Open Systems Dependability

December 16, 2010

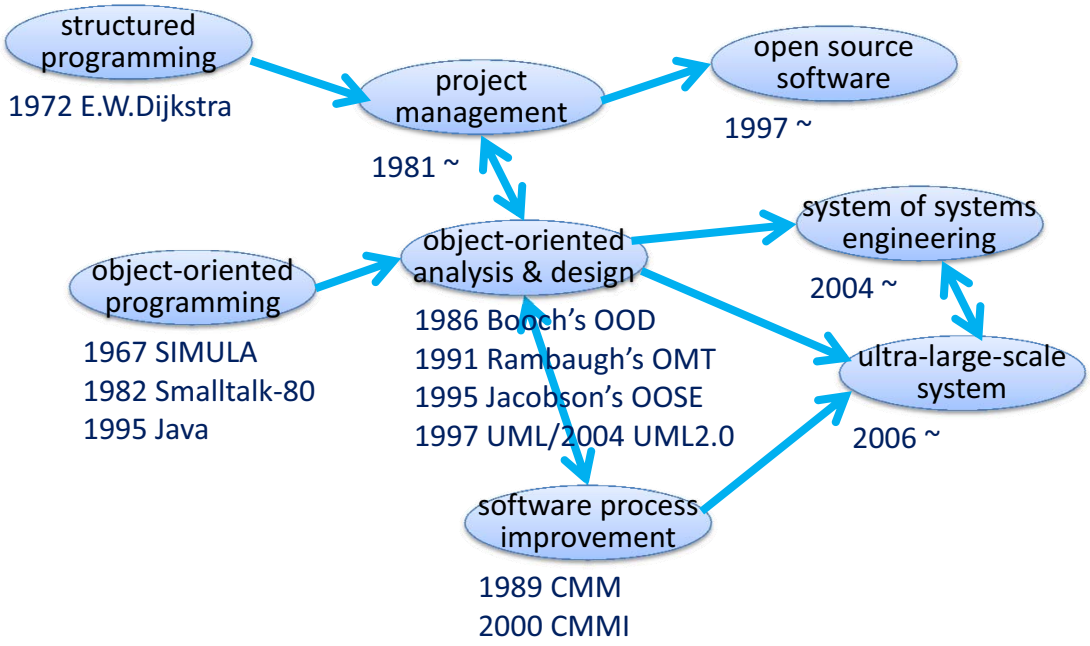
Mario Tokoro
Research Supervisor
JST/CREST Dependable Embedded OS for Practical Use
(Sony Computer Science Laboratories, Inc.)

科学技術振興機構
Japan Science and Technology Agency

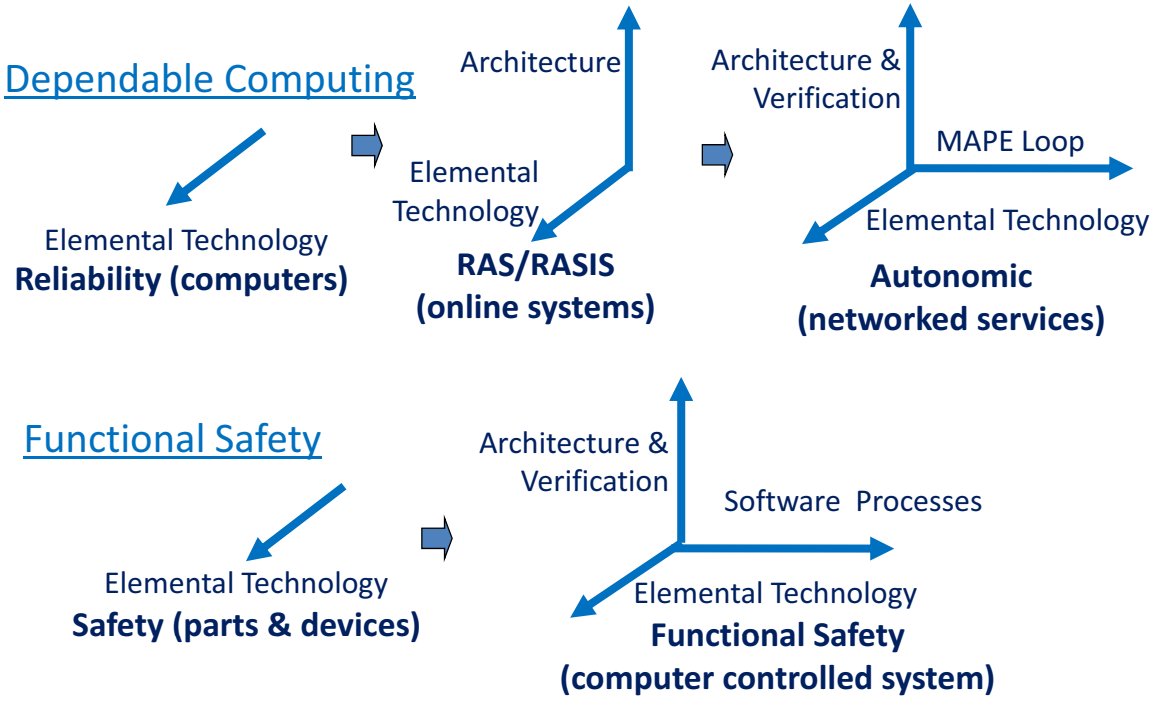
Background

- Demands for the dependability of huge and complex software systems
  - which include black box software such as legacy codes and off-the-shelf components
  - which are connected to networks that may cause security and integrity problems
- Demands for coping with environmental and requirement changes in operation
  - functions, user interfaces, performance, etc
  - networks and services on networks
- Necessity of Continuous Operations
- Consciousness to performance/cost over lifecycle
- Increased accountability to service/system providers

