Avionic Software Certification and Regulation

Kyle DeBates

University of Minnesota, Morris

April 20th, 2019

What are Avionics?

Introduction

The avionics industry currently lacks an explicit and transparent design methodology

Commercial guidelines direct the design process, however they do not determine ***how*** to design avionics

A clear need for generalized design methodology as systems complexity increases, which this research aims to provide

Outline

DO-178C Commercial Avionics Guidelines

The DO-178C is the third and newest revision of the industry guidelines for commercial aviation software approval [BKPK15]

Defines expected functionality and safety requirements as well as requirements to avoid common errors

Used to establish compliance of avionics components and full systems of components for commercial airline use

DO-178C Terminology

System and Safety Requirements Allocated to Software (SRATS) are the required goals for software design

This is reflected in the Design Assurance Levels (DALs) prioritization hierarchy

These assurance levels are heavily influenced by *Contributions to Failure Conditions* (CFCs)

Design Assurance Levels (DALs)

Five distinct Design Assurance Levels of DO-178C:

Level A	71 Obj	Catastrophic
Level B	69 Obj	Hazardous
Level C	62 Obj	Major
Level D	26 Obj	Minor
Level E	0 Obj	No Effect

[Rus11]

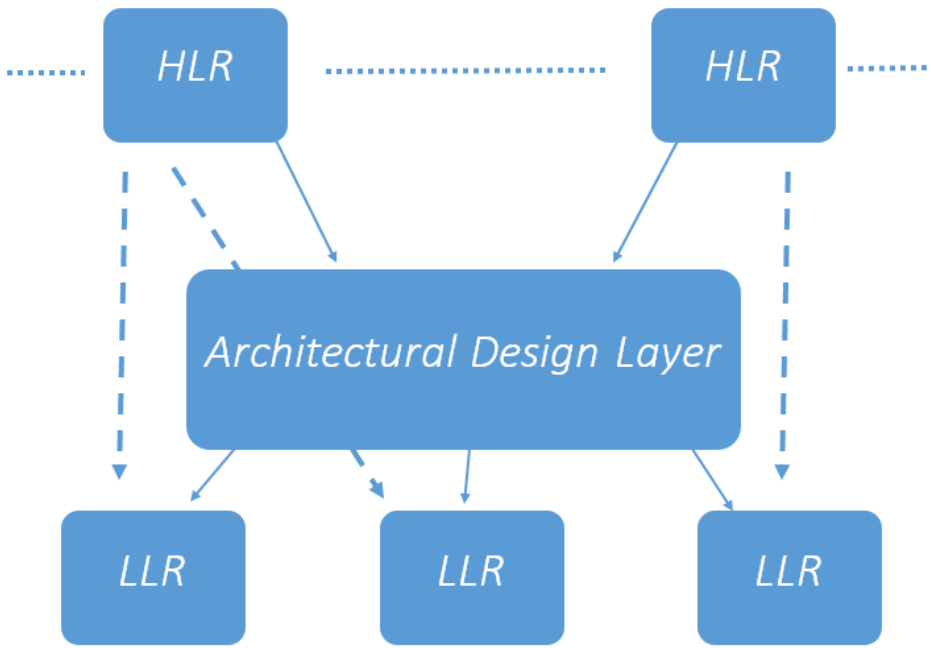
Controllable variable: Data values manipulated arbitrarily by software

Monitorable variable: Data values recorded in operational environment

Terminology for Proposed Software Development Methodology

System functionality categorized to emulate Design Assurance Levels:

- High Level Requirements (HLRs)
- Architectural Design Layer
- Low Level Requirements (LLRs)



DO-178C Vs. Software Development Methodology

Primary Differences:

- SRATS are the general requirements of any system to achieve desired functionality
- CFCs define possible issues precluding the failure of any system component
- HLRs are used to specify the *how* and *what* of the operational requirements of SRATS
- Architectural Layer is abstracted to assure necessary interconnectivity and further exhaustion of SRATS
- LLRs are the specific software classes and methods required to actualize the HLRs with the proper outlined architecture

DO-178C Vs. Software Development Methodology

Primary Differences:

- SRATS are the general requirements of any system to achieve desired functionality
- CFCs define possible issues precluding the failure of any system component
- HLRs are used to specify the *how* and *what* of the operational requirements of SRATS
- Architectural Layer is abstracted to assure necessary interconnectivity and further exhaustion of SRATS
- LLRs are the specific software classes and methods required to actualize the HLRs with the proper outlined architecture

DO-178C Vs. Software Development Methodology

Primary Differences:

- SRATS are the general requirements of any system to achieve desired functionality
- CFCs define possible issues precluding the failure of any system component
- HLRs are used to specify the *how* and *what* of the operational requirements of SRATS
- Architectural Layer is abstracted to assure necessary interconnectivity and further exhaustion of SRATS
- LLRs are the specific software classes and methods required to actualize the HLRs with the proper outlined architecture

DO-178C Vs. Software Development Methodology

Primary Differences:

