CLASS LIFECYCLE TECHNOLOGY SURVEY

Jonathan Rowanhill
Dependable Computing LLC

Dependable Computing Technical Report TR-2014-3

12/1/2014

Dependable Computing LLC
2120 North Pantops Drive, Charlottesville, VA 22911-8648
www.dependablecomputing.com

CLASS Lifecycle Technology Survey

Abstract

In this paper, we review the state of the art in lifecycle technology and use the results to analyze early CLASS prototypes. The result of the analysis led to substantial modifications to CLASS. An engineered system can be modeled with a lifecycle describing the temporal phases of its existence. Engineering systems with safety properties must consider safety of the system in each and every phase. Thus, a lifecycle model of a system is an excellent representation through which to prescribe safety processes and concerns. CLASS is designed to "enhance system safety in which the safety case becomes the focus of safety throughout the system lifecycle". CLASS defines lifecycles for systems with strict safety requirements. The central role of the safety case supports the analytical rigor required to both develop a safe system and reasonably assure safety. With safety cases as the focus, CLASS lifecycles highlight the relationship between the system's safety, safety arguments, and supporting evidence. These arguments and evidence are:

- Documents: inherently language-based communication between people and tools.
- Rigorous: structured and well-defined in order to reduce ambiguity and maintain precision. About systems: made to support systems.
- Safety focused: focused on system safety.
- Managed: living documents that must be accessed and modified over time.

There are lifecycle models defined separately for each of these attributes. These models have been used to enhance CLASS.

1 Introduction

In this paper, we review the state of the art in lifecycle technology. An engineered system can be modeled with a *lifecycle* describing the temporal phases of its existence. Engineering systems with safety properties must consider safety of the system in each and every phase. Thus, a lifecycle model of a system is an excellent representation through which to prescribe safety processes and concerns.

The *Comprehensive Lifecyce for Assuring System Safety* (CLASS) is designed to "enhance system safety in which the safety case becomes the focus of safety throughout the system lifecycle". CLASS defines lifecycles for systems with strict safety requirements. The central role of the safety case supports the analytical rigor required to both develop a safe system and reasonably assure safety.

With safety cases as the focus, CLASS lifecycles highlight the relationship between the system's safety, safety arguments, and supporting evidence. These arguments and evidence are:

- **Documents**: inherently language-based communication between people and tools.
- **Rigorous**: structured and well-defined in order to reduce ambiguity and maintain precision.
- **About systems**: made to support systems.
- **Safety focused**: focused on system safety.
- **Managed**: living documents that must be accessed and modified over time.

Interestingly, there are lifecycle models defined separately for each of these attributes. These might be of value to CLASS, as they define processes and tools that could potentially enhance the CLASS model.

1.1 Lifecycle Domains

In order to determine the potential value of existing lifecycle technology for CLASS, we examine the following lifecycle model domains:

- **Document creation lifecycles**.
- **Systems development lifecycles**.
- **Transformational lifecycles**.
- **Document and information management lifecycles**.

We will use the information gained to compare these models with CLASS. From this analysis, we will consider updates to the CLASS lifecycle models, and consider where we should focus our attention in further refinement of CLASS lifecycle processes and tools.

1.2 Organization

Section 2 discusses the various models of these lifecycles in detail. Section 3 discusses the current CLASS lifecycles and compares and contrasts them with those of the previous section. General conclusions are raised in Section 4.

2 Life Cycle Models

2.1 Lifecycle Concept

A system lifecycle is a temporal model of its states of existence. For many engineered systems this begins with the mandate for the system and continues through to its decommissioning.

Almost universally, a system lifecycle model consists of *phases*, a discrete set of states describing the system over time. Sometimes, lifecycles include *events* and *operations*, representing transitions between phases. *Events* are typically environmental occurrences, while *operations* are actions taken to deliberately manipulate the system's current phase.

As we consider various lifecycle models, we will use this terminology as well as a standard graphical representation of phases and events/operations. Many lifecycles we look at will describe phases in the form of the operations that are conducted in that phase, but will not explicitly describe the operations themselves. In such cases we will point this out for disambiguation.

2.2 Document Creation Lifecycles

Safety cases are documents or collections of document arguing for the safety of a system. We will consider the lifecycle of documents, and then consider the lifecycles of various types of documents relevant to a safety case.

2.2.1 Documentation Development Lifecycle

Software documentation development is traditionally defined by the Documentation Development Lifecycle. One model of this lifecycle [1] [2] [3] consists of:

1. *Prepare*: Establish purpose, assess the audience, determine scope, select medium
2. *Research*: conduct Subject Matter Expert interviews and perform background research
3. *Organize*: Analyze and categorize obtained information. Outline.

4. *Write*: Include a first draft with conclusion/summary
5. *Review*: Proofreading and peer reviews, iterative revisions.
6. *Packaging/Publishing*: Finalizing the document for availability
7. *Maintenance*: Updates are done as the documentation changes

This appears to be the basic creative process for document writing. Its value lies in that its phases might be considered in the creation of evidence and safety case documents. Yet, safety cases are rigorous and well structured, and not general documents.

