Keynote Speech at ICFEM 2012, Kyoto

Dependable Embedded Operating Systems • Dependability Engineering for Open Systems •
DEOS

Toward Practical Application of Formal Methods in Software Lifecycle Processes

November 14, 2012

Mario Tokoro
Research Supervisor, JST/CREST DEOS Project
Sony Computer Science Laboratories, Inc.

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

Background

- Large Information Systems are continuously used for a long period of time, while constantly being modified due to unexpected changes in, e.g.,
  - Service Objectives
  - Users’ Requirements
  - Evolving Technologies
  - Regulations and Standards

- These systems include externally-developed modules, and are often connected to external systems, and might run on unknown environments (clouds)

- The development and modification of a system is performed concurrently with the system’s operation, and it is almost impossible to view the life of the system as a temporal and spatial concatenation of static and isolated systems.

- Existing software lifecycle processes can hardly cope with these situations, whereas, achieving dependability in such an ever-changing system has become one of the most demanding system issues to be solved.

Background

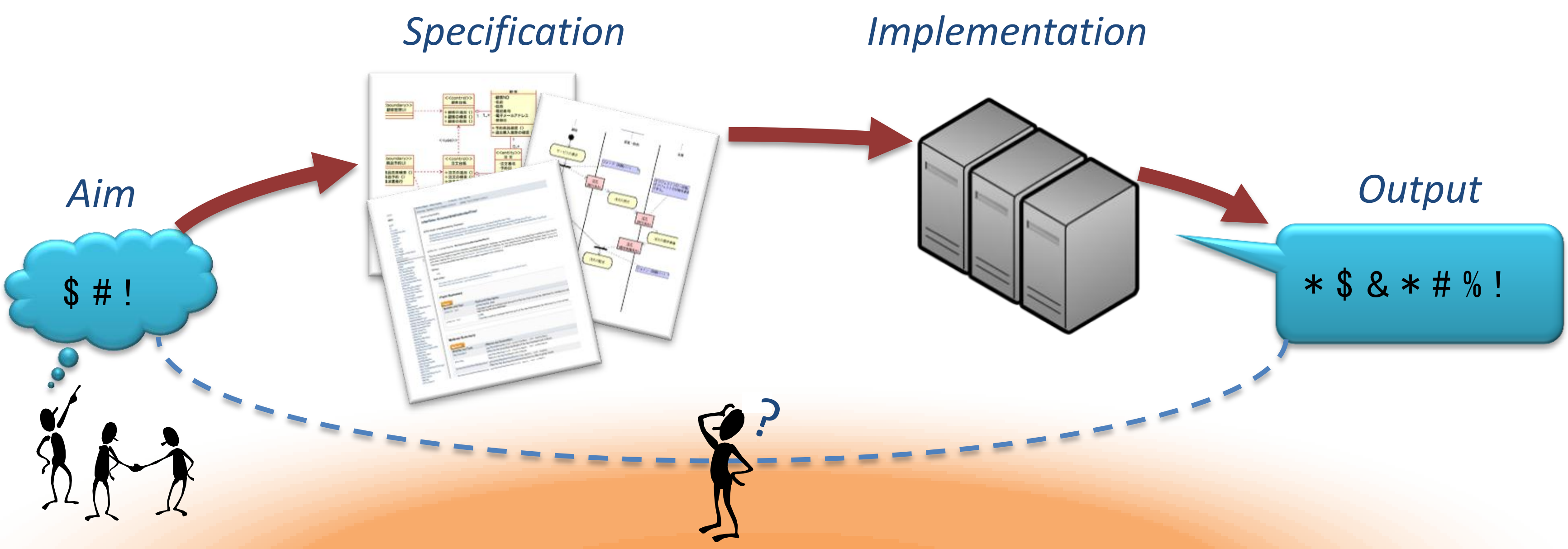
- Large Information Systems are continuously used for a long period of time, while constantly being modified due to unexpected changes in, e.g.,
  - Service Objectives
  - Users' Requirements
  - Evolving Technologies
  - Regulations and Standards
- These systems include externally-developed modules, and are often connected to external systems, and might run on unknown environments (clouds)
- The development and modification of a system is performed concurrently with the system's operation, and it is almost impossible to view the life of the system as a temporal and spatial concatenation of static and isolated systems.
- Existing software lifecycle processes can hardly cope with these situations, whereas, achieving dependability in such an ever-changing system has become one of the most demanding system issues to be solved.