- SRATS are the general requirements of any system to achieve desired functionality
- CFCs define possible issues precluding the failure of any system component
- HLRs are used to specify the *how* and *what* of the operational requirements of SRATS
- Architectural Layer is abstracted to assure necessary interconnectivity and further exhaustion of SRATS
- LLRs are the specific software classes and methods required to actualize the HLRs with the proper outlined architecture

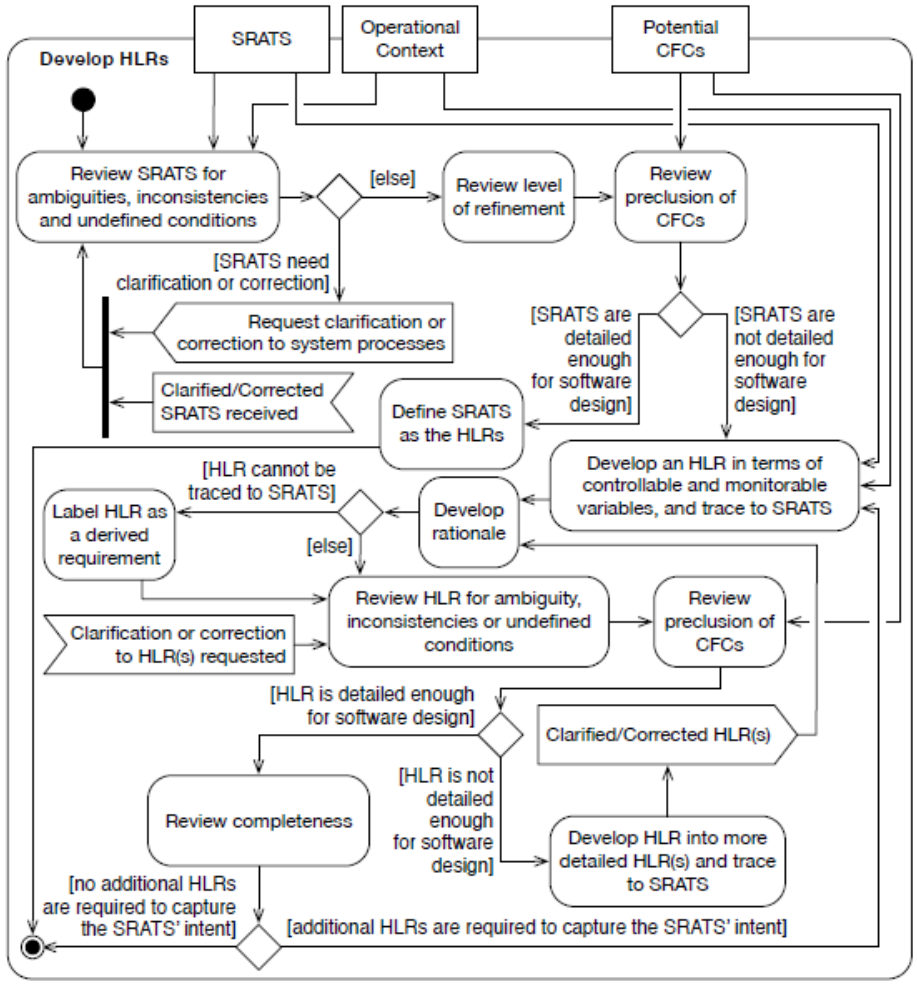
DO-178C Vs. Software Development Methodology

Primary Differences:

- SRATS are the general requirements of any system to achieve desired functionality
- CFCs define possible issues precluding the failure of any system component
- HLRs are used to specify the *how* and *what* of the operational requirements of SRATS
- Architectural Layer is abstracted to assure necessary interconnectivity and further exhaustion of SRATS
- LLRs are the specific software classes and methods required to actualize the HLRs with the proper outlined architecture