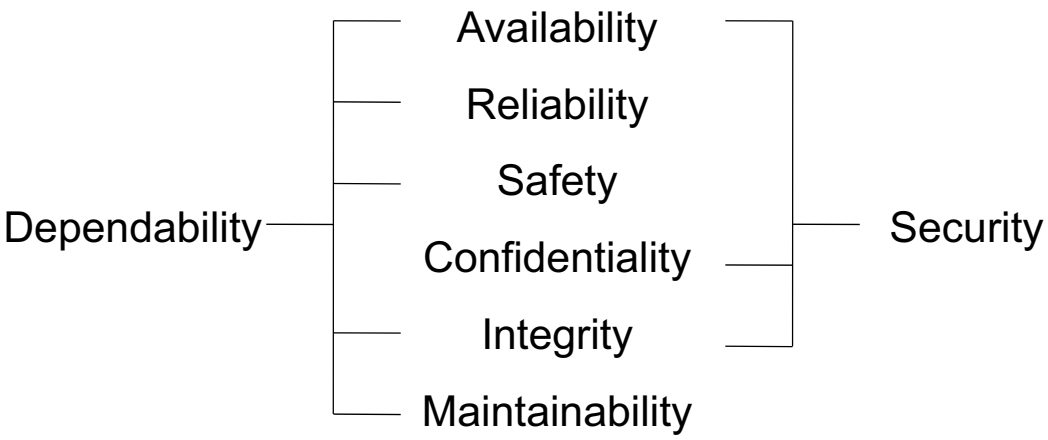
Software Engineering
A Brief Historical Review



Dependability
A Brief Historical Review



Dependability and Security
by IFIP WG10.4 (2004)



Standards and Guides

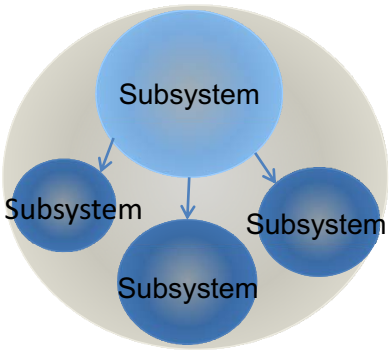
- Standards
  - IEC 61508: Functional Safety
  - IEC 60300-1: Dependability Management
  - IEC 60300-2: Dependability Program Elements and Tasks
  - ISO/IEC 1207: Software Life Cycle Processes
  - ISO/IEC 15288: System Life Cycle Processes
  - etc.
- Guides
  - CMMI: Capability Maturity Model Integration
  - DO-178B: Software Considerations in Airborne Systems and Equipment Certification
  - MISRA-C: Guidelines for the Use of the C Language in Vehicle Based Software
  - IEC 61713: Software Dependability through the Software Life-Cycle Processes – Application Guide
  - IEC 62347: Guidance on System Dependability Specifications
  - etc.

Are We OK with Existing Ones? or Do We Need a New Approach?

- Previous approaches to huge and complex software systems were based on the *Closed Systems Hypothesis*
  - a system can be composed of *complete* components
  - we suppose we can know the *whole* system and the *behaviors* of the whole system
- However, the hypothesis *cannot* hold, due to
  - the *incompleteness* of specifications and implementations
  - the *uncertainty* of environment and requirement changes to systems in operation
- We need a new approach based on the notion of *Open Systems*.

