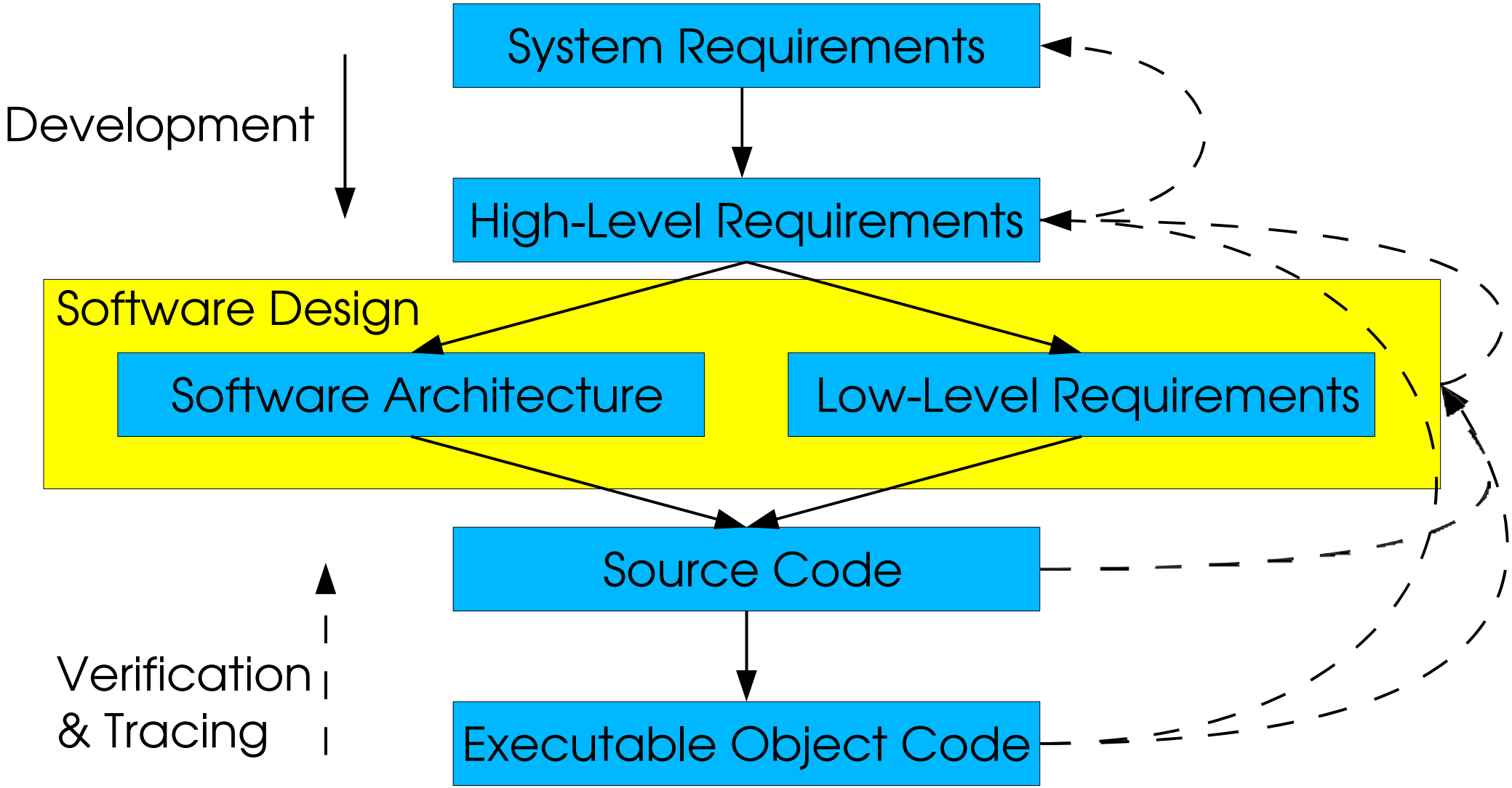
aicas technology brief

New Standards for Software in Aviation

Realtime Java Technology in Avionics Systems

Dr. James J. Hunt
CEO, aicas
JTRES 2010, Prague

DO-178B Verification



SC-205 / WG-71

Lead by RTCA and EUROCAE

Update software standards for aviation

- DO-178B/ED-12B: flight software regulations
- DO-248B/ED-94B: flight software addendum
- DO-278/ED-109: ground support software