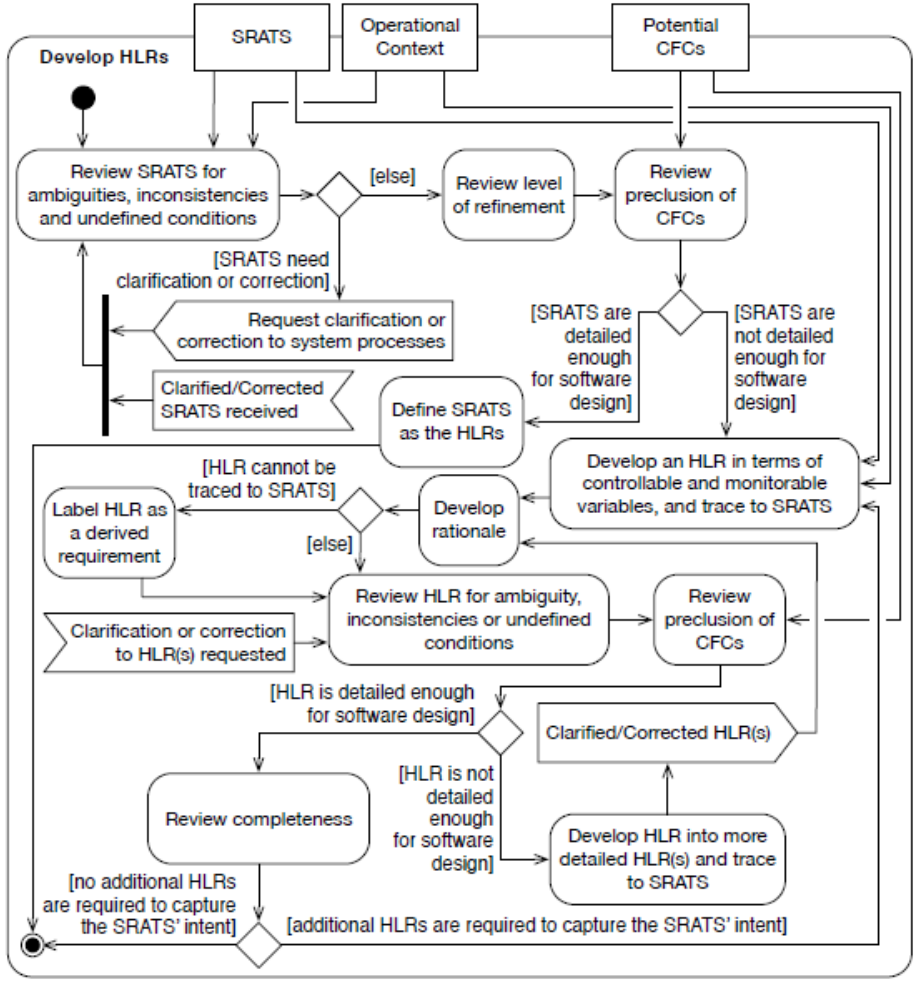
Establishing High Level Requirements

1. HLRs are defined by natural language
2. Refine SRATS to eliminate ambiguity and envelop operations requirements
3. Refer to SRATS for clarity if HLRs become convoluted



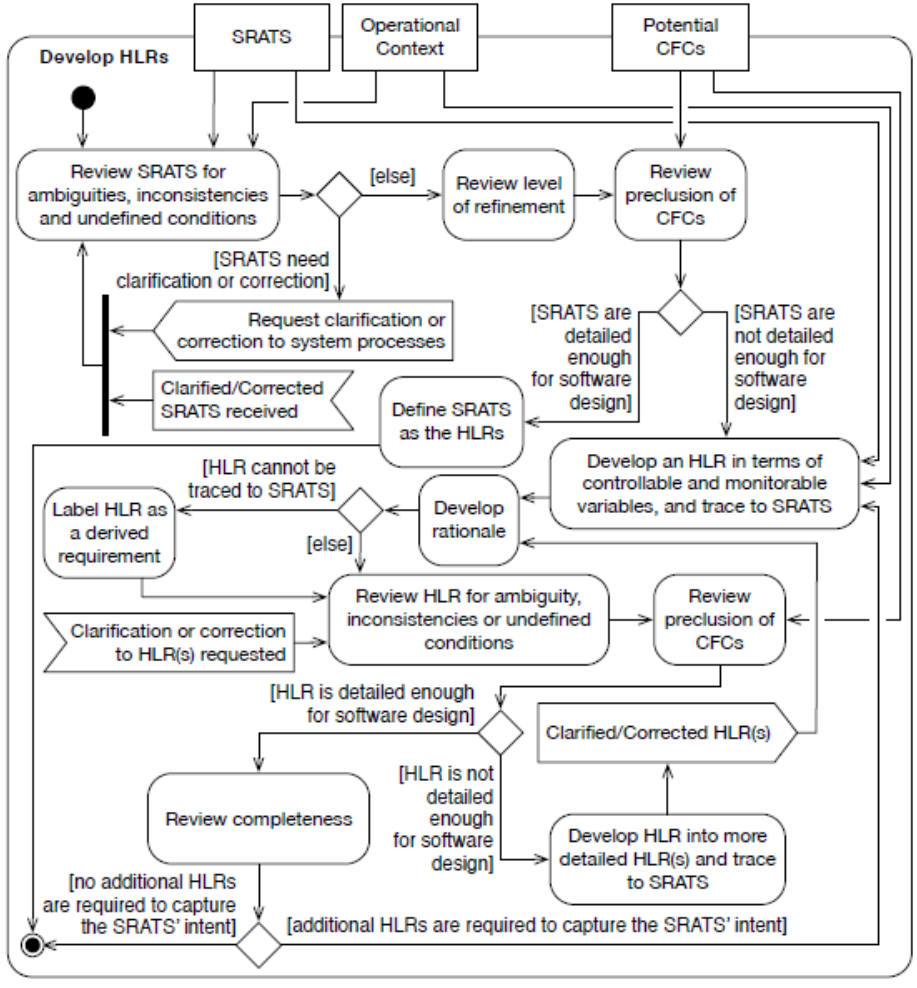
Establishing High Level Requirements

1. HLRs are defined by natural language
2. Refine SRATS to eliminate ambiguity and envelop operations requirements
3. Refer to SRATS for clarity if HLRs become convoluted