Achieving Dependability

=

Coping with CHANGES

Consistency throughout Lifecycle is hard to be kept



Aim

$ # !

Specification

Implementation

Output

* $ & * # % !

?

Changing Environments

- Service objectives
- Users' requirements
- Available technologies
- Standards
- Regulations, etc…

Indeterminacy Problem

Our Approach

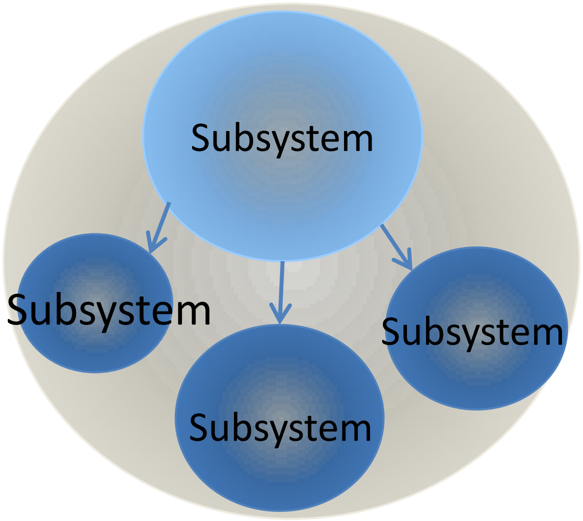
- An approach to ever-changing systems
  - the boundary
  - Functions
  - structures
  - interfaces
- We need to consider the Indeterminacy Problem
- Thus, we need to give up completeness

⇩

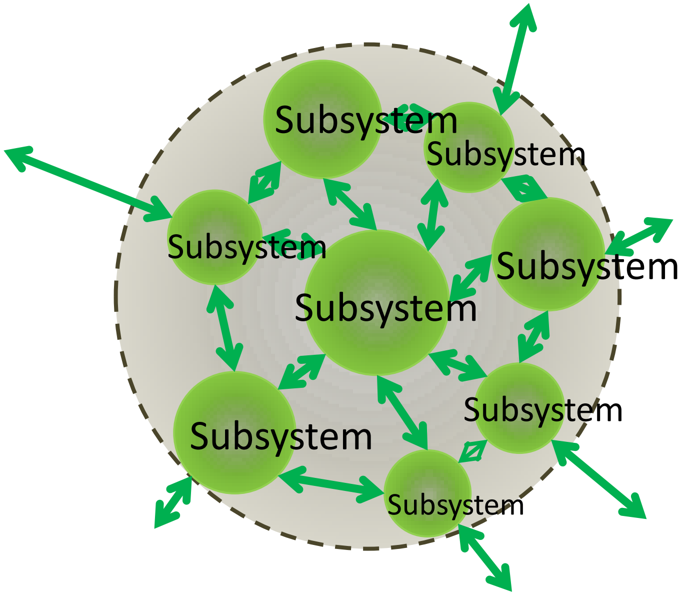
- We view such system as *Open Systems* (than Closed systems).

Closed Systems vs. Open Systems

Closed Systems



Open 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.

- 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.

Open Systems Dependability

- A system whose function, structure and boundary keep changing over time is called an open system in contrast to a closed system whose function, structure, and boundary stay the same through the life of the system.