Open to all interested parties

Organized in seven subgroups

SC-205 / WG-71 Subgroups

SG-1: Document Integration

SG-2: Issues and Rationale

SG-3: Tool Qualification

SG-4: Model Bases Design and Verification

SG-5: Object-Oriented Technology

SG-6: Formal Methods

SG-7: Safety and CNS Related Considerations (communication, navigation, surveillance)

SG-5: Object-Oriented Technology

Provide a supplement to DO-178C/ED-12C for object-oriented and related technologies (OOT)

- Not just pure object-oriented langue features
- Identifies technology specific vulnerabilities
- Provide guidance for OOT software development
- Resulted in both new objectives and clarification of existing objective for OOT
- Address outstanding certification Issues w/OOT

Work started w/OOTiA Handbook

Object-Oriented Technology in Aviation

List of possible problems, but no real solutions

- 124 Issues raised
- 40 deemed irrelevant to Supplement
- Many code style issues for C++

Volumes

- 1: Handbook Overview
- 2: Considerations and Issues
- 3: Best Practices
- 4: Certification Practices

Example OOTiA Guidance

Three parents rule:
Any class near the top of the hierarchy with three or more parents warrants careful review.

Top heavy composition rule:
Any class near the top of the hierarchy that inherits more than 20 features from each of two or more parent classes warrants careful review.

Top to bottom rule:
Any class hierarchy that contains more classes near the top of the hierarchy than near the bottom warrants careful review.

Basic Concepts

Classes and Object

Method Dispatch

Hierarchic Encapsulation

Polymorphism

Types and Safety

Function Passing and Closures

Polymorphism

Universal polymorphism

- Inclusion polymorphism:
  inheritance, subtyping, and subclassing
- Parametric polymorphism:
  generics and templates

Ad hoc polymorphism

- Overloading
- Coercion:
  some forms of type casting

Types and Safety

Subclass, subtype equivalence

- Liskov (Leavens) substitution principle
- Arrays and collections

Method and class specification:
design by contact

- Preconditions: acceptable input values
- Postconditions: return values, including exceptions and errors, and side effects
- Invariants

LSP and Requirements

A subclass must fulfill the requirements of all its superclasses.

Each method in the subclass that is also declared in a superclass should have

- preconditions that are the same or weaker than the method in the superclass,
- postconditions that are the same or stronger than the method in the superclass, and
- Invariants that are not weaker.

Key Features

Inheritance and redefinition

Parametric Polymorphism

Type conversion

Overloading

Exceptions and exception handling

Virtualization Techniques

Dynamic memory management

Inheritance and Redefinition

Interface vs. Implementation

Single vs. Multiple

Vulnerabilities

- Nondeterministic dispatch time
- Semantic dissonance
- Implementation dissonance

Objectives

- Ensure local type consistency
- Include full class model in design

Local Type Safety

Subclasses are Subtypes

- Subclasses fulfill requirements of superclasses
- Think Liskov Substitution Principle
- Use delegation instead of inheritance for reuse

Local

- Where substitution can occur
- Declared type vs. Actual type

Alternative: Exhaustive Testing

Parametric Polymorphism

Enables reuse without subtyping

Vulnerabilities

- Substitution mismatch
- Unverified code

Objectives

- Ensure semantic consistency
- Ensure all code is covered

Type Conversion

View change vs. Representation change

Vulnerabilities

- Data loss
- Data corruption or exception

Objectives

- Ensure that type conversions are safe

Overloading

Can aid in program understanding

Vulnerabilities

- unintended behavior when combined with automatic type conversion
- Naming dissonance

Guidance

- Address in coding standards

Exceptions and Exception Handling

Helps with program clarity by separating exceptional behavior from normal behavior

Vulnerability
failure resulting from uncaught or improperly handled exception

Objective
ensure that all exceptions that can be thrown are caught and properly handled,
i.e., test coverage includes exceptional as well as normal control paths

Virtualization Techniques

Vulnerability
interpreted code is not adequately validated
because it was treated as data, not code

Objective
Certify system in layers

- Certify interpreter where its input is treated as data
- Certify interpreted program as code where interpreter is treated as execution platform