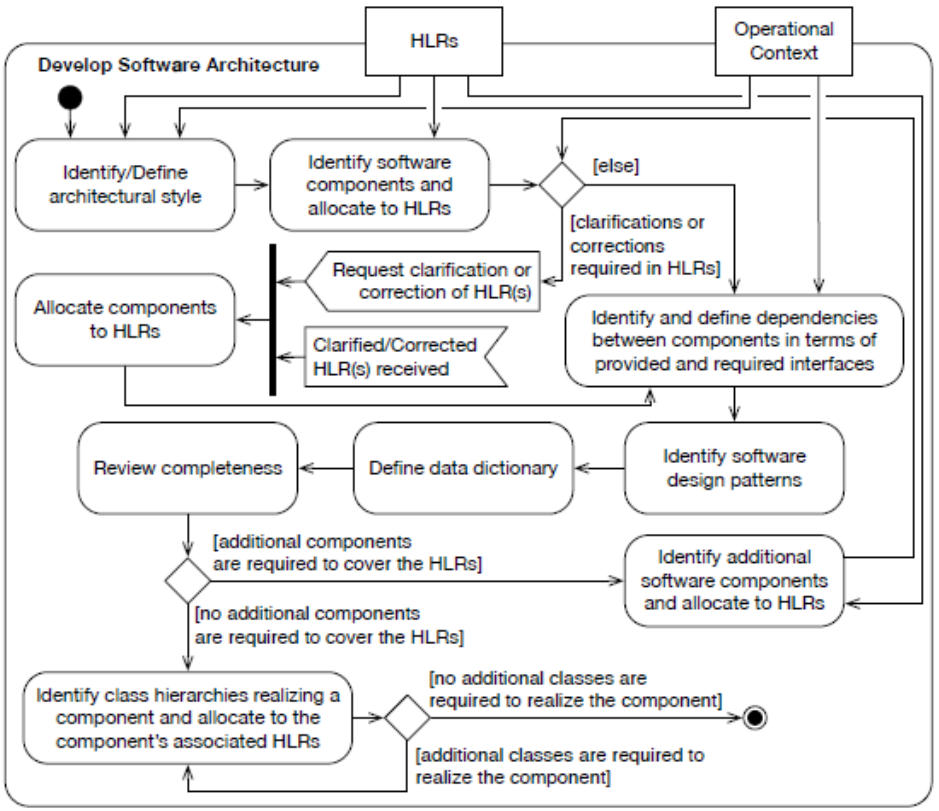


Establishing High Level Requirements

1. HLRs are defined by natural language
2. Refine SRATS to eliminate ambiguity and envelop operations requirements
3. Refer to SRATS for clarity if HLRs become convoluted