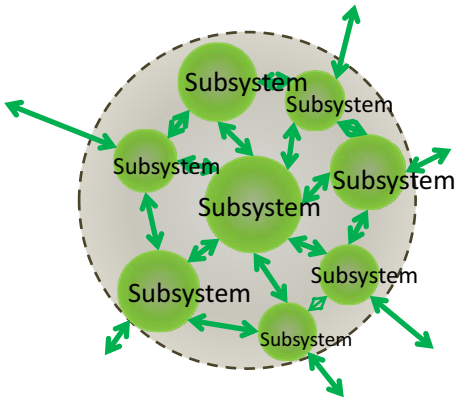
Closed Systems vs. Open Systems

Closed Systems



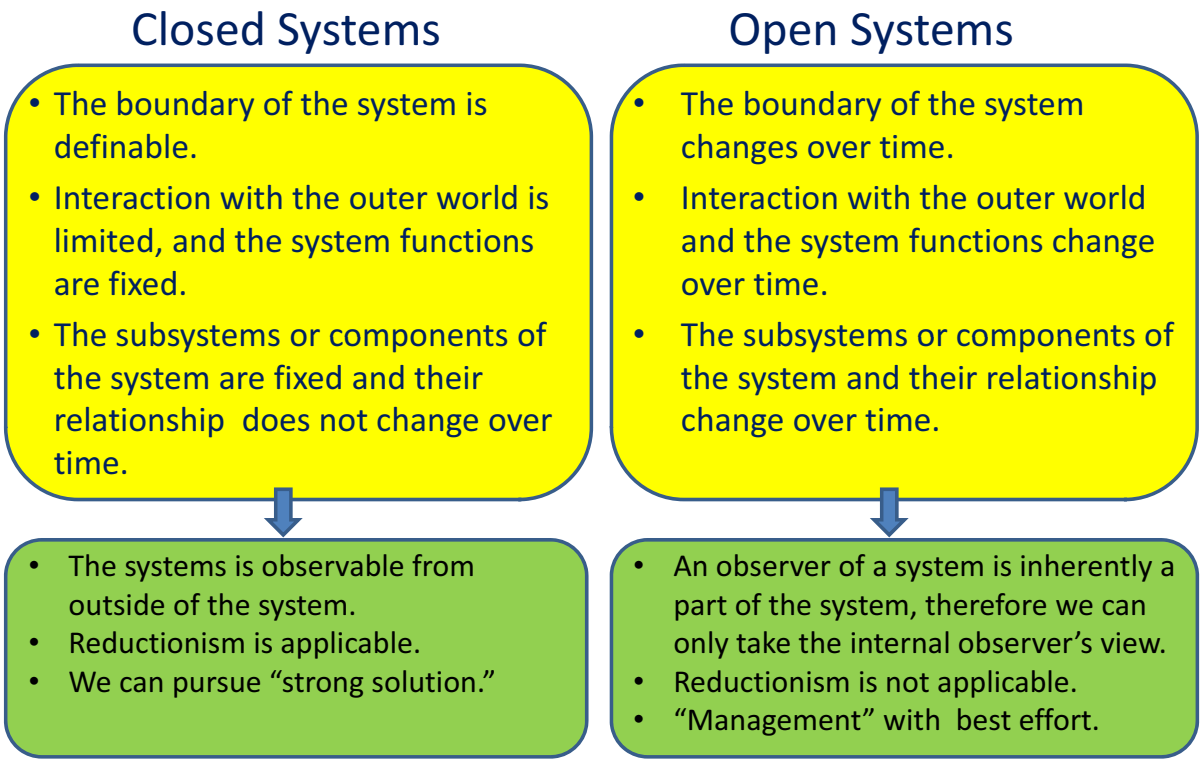
- The boundary of the system is definable.
- Interaction with the outer world is limited, and the system functions are fixed.
- The subsystems or components of the system are fixed and their relationship does not change over time.

Open Systems



- The boundary of the system changes over time.
- Interaction with the outer world and the system functions change over time.
- The subsystems or components of the system and their relationship change over time.

Comparison of Closed and Open Systems



A Huge and Complex Software System is an Open System

- It has potentiality of incidents due to
  - the *incompleteness* of specifications and implementations
  - the *uncertainty* of environment and requirement changes to systems in operation
- We need to secure dependability of a huge and complex system over lifecycle in a practical way, based on the perspective of Open Systems

Can We Consider a Huge and Complex Software System as a Closed System?

Yes, if we can assume:

- a system does not change for a certain period of time and
- the lifecycle of the system can be the accumulation of these periods of time.

However, if we need the system continues to give services while being modified, and possibly even in the case of incidence, it is extremely difficult to separate the phase that

- the system is in normal operation
- the system is being fixed and
- the system is being modified.

Open Systems Dependability: Definition

- Functions, structures, and boundaries of a huge and complex software system change over time. Hence, incompleteness and uncertainty are inherent to such a system, which may result in failures in the future.

- Open Systems Dependability is the ability
  1. to continuously prevent the said factors from causing failure,
  2. to take appropriate and quick action when failures occur to minimize damage,
  3. to safely and continuously provide the services expected by users as much as possible, and
  4. to maintain accountability for the system operations and processes.

How to Achieve Open Systems Dependability

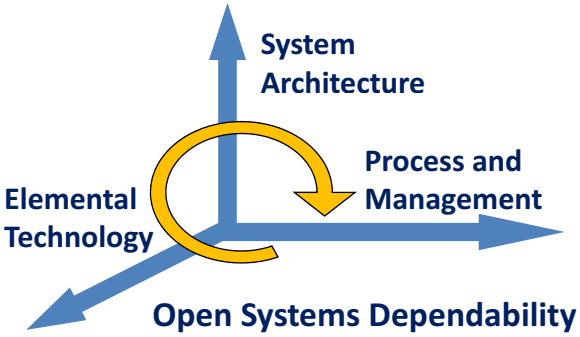
Can it be achieved by

- elemental technology?
- architecture? or
- process and management?

How to Achieve Open Systems Dependability

We think it can be achieved by

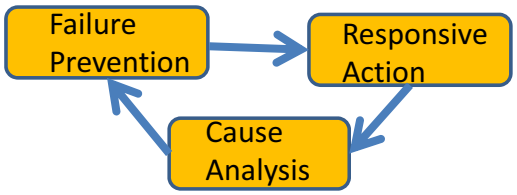
- process and management
- which is supported by architecture
- which is supported by elemental technologies



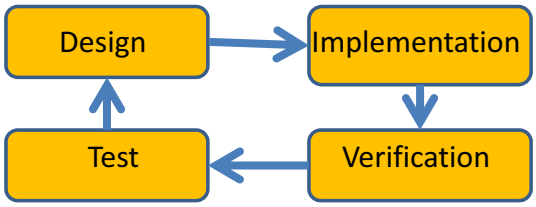
The DEOS Process (1)

Consists of two cycles:

- Failure Reacting Cycle
  - Failure prevention
  - Responsive action
  - Cause analysis



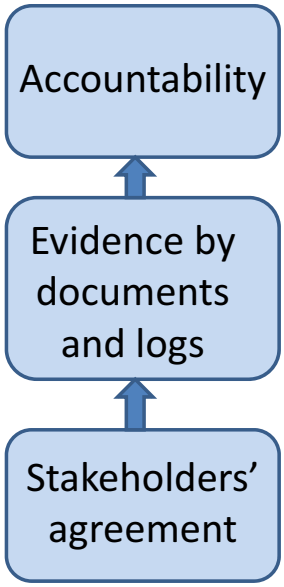
- Requirements/Environment Change Accommodation Cycle
  - Design
  - Implementation
  - Verification
  - Test