2.2.2 Proof Lifecycle

A Safety Case is not a proof of safety, nor is it likely to be one. However, a safety case does contain rigorous arguments and might cite proofs in evidence, and therefore an analysis of proof lifecycle might be valuable. The creators of the Larch theorem prover describe the lifecycle of a proof [4] as follows:

1. *Design*
   (a) formalize objects being reasoned about
   (b) formalize a conjecture to be proved
   (c) outline a structure for the proof
2. *Code the proof into the theorem language*
3. *Proof Debugging*
   (a) Proof Success
   (b) Proof failure, iteration until success

A paper discussing the role of run-time generated proofs in proving that collaboration protocols meets compliance regulations, talks about a 'proof lifecycle' for the interactive generation of proof of compliance with protocols while monitoring a system [5]:

1. *Design Phase*
   (a) Formalize proof structure
   (b) Formalize compliance path (a document of compliance info)
2. *Injection/Enhancement Phase*
   (a) injected calls to verifier routines that will do verification and produce evidence at run time (Monitoring)
3. *Run-time Phase*
   (a) verification (tools check that previous evidence has been collected and is valid)
   (b) run the protocol step
   (c) evidence production from the result of the protocol step
   (d) or halt and fail proof if verification failed

The process either results in a failed protocol detection or a proof that the protocol was followed properly. This has value for legal compliance regulation.

2.2.3 Safety Case Lifecycles

Above we saw lifecycles for documents and formal arguments, both of which are aspects of a safety case. In addition, lifecycles specific to safety cases have been developed.

Kelly argues for an evolving safety case that iterates and changes in parallel with the development of the system for which it argues [6]. This evolving safety case is planned, designed, implemented, and tested in parallel with system development. Both the safety case and the system are modified in response to the results of the development phases for themselves and one another.

In this model, there are three phases of safety case in a lifecycle:

1. *Preliminary Safety Case*: The safety case after "definition and review of the system requirements specification". This occurs after safety plan production, required safety properties identification, preliminary hazard analysis, risk estimation, and integrity level requirements analysis.
2. *Interim Safety Case*: The safety case after system design and preliminary validation.
3. *Operational Safety Case*: The safety case just prior to in-service use for an operational system.

Prior to this work, Kelly and McDermid argued for safety case maintenance in a live system [7]. In this process, the safety case is modified in two phases:

1. *Damage Phase*: The damage caused by changing the system/or safety case is assessed.
2. *Recovery Phase*: This damage analysis and any recovery actions determined, are input to change processes in the safety case to recover the safety argument.

These phases can occur iteratively. They may occur whenever a system undergoes maintenance, upgrades, retrofits, *etc.*

Combined, these two models define an iterative refinement method for a safety case, carrying it through the design and into the operations phases of a system lifecycle.

Bishop and Bloomfield provide a similar safety case lifecycle model that extends from the build phase of a system into its operational phase [8]. It includes the following lifecycle phases:

1. *Functional Identification*: Safety functions and top-level safety attributes identified
2. *Architecture Identification*: System architecture and safety case outline defined
3. *Preliminary Design Assessment*: Cost and risks, and long-term support, determined for a set of design options.
4. *Progressive Elaboration*: Through iteration and increasing specificity, the system design and safety case are refined in parallel
5. *Integration*: A final safety case is produced
6. *Support Plans*: A final long term support plan is drafted
7. *Approval*: Stakeholders sign off on the design, plans, and safety case
8. *Monitoring and Audits*: Over operations and maintenance of the system, it is monitored over areas of concern and support processes. Evidence is gathered from the system to support assumptions made in the safety case.
9. *System Updates and Corrections*: In response to monitoring, the safety case is iteratively corrected.

This design begins in a waterfall manner, becomes iterative refinement and correction during design, and then proceeds into a maintenance loop where the safety case remains a living document, responding along with the system implementation and design to observations, monitoring, and auditing with corrections and changes.

Greenwell, Strunk, and Knight describe in detail a process for using a safety case to guide failure analysis of a live system [9]. Rather than responding to system changes after the fact, this process applies system failure analysis to the safety case prior to system updates. The phases of the enhanced safety case lifecycle are:

1. *Failure Analysis*: Failure evidence and the orignal safety case are the input to the failure analysis. Given the failure evidence and the arguments and evidence of the safety case, defects in the safety case can be identified by contradiction in the failure evidence. This allows improvement of the safety case and can help narrow down underlying causes of failure. The result of this phase is the production of lessons learned and an updated safety case.
2. *System and Process Revision*: The lessons learned and revised safety case are input to a process and system review. During this phase, the system and its processes are updated based on the lessons learned and new safety case.
3. *Operation*: In this phase, the system returns to operation.
4. *Mishap*: In this phase, a mishap in the system has occurred. Failure evidence is collected, and input back into the Failure Analysis phase.

2.2.4 Discussion

Common themes in document creation lifecycles are evident in key phases of planning, iteration, and acceptance. This can be followed by further iteration, refinement, and ammendment during the maintenance phase of a living document. It is the existence of these later lifecycle phases that distinguishes a 'living' from a static document artifact.

2.3 Systems Development Lifecycles

We have seen how safety case lifecycles have evolved to become part of an overall system development lifecycle. We now consider the basis for system development lifecycles, and see how they have been extended to support issues such as security and safety.

2.3.1 Standard Systems Development Lifecycle (SDLC)

The standard systems development lifecycle consists of the following phases [10]:

1. *Planning*: Establish a high level view of the intended project.
2. *Analysis*: Refine project goals from requirements.
3. *Design*: Describe desired features and operations in detail.
4. *Develop*: Building of the implemented system.
5. *Integration and Testing*: Testing of the system, iterating for issues and errors.
6. *Finalization and Maintenance*: System is placed into production.