Applies to any data that is interpreted

Dynamic Memory Vulnerabilities

1. Ambiguous references
2. Fragmentation starvation
3. Deallocation starvation
4. Premature deallocation
5. Indeterministic allocation or deallocation
6. Lost update or stale reference
7. Heap memory exhaustion

Dynamic Memory Safety Objectives

1. Timely Deallocation
2. Fragmentation Avoidance
3. Unique Allocation
4. Reference Consistency
5. Deterministic Execution
6. Atomic Move
7. Sufficient Memory

Memory Management Techniques

Technique	Objectives						
	Unambiguous Reference	Fragment. Avoidance	Timely Deallocation	Reference Consistency	Determinisitc Deallocation	Atomic Move	Sufficient Memory
Object Pooling	AC	AC	AC	AC	MMI	N/A	AC
Stack Allocation	AC	MMI	MMI	AC	MMI	N/A	AC
Scope Allocation	MMI	MMI	MMI	AC	MMI	N/A	AC
Manual Heap Allocation	AC	?	AC	AC	MMI	N/A	AC
Garbage Collection	MMI	MMI	MMI	MMI	MMI	MMI	AC

AC = application code, MMI = memory management infrastructure, N/A = not applicable, and ? = difficult to ensure by either AC or MMI.

Certifying a Garbage Collector

Not possible for all collector

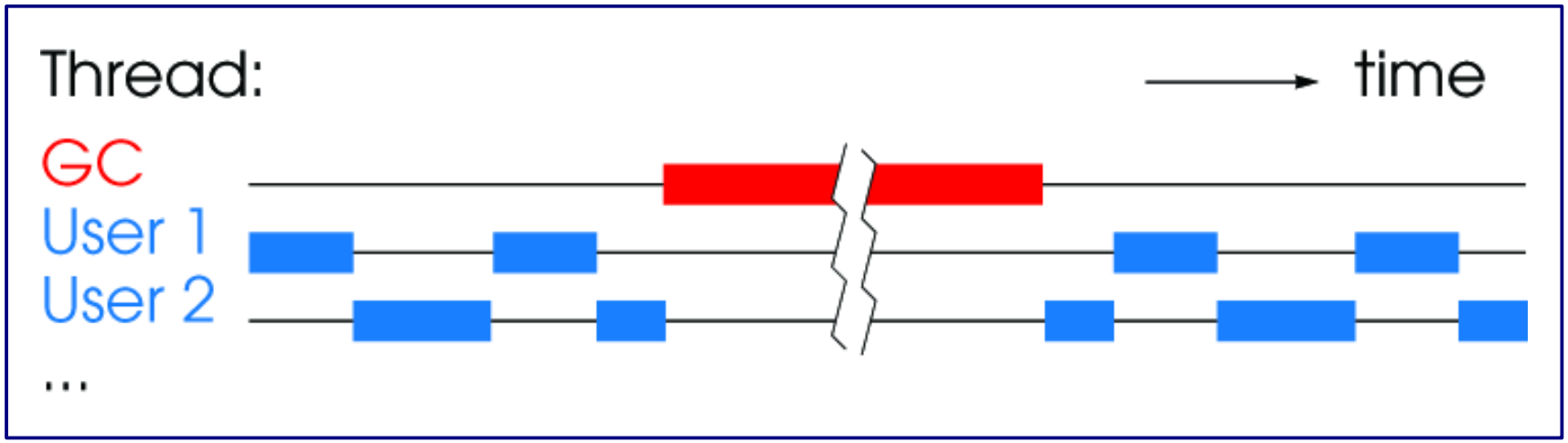
- Must be exact
- Must be deterministic; no unbound steps
- Must consider impact on scheduling and WCET

Some types of commercial realtime collectors

- Paced GC
- Slack GC
- Work-based GC

Classical Garbage Collection

GC can interrupt execution for long periods of time:

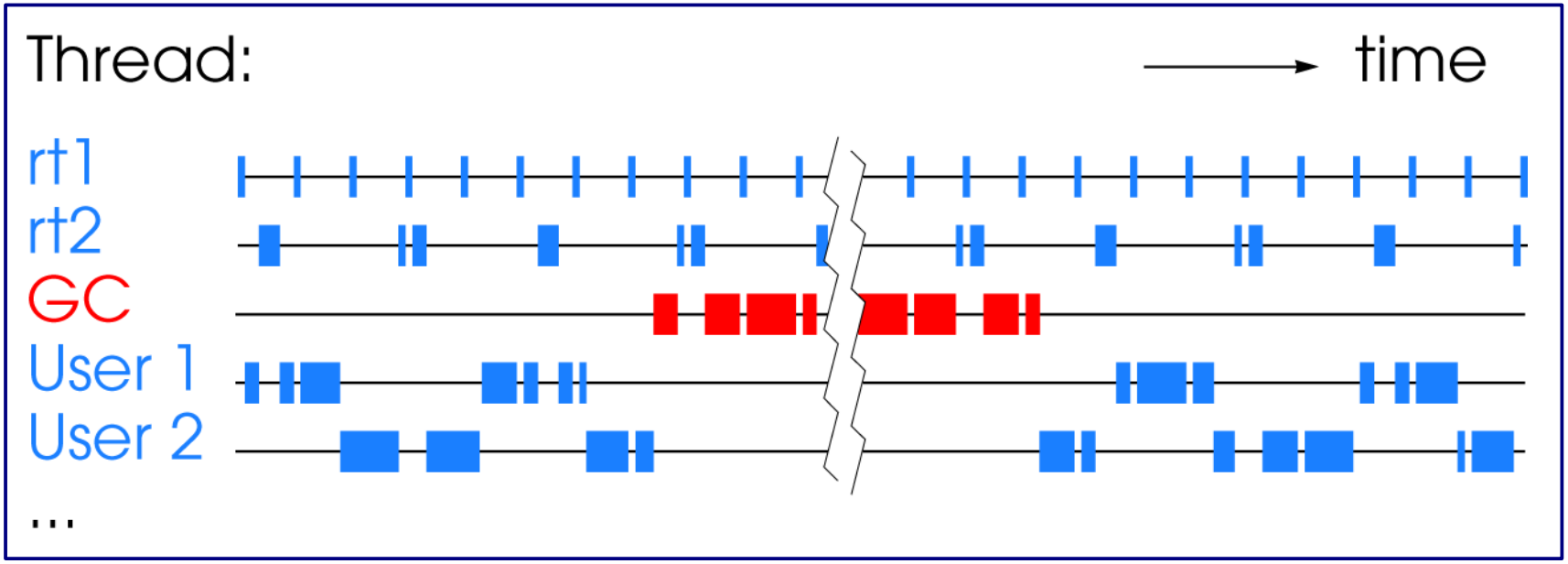


Problem

long, unpredictable pauses during execution

RTSJ with Classic Garbage Collection

No heap threads can interrupt garbage collector:



The application must be split into a realtime and a nonrealtime part.

Realtime Garbage Collection

Paced garbage collector

- Run GC at a high priority
- Runs at given interval, for given amount of time
- Programmer must provide both maximum memory use and maximum allocation rate

Slack garbage collector

- Run GC at lower priority than realtime tasks
- Runs when processor cycles are available
- Programmer must provide both maximum memory use and maximum allocation rate