The DEOS Process (2)

Two most important issues are lacking:

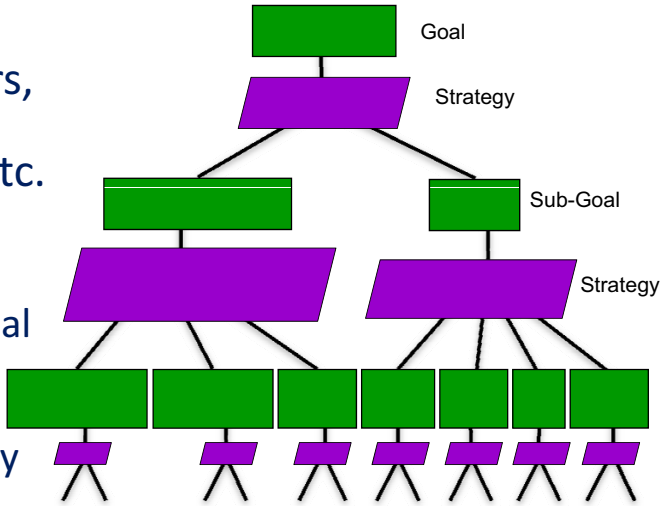
- We need to achieve accountability in case of incidence
- For this, we need to show evidence by
  - process documents e.g. design, implementation, verification, test, failure prevention, responsive action, cause analysis, and daily check and maintenance,
  - Logs that record the behavior of the system
- Based on stakeholders' agreement through
  - argumentation and documentation tools



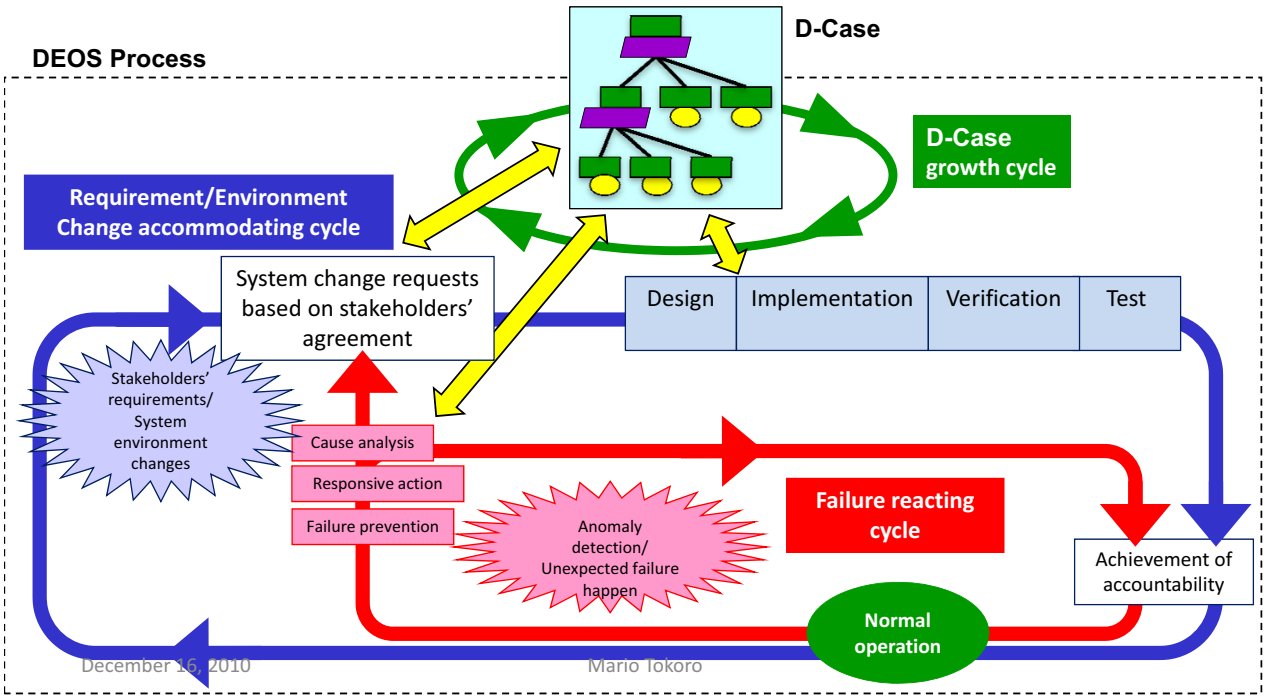
The DEOS Process (3)

How to make Stakeholders' Agreement?