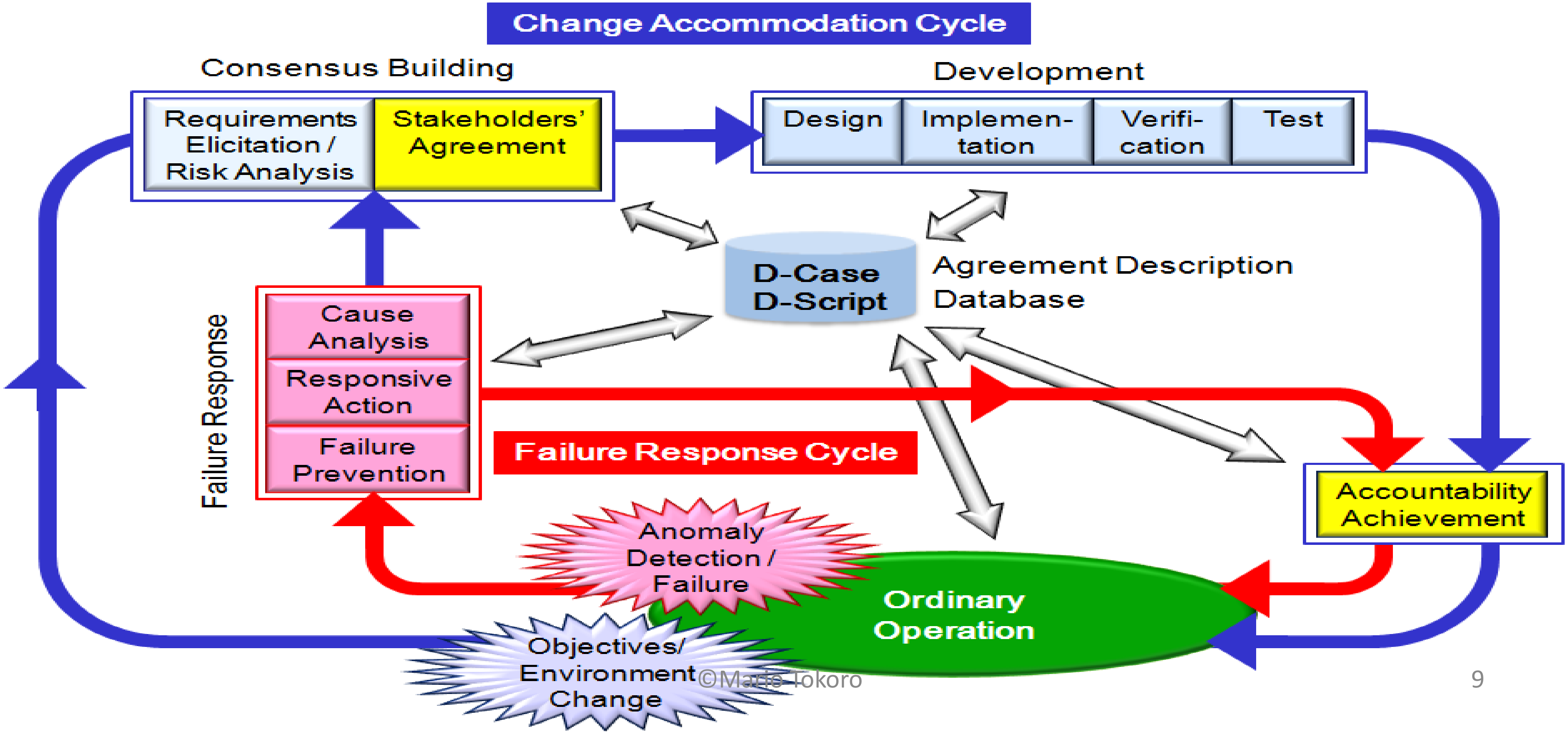
- We define Open Systems Dependability as the property of a system such that it has the ability
  - to continuously remove problem factors which may cause failures,
  - to take a quick and appropriate action when a failure occurs to minimize damage,
  - to safely and continuously provide the services expected by users as much as possible, and
  - to maintain accountability for the system operations and processes.

DEOS Process

- DEOS process implements Open Systems Dependability
- DEOS process treats the initial development, the modification of a system, and system operation as an integrated iterative lifecycle process.
- It includes
  1. *Change Accommodation Cycle* to accommodate requirement changes in service objectives and environments
  2. *Failure Response Cycle* to respond quickly and properly to failures
  3. D-Case, which is an extension of Assurance Cases, for stakeholders to achieve consensus on dependability issues, and
  4. The DEOS architecture which provides a database called D-ADD to retain D-Cases and an application-oriented runtime environment for flexible monitoring and control functions.