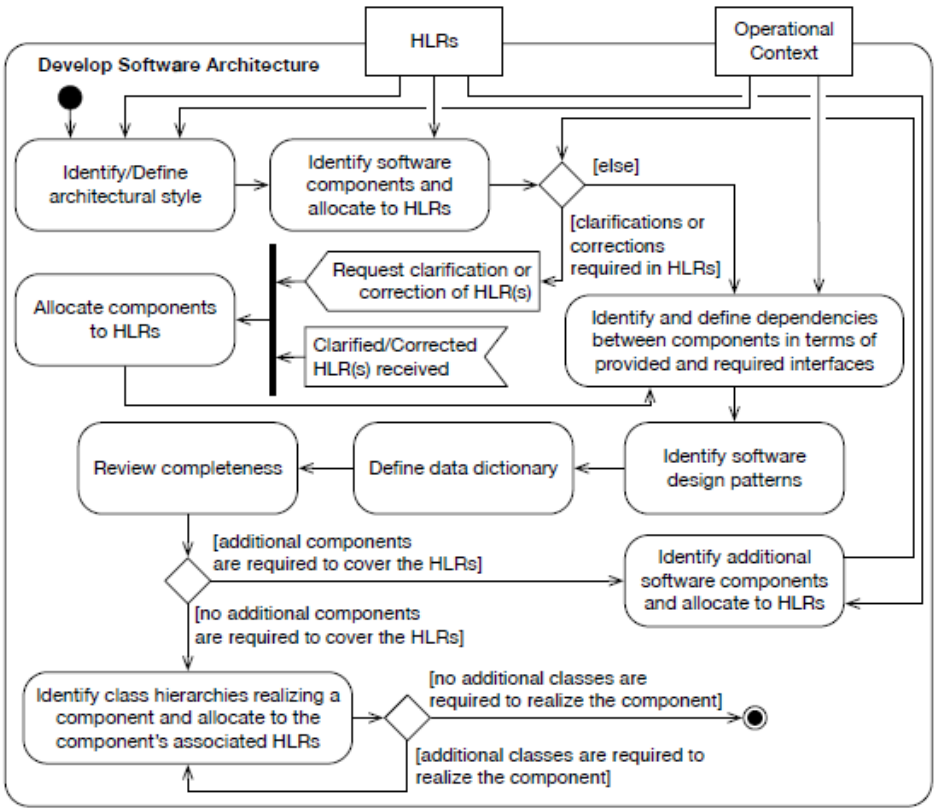


Establishing Software Architecture



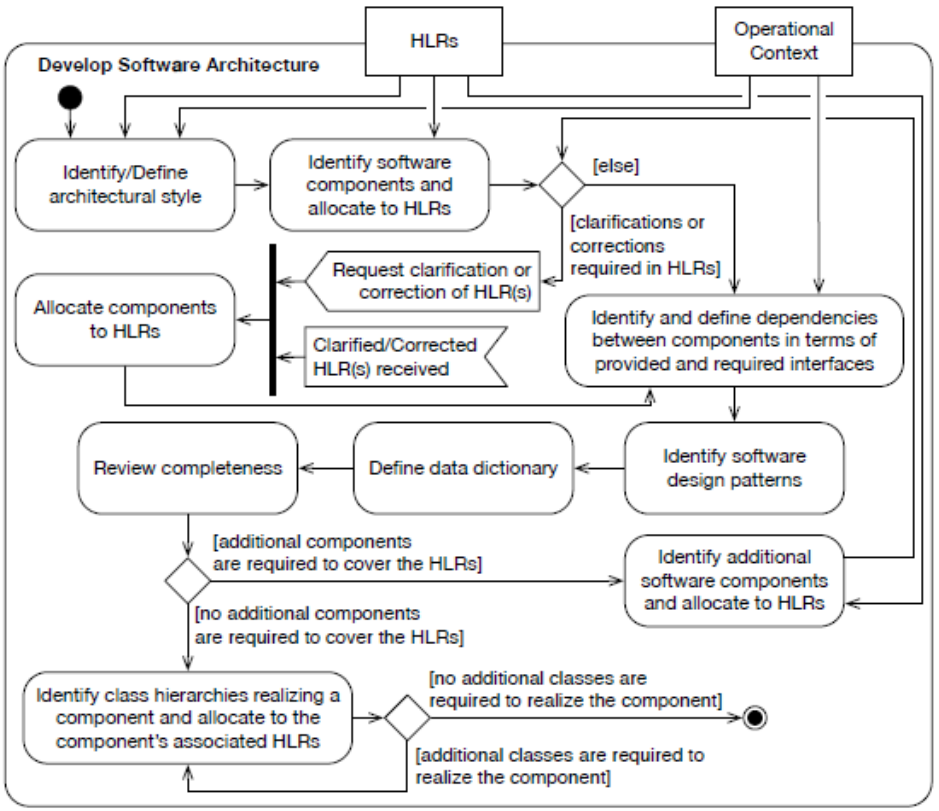
1. Establish what software *components* will be necessary for each HLR
2. Identify the necessary interdependencies and required interfaces between components
3. Establish software *class* hierarchies within each HLRs components

Establishing Software Architecture



1. Establish what software *components* will be necessary for each HLR
2. Identify the necessary interdependencies and required interfaces between components
3. Establish software *class* hierarchies within each HLRs components