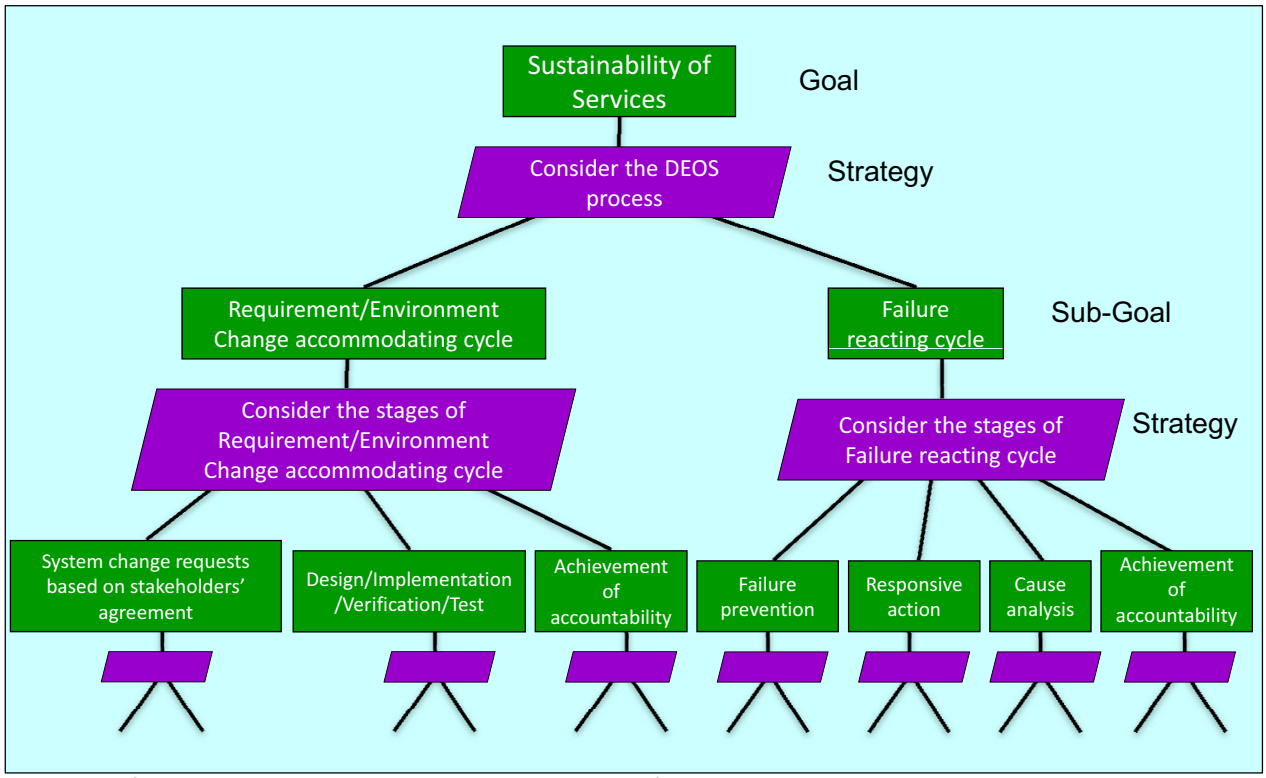
- Stakeholders: Users, Service/Product Providers, Systems Providers, Certifiers (Authorities), etc.
- D-Case:
  - Modified version of Assurance Case using Goal Structured Notation
  - Describes dependability agreements supported by evidence



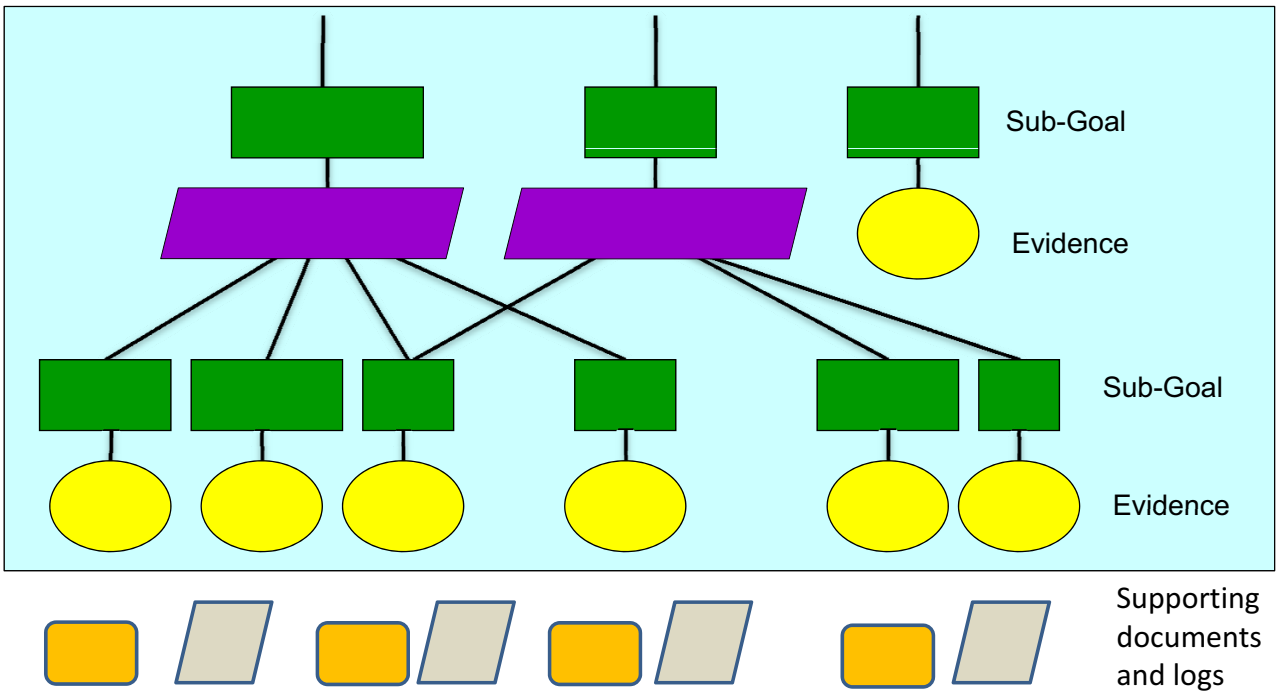
The DEOS Process (4)