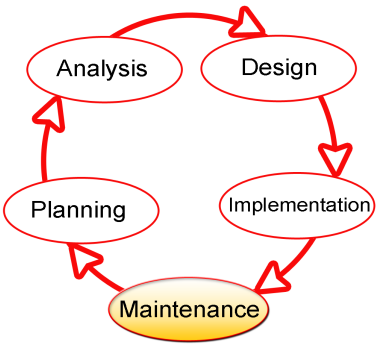

Figure 1: System Development Lifecycle

When implemented linearly, this is the waterfall model [10]. The fountain model allows phases to occur concurrently although there are dependencies between them [10]. The spiral model assumes iteration over these dependent phases in order to refine specification into implementation, where each iteration of the spiral is a short waterfall activity [10].

Extensions to this model for legacy systems generally refine the maintenance phase to consist of the following phases [11]:
Finalization and Maintenance

1. *Evolution*: An iterative phase in which the system continues to evolve to better meet the needs of its users through evolution of its services and functions.
2. *Servicing*: A phase in which the system tends not to further evolve but requires servicing in order to continue to function.
3. *Phaseout*: A system is no longer to be upgraded, with only necessary emergency changes made or effects mitigated. Some non-functional loss may occur but is accepted.
4. *Closedown*: A system is terminated and removed from the system.

These extended phases better represent the phases of large software system use in larger organizations, but also can model the role between, for example, a user and an owned vehicle or operating system.

2.3.2 Enterprise Architecture Lifecycle

The concept of *Enterprise Architecture* is a form of data-flow and architecture-centric design for systems-of-systems within a large enterprise. Through use of Enterprise Architecture techniques, large scale business processes can be implemented efficiently, soundly, and robustly within large companies with evolving workflow requirements.

Enterprise Architecture involves the Enterprise Lifecycle (ELC), described in Figure 2, the phases of which are relatively similar to those of a traditional SDLC.

This process is largely the same as a design and iteration cycle as seen in other creative lifecycle processes.

2.3.3 Trustworthy Computing Security Development Lifecycle

Lipner at Microsoft Corporation proposed a trustworthy computing security development lifecycle [12]. This process consisted of the following phases in linear order:

Figure 2: Enterprise Architecture Lifecycle

1. *Requirements*: During system requirements gathering, a security advisor is assigned. Security requirements are understood and milestones established.
2. *Design*: Threat models are produced. Security architecture is documented.
3. *Implementation*: Coding and test standards are followed for security. Test plans include significant security tests. Code analysis tools are run to detect security issues.
4. *Verification (Beta)*: Threat models are reviewed along with code. Attack testing is carried out against the beta candidate. New determined threats are evaluated.
5. *Release*: The threat model is reviewed. Unfixed bugs are reviewed for security implications. Penetration testing is completed. Documentation is archived.
6. *RTM*: Security team signoff on the application.
7. *Response*: Tools/processes are reevaluated. Postmortem analyses are completed.

Essentially, this lifecycle follows the waterfall model, except that Threat Models and Security Testing and Analysis, as documents and tests, become central to the development lifecycle.

2.3.4 Augmenting the System Development Lifecycle

Researchers at Jet Propulsion Laboratory proposed using a Software Security Checklist (SSC) as a central document to follow along the lifecycle of software system development.

In this model, a Security Risk Analysis is conducted during the design phase. This is used to generated the Software Security Checklist. An example security checklist provided in their work [13] augments a typical software lifecycle as follows:

1. **ALL PHASES**: Security audit review, formal review, or security walkthrough. Introduce security metrics.

2. *Analysis*: Determine stakeholders, and elicit and specify associated security requirements. Determine context and potential usage of software product along with operating environment and specify requisite security requirements. Analyze functional requirements for security inconsistencies.

3. *Before Design*: Educate developers and reviewers in the vulnerabilities of programming languages and environments before the design phase.

4. *Design*: Analyze the security of proposed middleware. Check for architectural vulnerabilities. Specify how data flows through the architecture. Identify trust boundaries.

5. *Implementation*: Check for implementation security errors such as buffer overflows and race conditions. Do not allow programmer back doors. Do not store secrets within code or documentation. Review and reduce software complexity. Perform security unit testing.

6. *Integration and Testing* : Review API calls to security modules and interfaces. Make sure secure connections are secure and connect to the correct places. Investigate security between systems and databases. Analyze configuration management for vulnerability. Perform security integration testing.

7. *Finalization and Maintenance*: Rate the application's security risk to the organization. Assure the system maintains highest security available and only reduces as needed following proper protocols.

As can be seen, the checklist consists of many items to be conducted in various phases of the lifecycle. In this sense, the SSC is an aspect of the development process for the software development lifecyle (SDLC).

Jones and Rastogi [14] propose a similar approach, with various testing techniques, policies, and analysis present during the lifecycle. In addition, they specifically call out the need for:

- **Patch management**: Consistent updates of software as more secure updates become available.
- **Monitoring**: Consistent checks on the behavior in the running system.
- **Logging**: Continuous recording of all activity of the system.