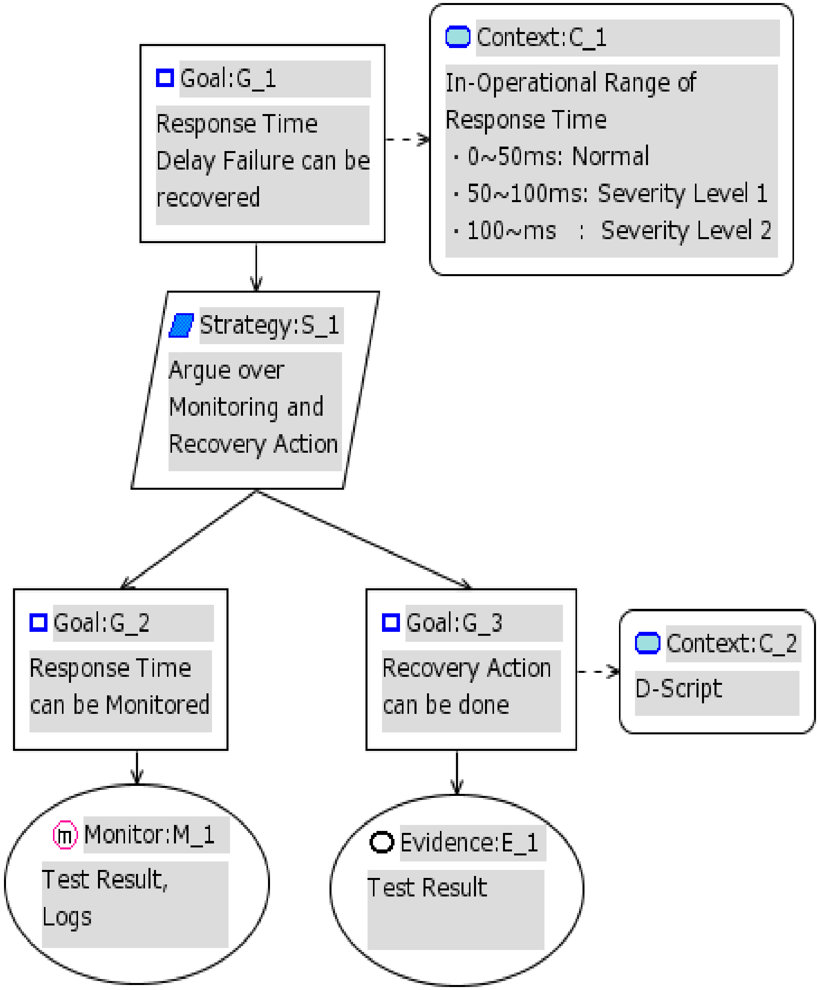
The DEOS Process

- Iterative process
  - Change Accommodation Cycle to accommodate requirement changes in service objectives and environments
  - Failure Response Cycle to respond quickly and properly to failures
- Agreement Description Database (D-ADD) including D-Case plays the key roles of consensus building and of integration of development and operation phases



Achieving Stakeholders' Agreement and Accountability through D-Case

- D-Case is an argumentation method/tool extended from assurance cases to be used in the development and operation phases
- We use the structural notation called GSN (Goal Structuring Notation) with Goal, Strategy, Context, Evidence, and Undeveloped Nodes
- We added Monitor Node to glue the development and operation phases
- We added External Node to incorporate externally developed modules and to use external services
- Description in Natural Language, Pseudo (Controlled) Natural Language such as SBVR, or more formal way in Agda
- Will be used to determine the level of dependability (like SIL or ASIL)



Goal:G_1
Response Time Delay Failure can be recovered

Context:C_1
In-Operational Range of Response Time
- 0~50ms: Normal
- 50~100ms: Severity Level 1
- 100~ms : Severity Level 2