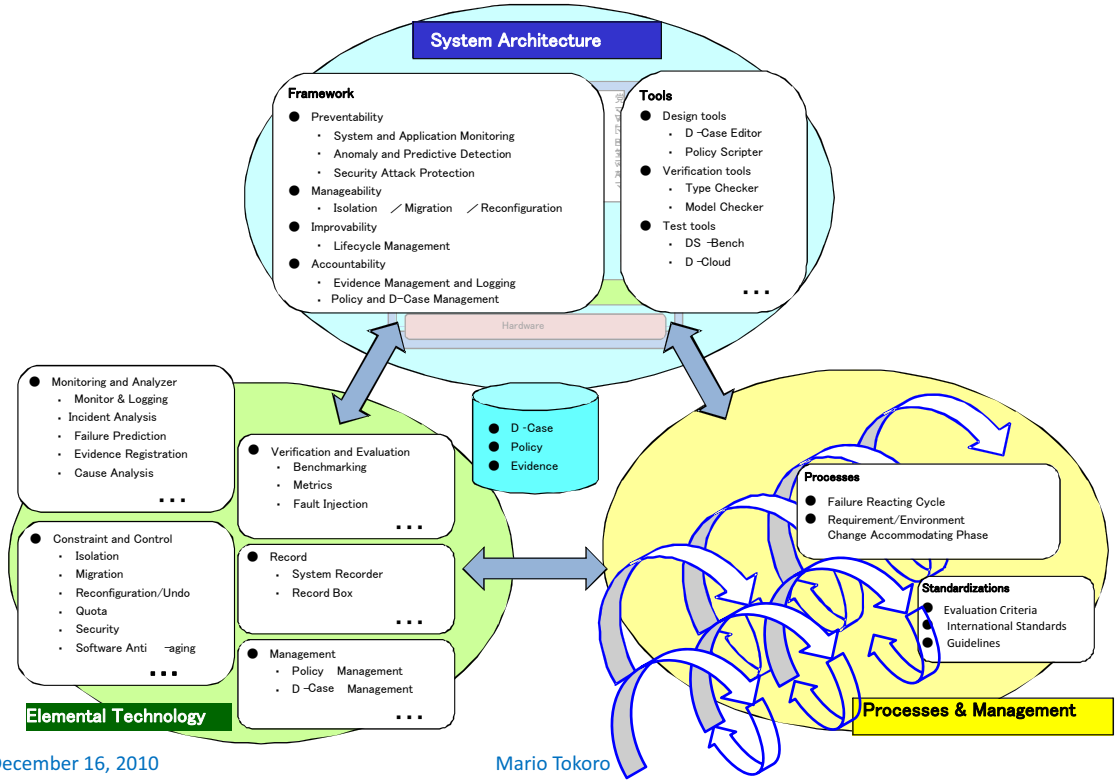
D-Case Top Structure



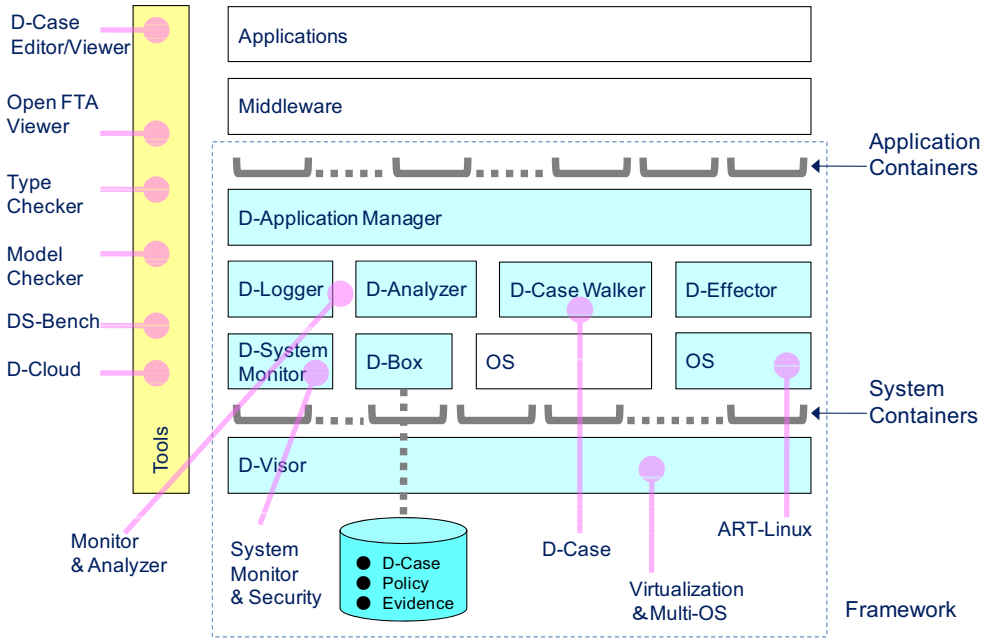
D-Case Bottom Structure



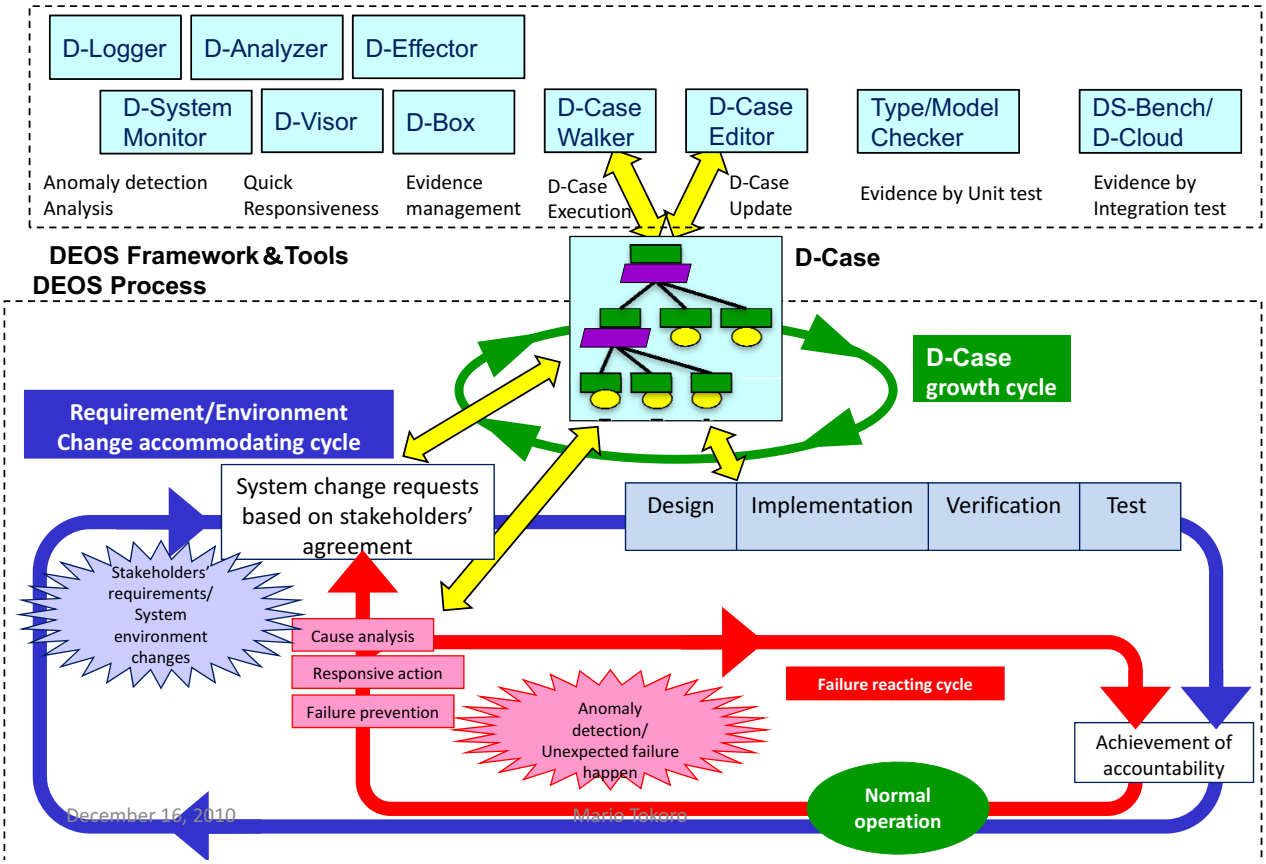
Realizing Open Systems Dependability



System Architecture: DEOS Framework/Tools



DEOS Framework/Tools and DEOS Process



Elemental Technologies

- Virtual Machines
  - give the mechanism of isolation
  - monitor for security, failure prevention, and logging
- Policy Script/Fault Scenario
  - script languages
  - responsive action, cause analysis
- Program Verification
  - model-based verification