The addition of these continuous operations into the maintenance phase of a software system obviously improves detection of safety violations and security improvement over time. These general principles are easily applicable to non-software systems.

2.3.5 IEC 61508 Safety Lifecycle

IEC 61508, "Functional safety of electrical, electronic, and programmable electronic safety-related systems" is a standard to allow "safe exploitation of programmable electronic systems used for safety applications" [15].

One of the strategies to achieve functional safety through IEC 61508 is to place "technical requirements" on each phase of a safety lifecycle [15]. IEC 61508 contains a recommended safety lifecycle which we present in Figure 3.

The phases of this lifecycle are:

1. *Concept*: The ideation of a system.

2. *Overall Scope Definition*: An overall description of the system's impact and definition of its safety.

3. *Hazard and Risk Analysis*: Specific hazard and risk analysis carried out on the system ideation over its proposed scope.

4. *Overall Safety Requirements*: A drawing of safety requirements required to mitigate hazards and accomodate risk based on the system concepts.

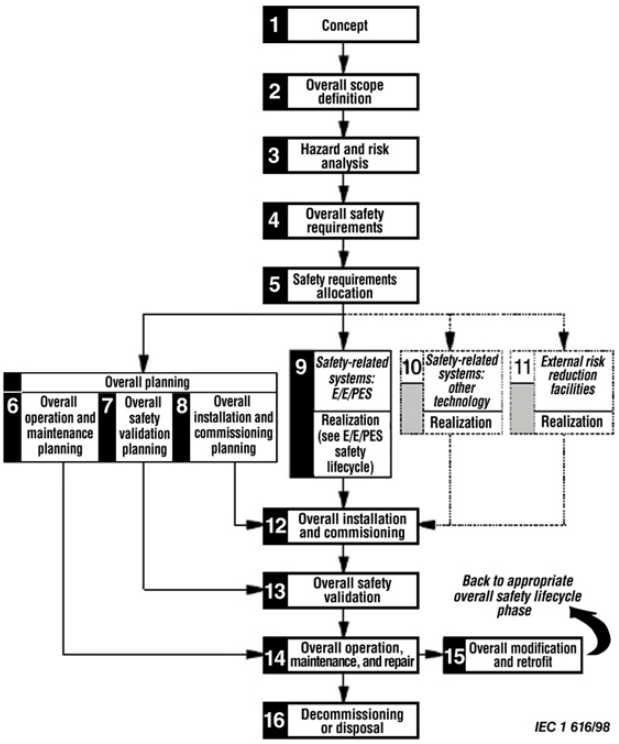

Figure 3: IEC 61508 Lifecycle

5. *Safety Requirements Allocation*: Allocation of safety requirements to hardware and software.
6. *Overall Operation and Maintenance Planning*: Planning for how the system will be developed.
7. *Overall Safety Validation Planning*: Planning for how the system safety requirements will be validated.
8. *Overall installation and commissioning planning*: Planning for how the system will be deployed.
9. *Safety Related Systems E/E/PES Lifecycle*: There is a separate sub-lifecycle for a component that is an electronic (E), electrical (E), or Programmable Electronic System (PES).
10. *Safety-related systems: other technology*: Planning for systems that are not E/E/PES.
11. *External risk reduction facilities*: Planning for the incorporation of risk reduction techniques that are external to the subject system.
12. *Overall installation and commissioning*: The system as a whole is installed and hand-off to operations occurs in commissioning.
13. *Overall safety validation*: The installed and commissioned system is validated for its safety, as defined in earlier planning.
14. *Overall operation, maintenance, and repair*: Operations groups operate, maintain, and repair the system.
15. *Overall modification and retrofit*: Modifications teams access the system to modify and change it to repurpose for a different use.
16. *Decommissioning or disposal*: The system is safely removed from operation and remains in a safe decommissioned state.

Within phase 9, *Safety Related Systems E/E/PES Lifecycle*, up to two subcomponent lifecycles per system component can exist. One exists for the electric, electronic, or programmable electronic component. The second consists of any software of that component. The safety lifecycle for a E/E/PES subcomponent is shown in Figure 4. The software component lifecycle is essentially the same, and occurs in parallel but separate from its electronics.

The phases of the component lifecycle are:

1. *Component safety requirement specification*: Drawing the safety requirements for this specific component.
2. *Safety functions requirement specification*: Requirements are drawn for the specific functions implementing/ supporting safety performed by this component for the system.
3. *Safety integrity requirement specification*: Requirements about limits on level of risk associated with each safety function requirement.
4. *Component safety validation planning*: Planning for how the safety of the given component, its functions and their integrity, will be carried out.
5. *Component design and development*: Design and implementation phases of the SDLC.
6. *Component integration*: Integration into the larger system.
7. *Component operation and maintenance procedures*: Development of the component operation and maintenance procedures.
8. *Component safety validation*: The validation, previously planned for, is performed.