Strategy:S_1
Argue over Monitoring and Recovery Action

Goal:G_2
Response Time can be Monitored

Goal:G_3
Recovery Action can be done

Context:C_2
D-Script

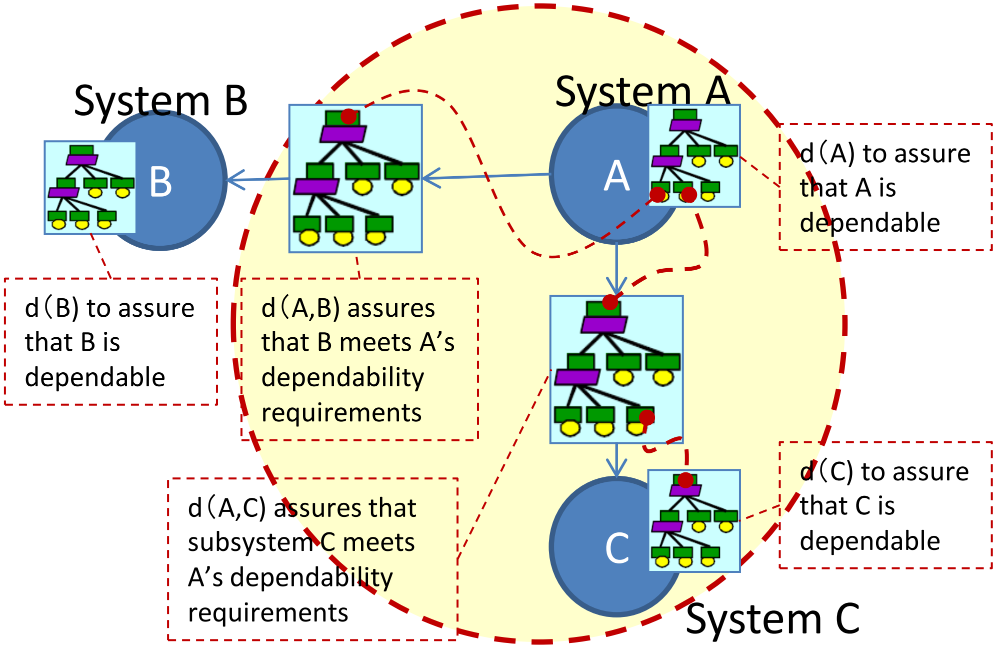
Monitor:M_1
Test Result, Logs

Evidence:E_1
Test Result

Coping with Externally-Developed Modules and Connection to External Systems

- Off-the-shelf modules
- Legacy codes
- External services through networks
- Systems may run in unknown environment such as clouds

- D-Case Reverse Engineering in addition to Forward Engineering



d (B) to assure that B is dependable

d (A,B) assures that B meets A's dependability requirements

d (A) to assure that A is dependable

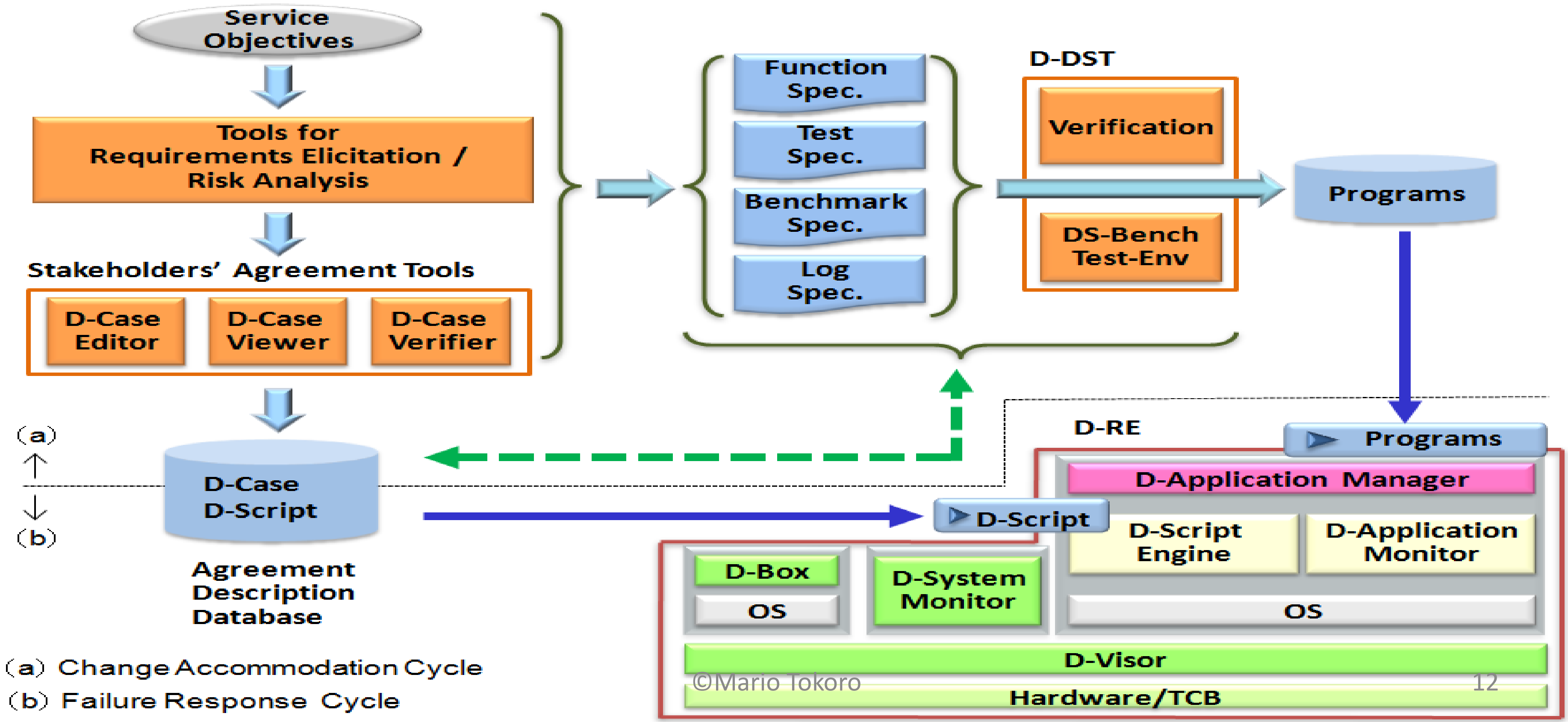
d (A,C) assures that subsystem C meets A's dependability requirements