  - type-based verification
- Program Test and Benchmarking (DS-Bench/D-Cloud)
  - correctness and performance test
  - fault insertion test
- And many others

Standardization Activities

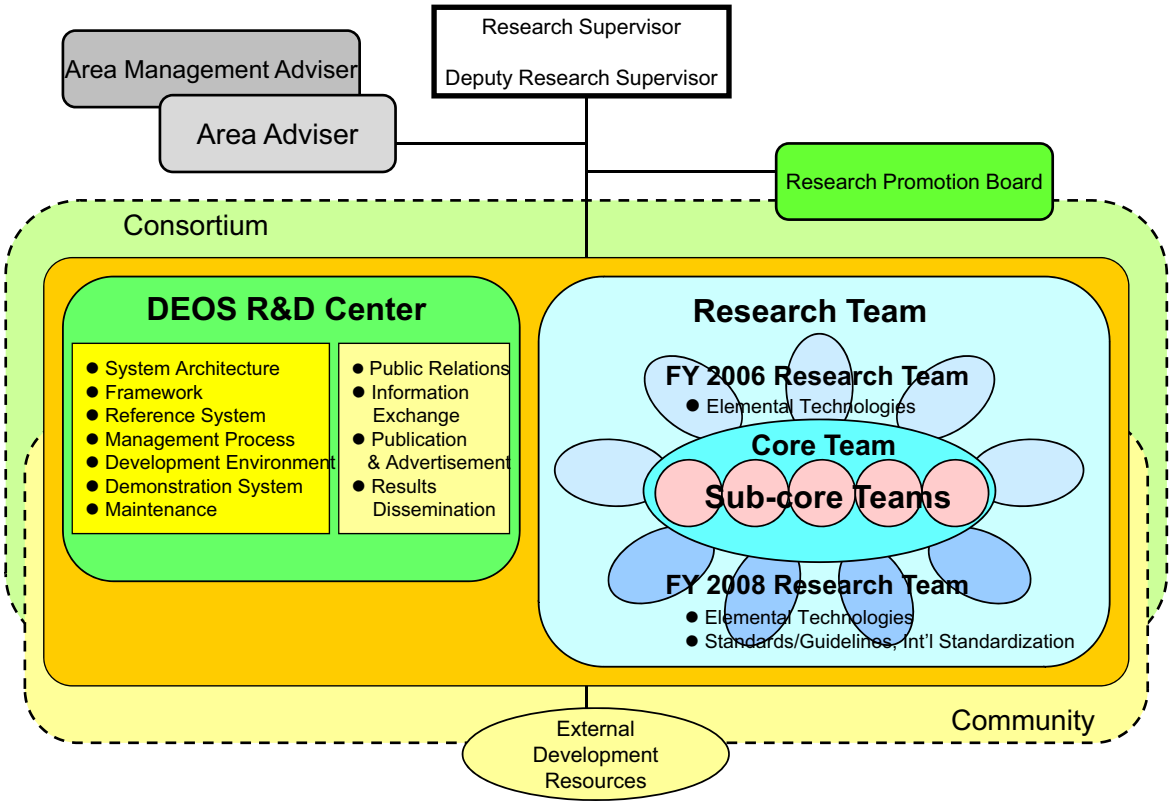
- Standardization of the DEOS Process
  - based on the notion of Open Systems Dependability
  - detailed specification of the DEOS Process
- Activities in
  - ISO/IEC JTC1/SC7/WG7 (Software and system engineering-Lifecycle management)
  - ISO/IEC 15026 for assurance case
  - Proposal of D-Case to OMG industrial standards