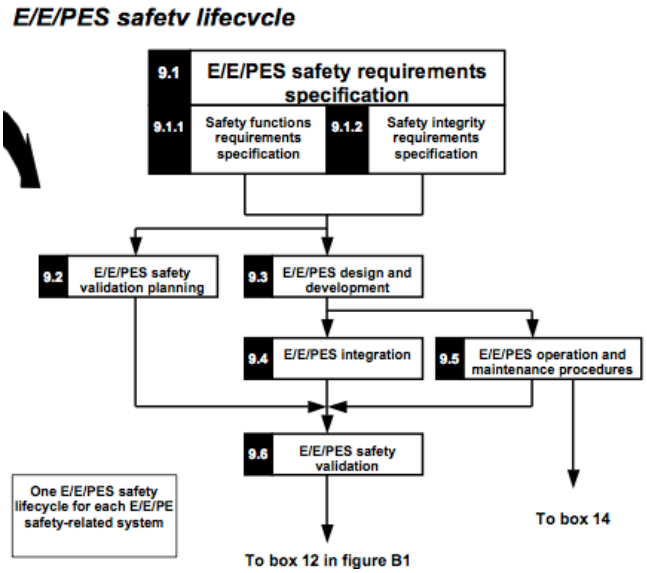

Figure 4: IEC 61508 E/E/PES Subcomponent Lifecycle

2.4 Transformational Lifecycles

Most system lifecycles take a refinement view of development. A system is designed and built, operated and then retired. The system is refined over time.

However, many real world systems cannot be refined, but must be subject to large scale changes. They are too big, too expensive, or too reliable to be easily replaced, but they are too far from a new required to simply be refined.

2.4.1 Safety Case Lifecycle for Neural Networks and Unpredictable Data-driven Systems

Kurd and Kelly apply the idea of a safety case lifecycle to the neural networks [16]. Determining the safety of a neural network can, roughly, be categorized as determining the safety of a largely data-driven system where prescriptive and active bounds are not easily enforced.

Kurd and Kelly modify the traditional safety case development integrated with system building to after the network training phase of the neural net. After this time, white-box analysis of symbolic data and black-box testing of neural net behavior can occur. Again, these lessons can be generalized to data driven applications where prescriptive bounds cannot be easily predicted or enforced.

The resulting phases of a safety lifecycle are:

1. *Initial Hazard List*: A preliminary hazard analysis is input in this phase.
2. *Functional Hazard Analysis*: A functional hazard analysis is created or refined in this phase (see step 3.)
3. *Iterative Neural Network Training*: The neural network is trained on data in this phase. In this phase, the symbolic form of learning and data can change, and this may drive needed changes to the hazard analysis. Therefore this phase can iterate back to phase 2.
4. *Refined Symbolic Information*: In this phase the neural network is allowed to continue to learn, but data schemas and symbolic forms cannot change, only content. This bounds changes to parameterizations within the existing finite hazard analysis.

5. *Final Safety Assessment*: Black box testing can occur during this phase to provide statistical evidence for conformance with the safety case.

2.4.2 Enterprise Integration Lifecycle

Enterprise Application Integration (EAI) is a form of enterprise architecture in which the interconnection of systems into systems-of-systems occurs. It is often a data-flow centric approach but can have dramatic influence on the operation of software systems and subsystems. The goal of EAI is to create more flexible and robust systems through a data flow architectural focus. Obviously, this can have a significant impact on their safety and other operational properties.

Traditional software design lifecycles are seen as an impediment to the data-flow centric approach of EAI [17], in which maximimal component and subsystem decoupling is maintained, and systems of systems are often loosely federated and separately owned.

2.5 Document and Information Management Lifecycles

CLASS is centered on safety case documents. These documents consist of arguments and evidence that can be thought of as information assembled over time. Therefore, we consider the state-of-the-art in document management to determine its impact on CLASS.

2.5.1 Collections of Canada

A detailed *information lifecycle*, presented in Figure 5, is given by Library and Archives Canada [18].

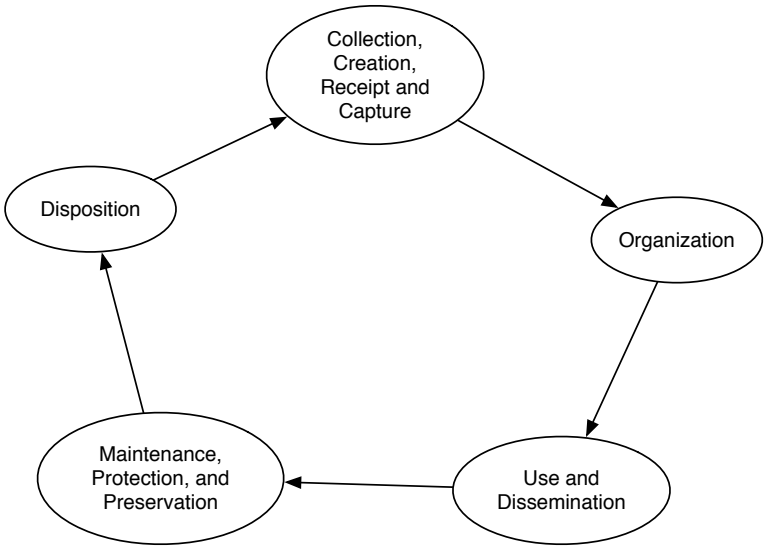

Figure 5: An Information Management Lifecycle

The lifecycle consists of the following phases (ignoring continuous improvement phases):

1. *Collection, Creation, Receipt & Capture*: Information is being originated or received. Marking reception and capturing proper input are important in this phase.
2. *Organization*: Information is organized and assigned policies and procedures describing the business rules of an organization.