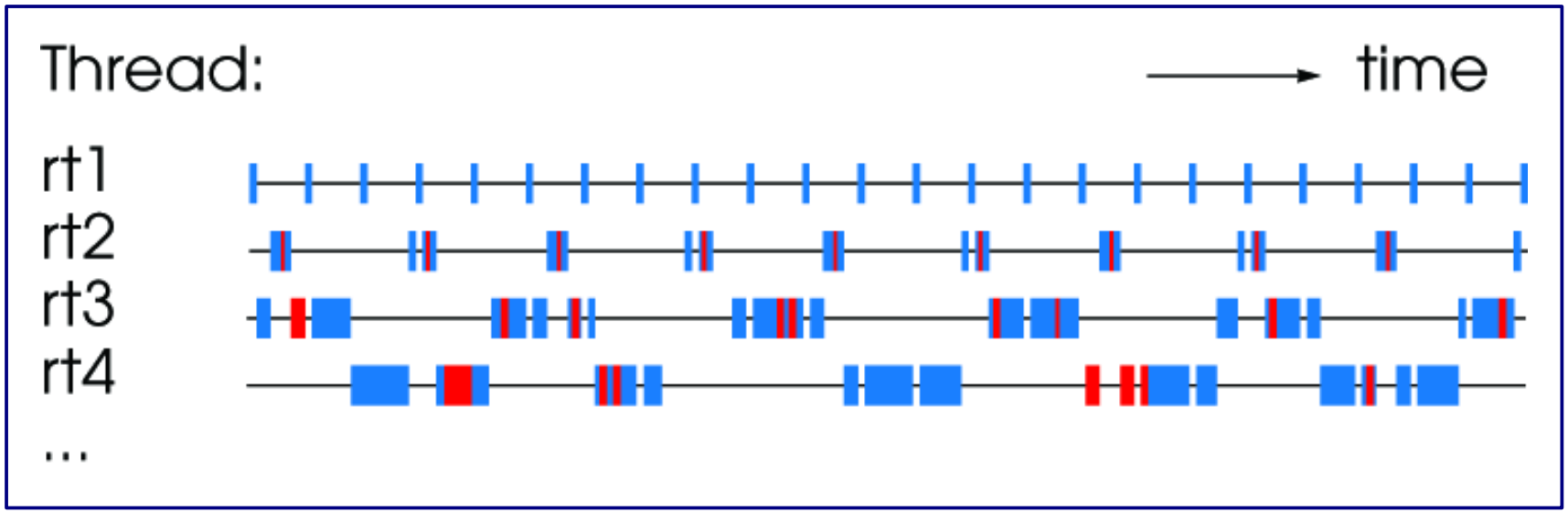
Realtime Garbage Collection

Work based garbage collector

- No GC thread
- GC borrows application thread
- Need only determine maximum memory use
- No read barriers needed
- Low latency

Work-Based Garbage Collector

All Java Threads are realtime threads



- GC work is performed at allocation time
- GC work must be sufficient to recycle enough memory before free memory is exhausted
- Execution time of all allocations must be bound

SG-3: Tool Qualification

Provides guidance for tools used to develop and verify avionic software such as

- UML code generator
- Model checker
- Formal analysis tool
- Test automation tool
- Emulator