DEOS Project

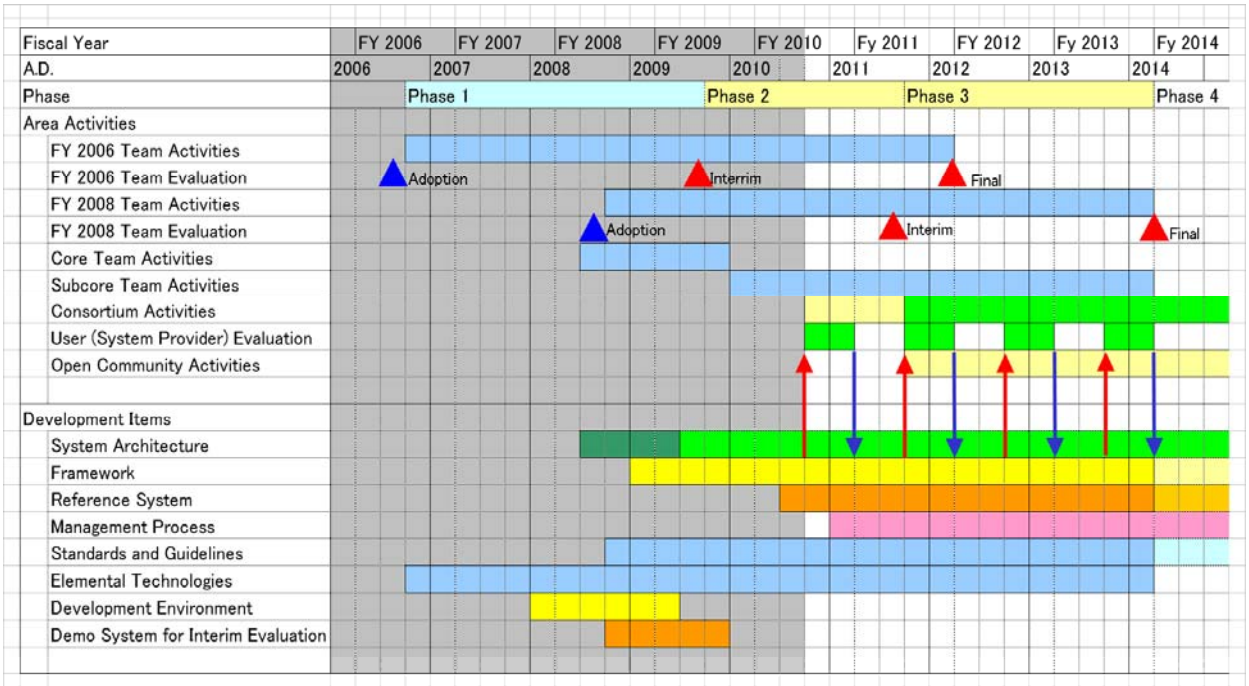
Dependable OS for Embedded Systems Aiming at Practical Applications

- A project under Japan Science and Technology Agency (JST)
- Roughly $50M in total over 7 years started in 2006
- 5 teams selected in 2006 and 4 teams in 2008
- To develop Dependable Embedded OS based on the notion of Open Systems Dependability
- R&D Center (DEOSC) was established in 2007 for supporting development, integrating technologies developed by the teams, and promoting the use.
- DEOS Consortium is planned to be established in 2011 to promote and distribute the DEOS Process and Systems.

DEOS Project Organization



DEOS Project Schedule



Summary

- A huge and complex system inherently has incident factors due to *incompleteness* and *uncertainty*
- We proposed a new approach called *Open Systems Dependability*
- Open Systems Dependability is achieved by the *DEOS Process*, supported by the *architecture* and *elemental technologies*
- Development of the DEOS Process and the OS will be released through *DEOS Consortium*

Thank you

JST/DEOS Project
http://www.jst.go.jp/kisoken/crest/en/category/area04-4.html

JST/DEOS Center
http://www.dependable-os.net/index-e.html

Sony Computer Science Laboratories, Inc.
http://www.sonycsl.co.jp