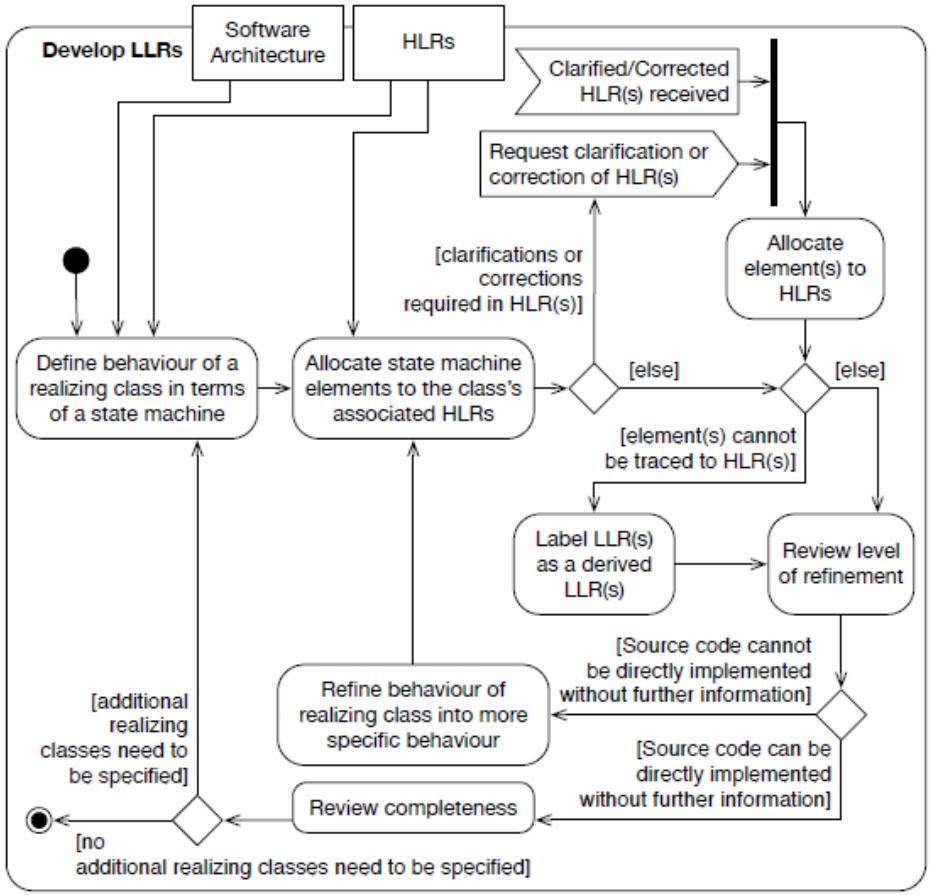
Establishing Software Architecture



1. Establish what software *components* will be necessary for each HLR
2. Identify the necessary interdependencies and required interfaces between components
3. Establish software *class* hierarchies within each HLRs components

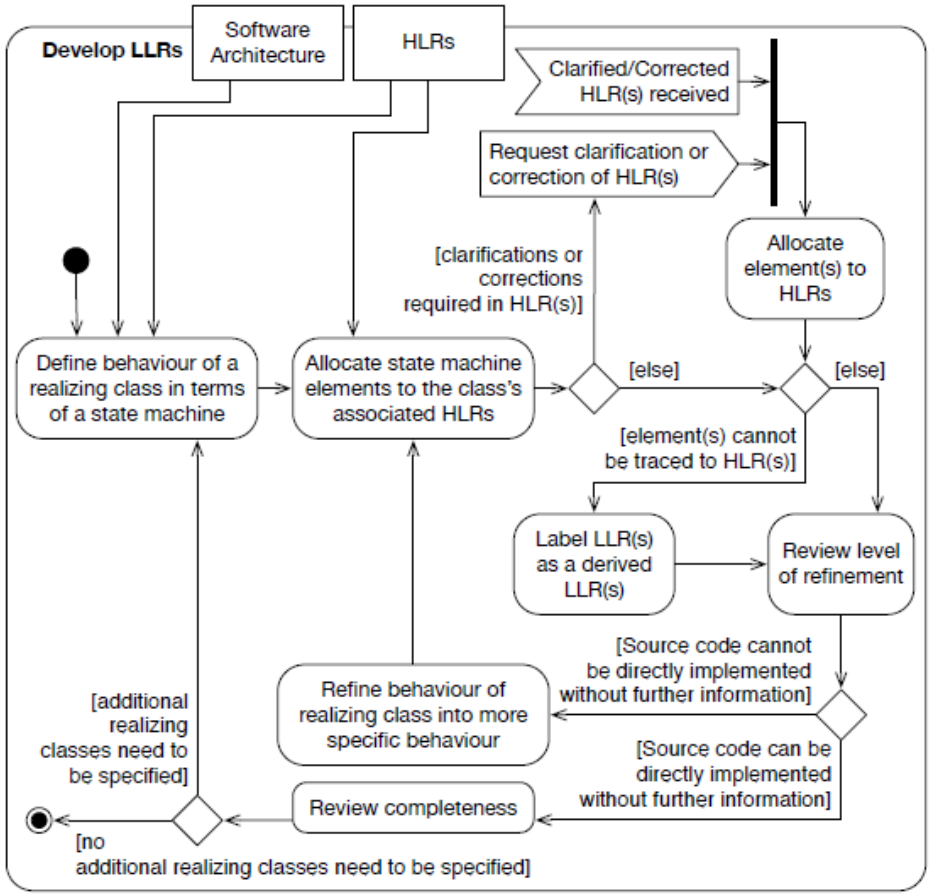
Establishing Low Level Requirements

1. Define expected behavior of each software class for their software components
2. Use HLR guidelines to refine ambiguous software requirements
3. Determine if more information is necessary to implement source code and refine where necessary