Covers full tool life cycle

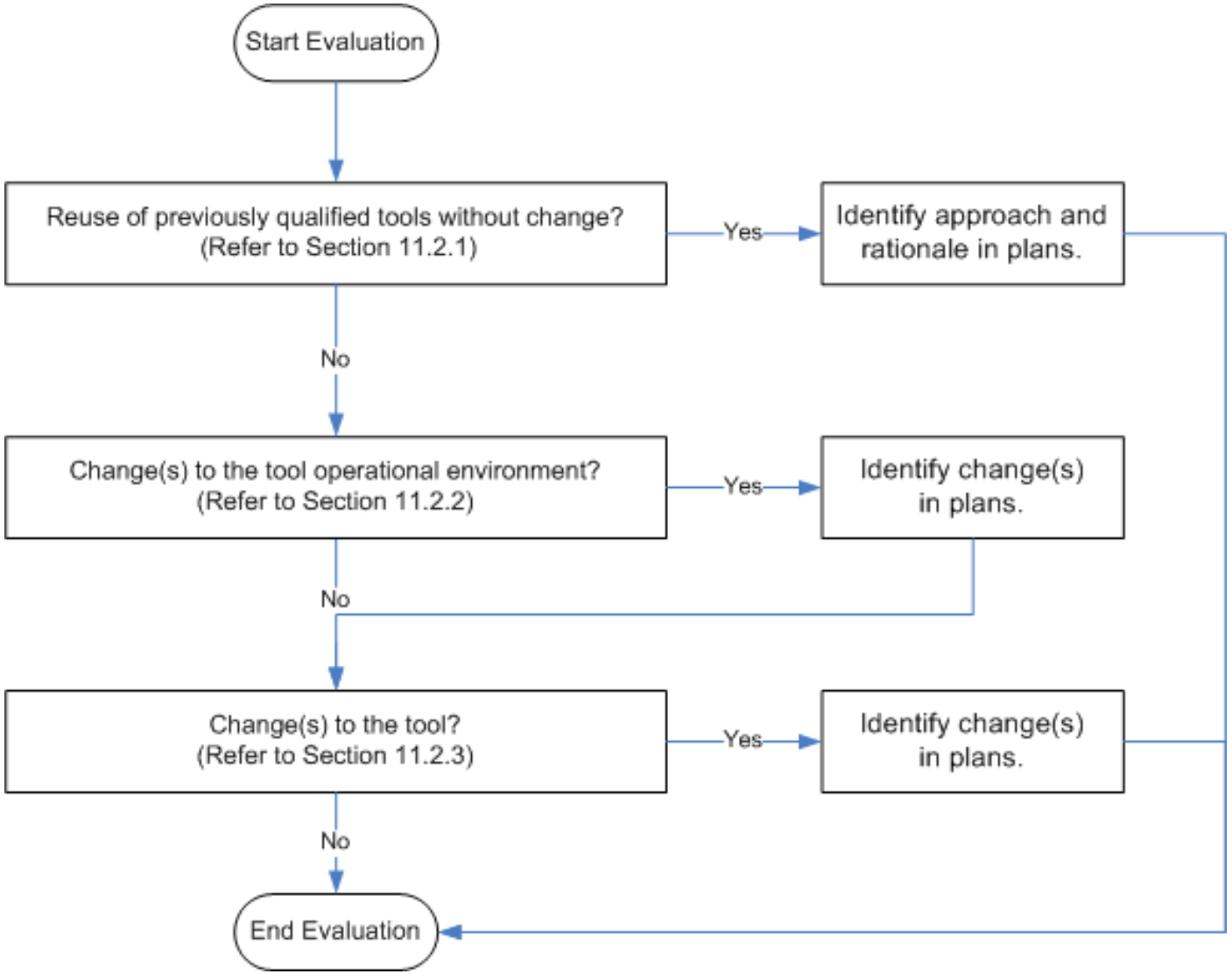
Tool Qualification Categories

DO-178B/ED-12B Tool Category & Definition	DO-178C / ED-12C Tool Qualification Criteria & Definition
Development tools: tools whose output is part of airborne software and thus can introduce errors.	Criteria 1: tool whose output is part of the resulting software and could insert errors.
Verification tools: Tools that cannot introduce errors, but may fail to detect them.	Criteria 2: A tool that automates the verification process and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of • verification process not automated by tool or • development process which could impact the resulting software. Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

Tool Qualification Levels

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Tool Reuse


Start Evaluation
Reuse of previously qualified tools without change?
(Refer to Section 11.2.1)
Yes
Identify approach and rationale in plans.
No
Change(s) to the tool operational environment?
(Refer to Section 11.2.2)
Yes
Identify change(s) in plans.
No
Change(s) to the tool?
(Refer to Section 11.2.3)
Yes
Identify change(s) in plans.
No
End Evaluation

SG-6: Formal Methods

analysis of software (and hardware) using rigorous mathematical methods such as calculi, logic, automata, or graph theory

Alternate means of verifying avionics software

- Reduce but not eliminate testing
- Increase safety

Provides guidelines for using formal methods

Strengths and Weaknesses of Testing

Strengths

- Well understood
- Mostly language independent
- Includes execution environment

Weaknesses

- Hard to cover all execution paths
- Hard to cover all possible parallel paths
- Internal states are not visible

Why Formal Methods?

Errors can not be tolerated in safety critical applications.

Security is not possible without safety.

System complexity is increasing dramatically.

Increasingly critical decisions are being made automatically in software.

Testing is not sound.

Static Analysis (Formal)

Type Analysis

Control Flow Analysis

Data Flow Analysis

Abstract Interpretation

Symbolic Execution

Model Checking

Deductive verification

Impact of Realtime Java Programming

Type safety is a crucial factor in OOT certification

Use alternates to inheritance for code sharing

- Generics
- delegation

Realtime constraints will limit what constructs and libraries can be used

Garbage can be used as long as timing constraints can be met.

Certification Issues

- Class initialization
- Dynamic dispatch
- Garbage collection
- Unchecked exceptions
- Dynamic class loading
- Just in time compilation
- Reflection
- Asynchronous transfer of control

Conclusion

DO-178C will provide more consistent treatment of OO and other nonprocedural languages.

Concrete guidance for dynamics memory management and interpretation.

Supports a stronger role for tools.

Encourages using formal methods.

Certification for realtime Java will become easier without endangering safety.

More attention to type safety.

Contact Information

jjh@aicas.com

+49 721 663 968 22

aicas GmbH

Haid-und-Neu-Straße 18

D-76139 Karlsruhe