3. *Use and Dissemination*: Information is being actively used and shared.
4. *Maintenance, Protection and Preservation*: Information is kept current and stored securely.
5. *Disposition*: Non-active information is archived (long term storage), alienated (given away to another system to manage), or destroyed.

Note from Figure 5 that this lifecycle is cyclic. In essence, information that is archived is dormant from the policies and procedures of the organization. When it is fetched from archive, it is 'recaptured' and then current business logic and policies are applied again. The formality of transitions in this model appears suspect. It is likely that the "Use" phase and the "Maintenance" phase operate concurrently. If not, this model is oddly linear in the order between use and maintenance.

2.5.2 AIIM Content Lifecycle

The concurrency of information phases is expressed more directly by the information management model of the Association for Information and Image Management [19]. Their model, depicted in Figure 6, focuses on the phase of individual copies of the information (information is replicated during management).

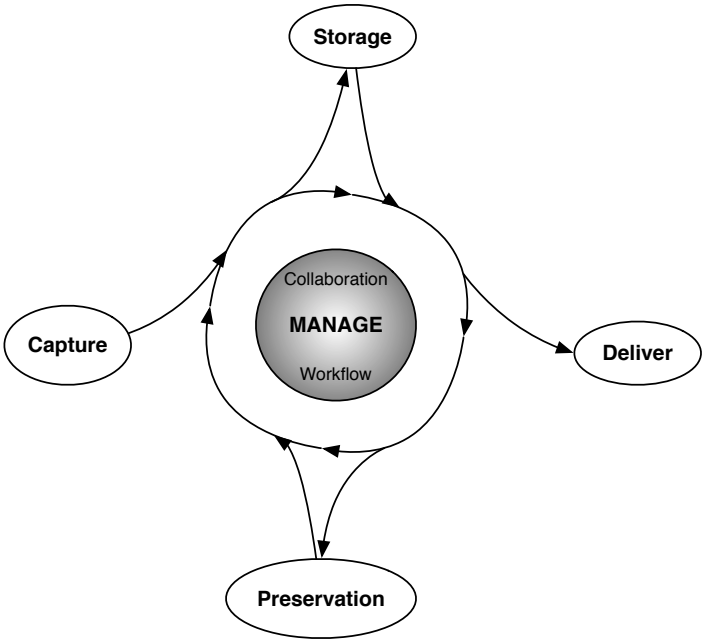

Figure 6: AIIM Lifecycle Management

In this model the core phases are:

1. *Capture*: Information is created or brought into management.
2. *Manage/Process*: Information passes through business logic in the system.
3. *Storage/Retrieval*: Information is stored for later location and use.
4. *Preservation*: Information is archived.

5. *Delivery*: Information is delivered to users of content.

The capture phase is an initial state. Delivery of information to the user is a final state. In this model, changing information and resubmitting it to a system would be an act of capturing new information.

The management phase is cyclic. Information flows through the business logic of management whenever it is needed. Information flows around the business logic and through user collaboration, passing in and out of storage and preservation phases.

Information in the storage/retrieval phase exists in data records that business logic can access easily. Information in the preservation state is often archived in a special mechanism to maintain the originality of a document, such as an original X-ray image or legal document.

Multiple copies of an information record can exist and copies can exist in different phases of the lifecycle. Information in the management phase is participating in business logic.

2.5.3 Dartmouth Library System defined Life Cycle for Documents

The Dartmouth library system defines the phases for a document lifecycle [20] as follows:

1. *Create*: making of an analog or digital document.
2. *Capture*: conversions to a standard format.
3. *Index*: cataloging process or creating metadata.
4. *Manage*: management and storage for rapid accessibility.
5. *Access*: searching process to find documents.
6. *Retrieve*: viewing documents from search results.
7. *Administer*: managing users, resources, content types, and structures.
8. *Repurpose*: reusing and re-versioning document for new products and/or cost savings.
9. *Share and Collaborate*: sharing, collaborating, and modifying in a work group.
10. *Distribute*: exporting and distributing documents in a secure manner.
11. *Retain*: hold low usage or inactive documents for their predetermined retention periods.
12. *Dispose* : securely destroy documents passed retention periods.
13. *Preserve* : ensure long-term accessibility for documents with enduring value.

Note that some might model these phases as operations. A way to think about this is that multiple copies of a document might exist simultaneously, and each is in one of these phases.

Again, transitions for this lifecycle are not explicitly described. This lifecycle captures the problems of long term document storage within a library system. Repurposing, for example, appears to have meaning not found in the more general information lifecycles described previously.

On the other hand, many of these phases can be mapped to the storage and disposition, delivery, and capture phases of the more general model.

Table 1: Phase Mapping of Information Management Lifecycles

General Phase	Model Phases		
General Phase	AIIM	Canada	Dartmouth
Capture	Capture Creation	Capture Creation Receipt Collection	Capture Create Repurpose
Map	Manage/Process	Organization	Index
Manage	Manage/Process	Maintenance Protection	Manage Administer Collaboration
Storage	Storage Retrieval	Preservation	Manage Retrieval Access
Delivery	Delivery	Use Dissemination	Distribute Share
Archive	Preservation	Disposition	Preserve Retain
Diposition		Disposition	Dispose

2.5.4 Generalization

Table 1 generalizes the lifecycle phases of the three models, and provides a mapping to the information management models presented. What we see are consistent themes but inconsistent representation of explicit phases across the three models. No model directly subsumes the others. Of the models, the AIIM model is the most generalized, while the Dartmouth model makes the least distinction between phase and operation but is also the most specific.