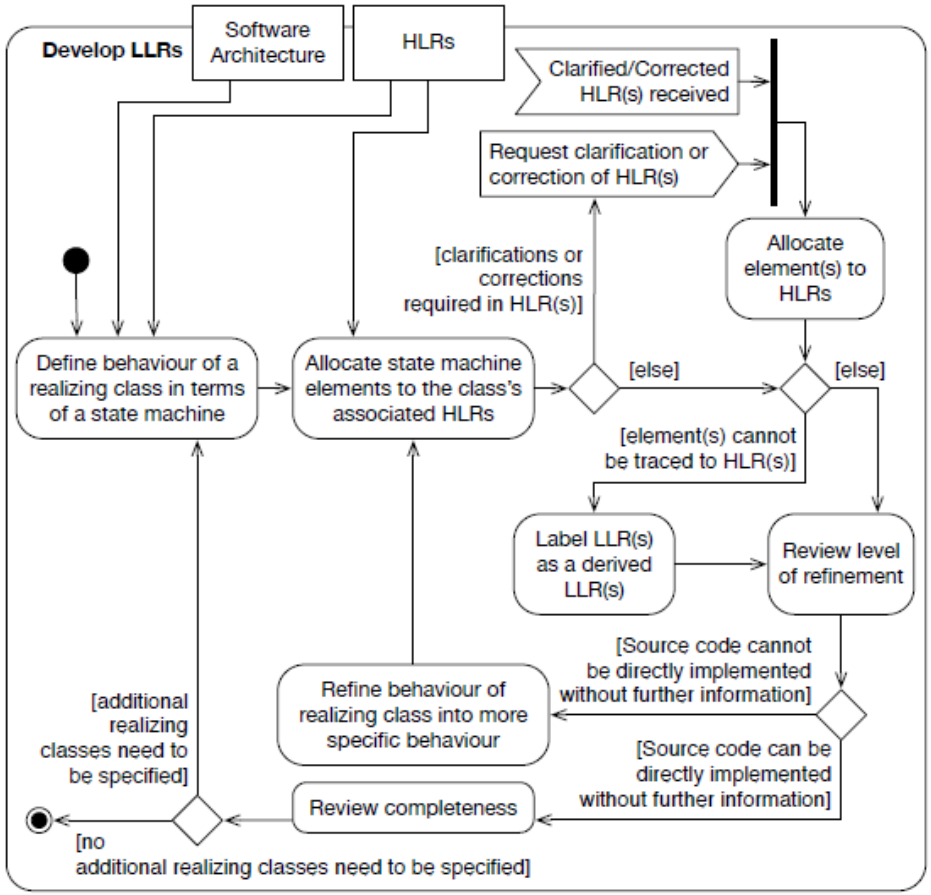
Establishing Low Level Requirements

1. Define expected behavior of each software class for their software components
2. Use HLR guidelines to refine ambiguous software requirements
3. Determine if more information is necessary to implement source code and refine where necessary



Establishing Low Level Requirements

1. Define expected behavior of each software class for their software components
2. Use HLR guidelines to refine ambiguous software requirements
3. Determine if more information is necessary to implement source code and refine where necessary



State Machines in Unified Modeling Language

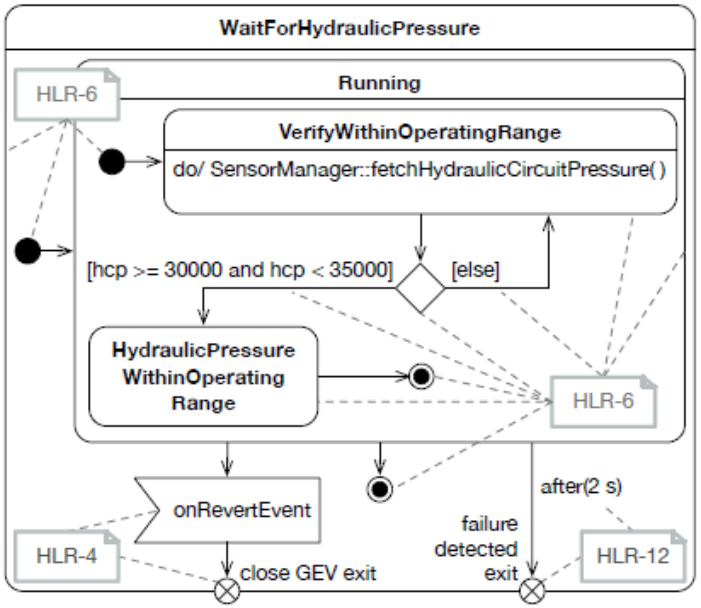
A state machine is a mathematical model of computation

Every state has a previous state, destination states, and the necessary conditions to change states

Unified Modeling Language is used to visualize system design

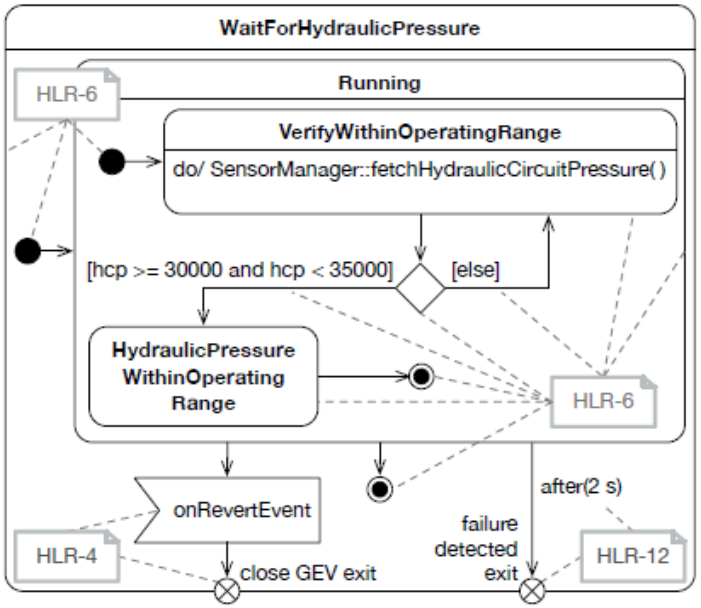
UML state machines represents the status of a system

