutherland J. The scrum guide. Scrum Alliance 2011;21(1).
[47] Myklebust T, Okoh P. Functional safety sprints and kanban approach in practice. In: ISSS 2022 annual conference. 2022.
[48] Arnold RD, Wade JP. A definition of systems thinking: A systems approach. Procedia Comput Sci 2015;44:669–78. http://dx.doi.org/10.1016/j.procs.2015.03.050, 2015 Conference on Systems Engineering Research.
[49] Myklebust T, Stålhane T, Lyngby N. The agile safety plan. 2016, 13th International Conference on Probabilistic Safety Assessment and Management (PSAM 13), Seoul, Korea.
[50] Vuori M. Agile development of safety-critical software. Report 14. Department of software systems, Tampere University of technology, Finland. URN: , 2011, http://URN.fi/URN:NBN:fi:tty-2011061414702.
[51] Atwal H. Practical DataOps: delivering agile data science at scale. Springer; 2019.
[52] ISO 21448:2022. Road vehicles — Safety of the intended functionality. 2022, International Organization for Standardization: Geneva, Switzerland.
[53] Wang H, Shao W, Sun C, Yang K, Cao D, Li J. A survey on an emerging safety challenge for autonomous vehicles: Safety of the intended functionality. Engineering 2024;33:17–34. http://dx.doi.org/10.1016/j.eng.2023.10.011.
[54] McBride T, Lepmets M. Quality assurance in agile safety-critical systems development. In: 10th international conference on the quality of information and communications technology. 2016, p. 44–51. http://dx.doi.org/10.1109/QUATIC.2016.016.
[55] Imrie C, Howard R, Thuremella D, et al. Aloft: Self-adaptive drone controller testbed. In: Proceedings of the 19th international symposium on software engineering for adaptive and self-managing systems, SEAMS 2024. 2024, p. 70–6.
[56] Schleiss P, Carella F, Kurzidem I. Towards continuous safety assurance for autonomous systems. In: 2022 6th international conference on system reliability and safety. IEEE; 2022, p. 457–62.
[57] Hodge VJ, Hawkins R, Alexander R. Deep reinforcement learning for drone navigation using sensor data. Neural Comput Appl 2021;33(6):2015–33. http://dx.doi.org/10.1007/s00521-020-05097-x.
[58] Sivakumar M, Belle AB, Shan J, Odu O, Yuan M. Design of the safety case of the reinforcement learning-enabled component of a quanser autonomous vehicle. In: 2024 IEEE 32nd international requirements engineering conference workshops. 2024, p. 57–67. http://dx.doi.org/10.1109/REW61692.2024.00012.
[59] Henriksson J, Ursing S, Erdogan M, et al. Out-of-distribution detection as support for autonomous driving safety lifecycle. In: Ferrari A, Penzenstadler B, editors. Requirements engineering: foundation for software quality. Springer Nature Switzerland; 2023, p. 233–42.
[60] Lindvall M, Basili VR, Boehm BW, et al. Empirical findings in agile methods. In: Procs of the 2nd XP universe and 1st agile universe conference on extreme programming and agile methods. Springer-Verlag; 2002, p. 197–207.
[61] Hodge VJ. Sensors and data in mobile robotics for localisation. In: Wang J, editor. Encyclopedia of data science and machine learning. IGI Global; 2023, p. 2223–38. http://dx.doi.org/10.4018/978-1-7998-9220-5.
[62] Maqsood HM, Guerra EM, Wang X, Bondavalli A. Patterns for development of safety-critical systems with agile: Trace safety requirements and perform automated testing. In: Procs of the european conference on pattern languages of programs. ACM; 2020, p. 1–6. http://dx.doi.org/10.1145/3424771.3424800.
[63] Suresh H, Guttag J. A framework for understanding sources of harm throughout the machine learning life cycle. In: Equity and access in algorithms, mechanisms, and optimization. New York, NY, USA: Association for Computing Machinery; 2021, p. 1–9. http://dx.doi.org/10.1145/3465416.3483305.
[64] Steghöfer J-P, Knauss E, Horkoff J, Wohlrab R. Challenges of scaled agile for safety-critical systems. In: International conference on product-focused software process improvement. Springer; 2019, p. 350–66.
[65] Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G. Learning under concept drift: A review. IEEE Trans Knowl Data Eng 2019;31(12):2346–63. http://dx.doi.org/10.1109/TKDE.2018.2876857.
[66] Polyzotis N, Zaharia M. What can data-centric AI learn from data and ML engineering? 2021, http://dx.doi.org/10.48550/ARXIV.2112.06439, arXiv Pre-Print.
[67] Hodge V, Hawkins R, Hilder J, Habli I. Analysing ultra-wide band positioning for geofencing in a safety assurance context. 2022, arXiv preprint arXiv:2203.05830.
[68] Kniberg H, Skarin M. Kanban and scrum - making the most of both. In: Enterprise Software Development Series, C4Media Incorporated; 2010, URL https://books.google.co.uk/books?id=XBHRRwAACAAJ.