2.6 Summary

In this section we have considered the lifecycle models for creating, maintaining, and managing documents, proofs, safety cases, systems, and systems requiring engineered safety and security.

2.7 Commonalities

We note that many of these lifecycles share common themes, including:

- **Planning**: Phases for plan and design of the subject. This can involve choosing a software development methodology or planning a document storage metadata model.
- **Analysis**: Phases for analyzing other things that have just been produced or input to consideration of the subject. This can include the production of a finite hazard analysis or the indexing and categorization of a document.
- **Building and instantiating**: Phases for generating the subject's key components from inputs and pre-existing products. These phases tend to be iterative in modern lifecycle models. Testing can occur in this phase and consist of proof-reading a document draft, reviewing a safety case argument with a subject matter expert, or running unit tests on code.
- **Maintaining**: Phases for storing, operating, fixing, and monitoring the subject. This phase is, by nature, iterative. This phase can include monitoring a live software system or auditing a safety case document.

- **Disposing**: Phases for retiring the subject, such as archiving software binaries or evidence in a safety case, destroying a document, or decommissioning an industrial plant.

3 CLASS Lifecycle

The Comprehensive Lifecycle for Assuring System Safety (CLASS) facilitates the creation and use of safe systems.

The CLASS process applies two lifecycles:

1. **MetaCLASS Instance Lifecycle Generation Process**: A process for creating and maintaining a CLASS process for a system. For brevity, this will be referred to in the remainder of this paper as the 'metaCLASS'.
2. **InstanceCLASS Process**: A process for creating and maintaining a safe system in tandem with a safety case. This will be referred to in the remainder of this paper as the 'instanceCLASS'.

In other words, metaCLASS is used to instantiate and maintain a useful instanceCLASS for a particular system. An instanceCLASS is used to create and maintain that particular system.

3.1 Comparisons

CLASS works with similar principles to the IEC 61508 safety lifecycle presented in Section 2.3.5. It is a comprehensive safety lifecycle, differing in the central role played by a rigorous safety case.

CLASS is also similar to the security lifecycle models presented in Section 2.3.3 where the aspect of safety is analyzed through specific techniques across all phases. Also similarly, in each phase particular artifacts are created, tests are run, and analyses are conducted.

CLASS stands out from these mechanisms, however, in that it strictly gates certification as a top-level, first-class phase of the overall lifecycle.

As seen in Section 2.2.2, CLASS safety cases can be iterated upon for acceptance. At the lifecycle level, creating proofs and safety cases can appear similar. However, because a safety case argument is unlikely to be proof (except for the smallest aspects or sub-arguments), a rigorous dialectic process is part of the CLASS workflow. Yet creation and maintenance of a safety case shares properties of formal acceptance with a proof that document creation does not generally include, including a rigorous acceptance process.

3.2 Deficits

CLASS might benefit from the explicit introduction of information management, where it is otherwise preoccupied with system creation processes.

We noted at the end of Section 2 that, in general, lifecycles tend to model a terminal *disposal* phase. Early CLASS prototypes lacked such a phase. This omission was important for at least two reasons:

1. When a system is decommissioned, its safety case is not necessarily unimportant. There are many possible roles for a safety case in CLASS during and after decommissioning, including:
   (a) the safe decommissioning of the system,
   (b) use in future historical forensics of past incidents, and
   (c) use in analysis for future safety cases.
2. Safety factors and hazards often come into play during decommissioning that are not immediately effective in the creation or maintenance phases. For example, the decommissioning of a nuclear plant is a critical safety phase.

As a *disposition* phase appears almost universally, such a phase was added to CLASS.

We should also consider the important role of document management lifecycles in the information management community. Given that CLASS requires safety case and evidence documents to play a central role, a lack of explicit modeling for information management might be an omission of the current CLASS process/lifecycle model.

The operation phase of complex system lifecycles often are enhanced with sub-phases describing the hand-off timeline of the system between various operational teams. As we move forward with defining maintenance processes available through metaCLASS, we should take such standard maintenance lifecycle models into account. As an example, consider how many large software systems are sub-phased during operation:

1. *Release*: During this sub-phase of maintenance, a system is new. Callibration of the system to the environment and late defect repair are common. Interaction of implementors with a launch team is required.
2. *Maturation*: Calibration and defect repair decreases. Performance tuning for increasing scale, load, and dependability takes precedence.
3. *Stasis*: Unaltered operations occur in this phase. Routine repair occurs in this phase. New features and integration cause a reentry to early phases.
4. *Phaseout*: When a system is being phased out of operations within a supersystem, as described in Section 2.3.1 it often is forced to take on risk that was not acceptable in its release and maturation phases. In this case, explicit attention to the safety case, in order to maintain reasonable risk, or accomodate increased risk, must be taken.

These phases often operate in a spiral process as features and enhancements are released and changes to the environment take place.

In the above example, the phases explicitly model enterprise software. Part of metaCLASS will be providing other operations and maintenance sub-processes for a wide variety of safety-critical system types.