Example of LLR in UML Notation



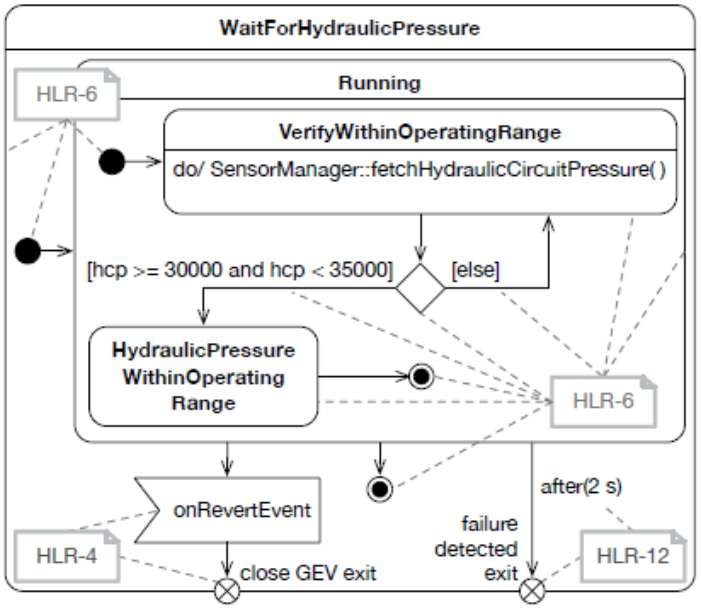
- HLR-6: Hydraulic Circuit Pressure Requirement 30,000kPa - 35,000kPa
- Initiate Verify Within Operating Range LLR
- HLR-4: Terminate and Revert Requirement
- HLR-12: Hydraulic System Failure Requirement
- Else HLR-6 terminates successfully

Example of LLR in UML Notation



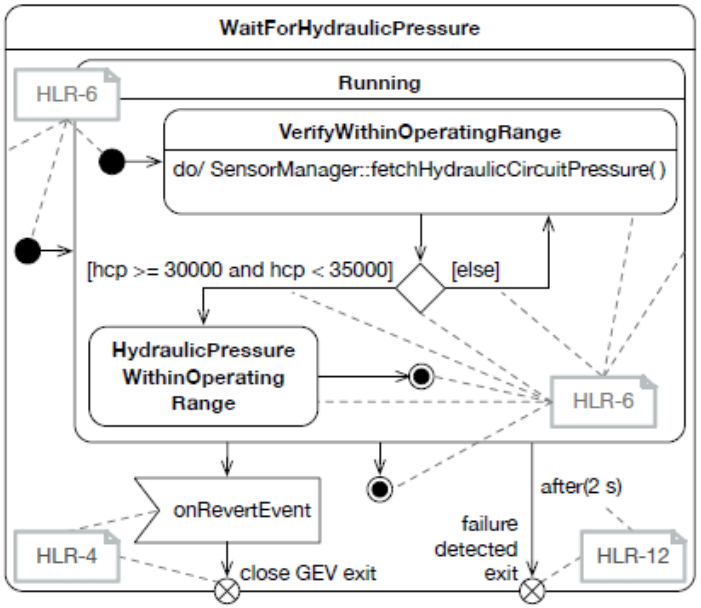
- HLR-6: Hydraulic Circuit Pressure Requirement 30,000kPa - 35,000kPa
- Initiate Verify Within Operating Range LLR
- HLR-4: Terminate and Revert Requirement
- HLR-12: Hydraulic System Failure Requirement
- Else HLR-6 terminates successfully

Example of LLR in UML Notation



- HLR-6: Hydraulic Circuit Pressure Requirement 30,000kPa - 35,000kPa
- Initiate Verify Within Operating Range LLR
- HLR-4: Terminate and Revert Requirement
- HLR-12: Hydraulic System Failure Requirement
- Else HLR-6 terminates successfully

Example of LLR in UML Notation



- HLR-6: Hydraulic Circuit Pressure Requirement 30,000kPa - 35,000kPa
- Initiate Verify Within Operating Range LLR
- HLR-4: Terminate and Revert Requirement
- HLR-12: Hydraulic System Failure Requirement
- Else HLR-6 terminates successfully

Conclusions

The lack of transparency in avionics industry design methodology and documentation are shortcomings in current design practices

An explicit and generalized design methodology similar to what as presented outlines the importance of a transparent requirements specification model

Acknowledgements and Special Thanks

I'd like to give thanks to:

Family and friends

Computer Science Professors and Colleagues

Audience

References


Erik Blasch, Paul Kostek, Pavel Paces, and Kathleen Kramer, *Summary of avionics technologies*, IEEE Aerospace and Electronic Systems Magazine **30** (2015), 6–11.


John Rushby, *New challenges in certification for aircraft software*, Proceedings of the Ninth ACM International Conference on Embedded Software (New York, NY, USA), EMSOFT '11, ACM, 2011, pp. 211–218.

Discussion

Questions?