3.3 CLASS Lifecycle

Based on this analysis of this work, CLASS was extended to include the following:

1. **Potential sub-phases for the CLASS Instance Operation Phase**: The sub-phases *Launch*, *Maturation*, *Stasis*, and *Phaseout*, as described previously, will be considered as a standard sub-lifecycle model for the maintenance phase of CLASS instances. Where particular system domains require a different model, it will be identified, and the reasons for the discrepancy recorded within the metaCLASS repository.
2. **Linking of Lifecycle Phases**: Currently, the metaCLASS and instanceCLASS lifecycles are tied together at the process level, but have not been modelled as linked at the lifecycle level. Furthermore, a document lifecycle will be added to the model. All three of these lifecycles are linked together. Going forward, we will explicitly model links between CLASS lifecycles.
3. **A Disposition phase for both Meta CLASS and Instance CLASS Lifecycles**: When a system is decommissioned, an explicit workflow policy should determine what happens to its safety case in its instanceCLASS instance. For example, is the safety case archived? Is a retrospective report generated? Likewise, the metaCLASS should have a disposition phase linked to the disposition phase of instanceCLASS. In this phase, metaCLASS can initiate activities closing out this instanceCLASS, such as forensic activities for improvement of the CLASS Resource Repository.

3.4 Additional Lifecycles

Finally, we note that this paper has focused on the metaCLASS instancing service and the generated instanceCLASS processes. We have not yet defined a lifecycle model for the CLASS Resource Repository system itself. Future CLASS prototypes would benefit from an explicit metaCLASS service lifecycle.

4 Conclusions

4.1 Summary

We have analyzed lifecycles from domains sharing similarities with CLASS and with safety case documents. These included

- Document creation.
- Safety case creation.
- Proof creation.
- Document and information management.
- System lifecycles.
- Security and safety system lifecycles.

From these an enhanced lifecycle model for CLASS was developed that dealt with omissions in early CLASS prototypes. CLASS now includes:

- **Disposition**: A *Disposition* phase for both metaCLASS and instanceCLASS processes.
- **Linking**: An explicit model of correlation between certain phases of metaCLASS and instanceCLASS processes.
- **Operation Sub-process**: An example model of *Operation* sub-process, with defined sub-phases of the lifecycle.

References

[1] I. Suri, "Understanding document development life cycle," 2013. [Online]. Available: http://www.slideshare.net/ishasuri6/document-development-life-cycle

[2] H. Sehgal, "Document development life cycle (ddlc)."

[3] "Ddlc." [Online]. Available: http://en.wikipedia.org/wiki/DDLChttp://en.wikipedia.org/wiki/DDLChttp://en.wikipedia.org/wiki/DDLC

[4] S. J. Garland and J. V. Guttag, "An overview of lp, the larch prover," in *Rewriting Techniques and Applications*. Springer, 1989, pp. 137–151.

[5] A. Svirskas, C. Courbis, R. Molva, and J. Bedzinskas, "Compliance proofs for collaborative interactions using aspect-oriented approach," in *Services, 2007 IEEE Congress on*. IEEE, 2007, pp. 33–40.

[6] T. Kelly, "A systematic approach to safety case management."

[7] T. Kelly and J. McDermid, "A systematic approach to safety case maintenance," *Reliability Engineering and System Safety*, vol. 71, no. 3, pp. 271–284, 2001.

[8] P. Bishop and R. Bloomfield, "A methodology for safety case development," in *Industrial Perspectives of Safety-critical Systems*. Springer, 1998, pp. 194–203.

[9] W. S. Greenwell, E. A. Strunk, and J. C. Knight, "Failure analysis and the safety-case lifecycle," in *Human Error, Safety and Systems Development*. Springer, 2004, pp. 163–176.

[10] (2002, May) Quickstudy: System development life cycle. [Online]. Available: http://www.computerworld.com/s/article/71151/System_Development_Life_Cycle

[11] V. T. Rajlich and K. H. Bennett, "A staged model for the software life cycle," *Computer*, vol. 33, no. 7, pp. 66–71, 2000.

[12] S. Lipner, "The trustworthy computing security development lifecycle," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 2–13.

[13] D. P. Gilliam, T. L. Wolfe, J. S. Sherif, and M. Bishop, "Software security checklist for the software life cycle," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*. IEEE, 2003, pp. 243–248.

[14] R. L. Jones and A. Rastogi, "Secure coding: building security into the software development life cycle," *Information Systems Security*, vol. 13, no. 5, pp. 29–39, 2004.

[15] R. Bell, "Introduction to iec 61508," in *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*. Australian Computer Society, Inc., 2006, pp. 3–12.

[16] Z. Kurd and T. Kelly, "Safety lifecycle for developing safety critical artificial neural networks," in *Computer Safety, Reliability, and Security*. Springer, 2003, pp. 77–91.

[17] M. Themistocleous, Z. Irani, J. Kuljis, and P. E. Love, "Extending the information system lifecycle through enterprise application integration: a case study experience," in *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*. IEEE, 2004, pp. 8–pp.

[18] Library and A. Canada, "Records and information life cycle management." [Online]. Available: http://www.collectionscanada.gc.ca/007/002/007002-2012-e.html

[19] AIIM. What is information management? [Online]. Available: http://www.aiim.org/what-is-information-management

[20] www.dartmouth.edu, "The document lifecycle: Definitions, supporting technologies, and applications." [Online]. Available: http://www.dartmouth.edu/~library/recmgmt/forms/DocLifeCycle.pdf?mswitch-redir=classic