d (C) to assure that C is dependable

DEOS Architecture

A DEOS architecture supports the execution of the DEOS process

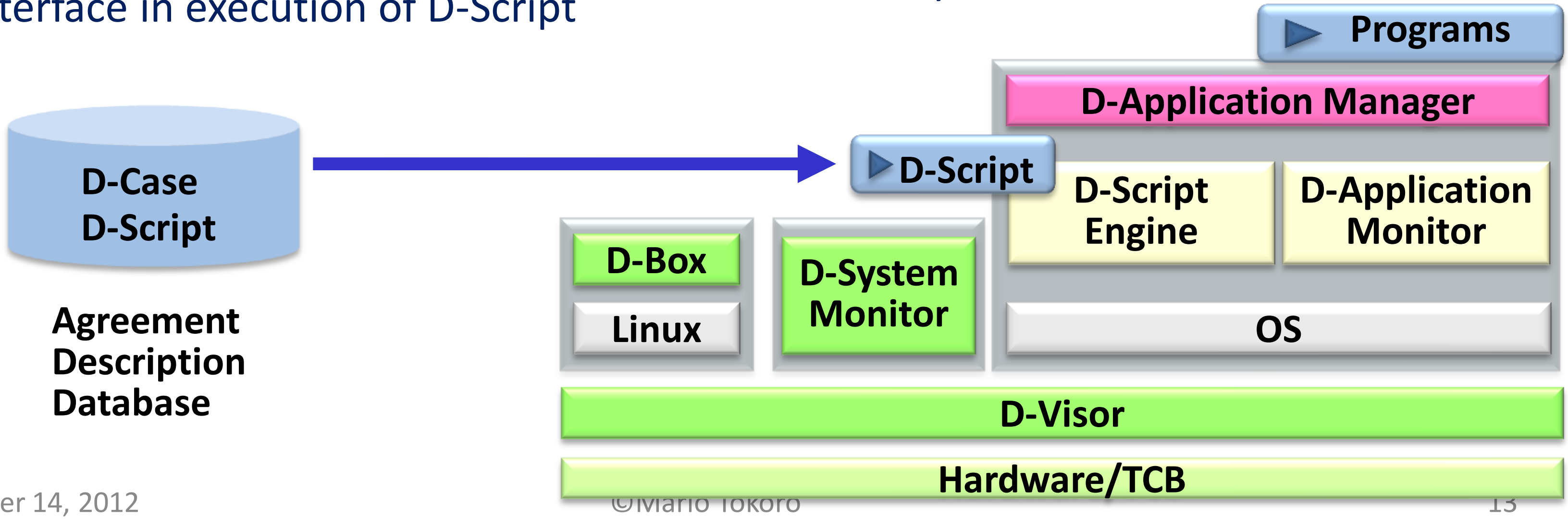
- Agreement Description Database (D-ADD) which retains all the D-Case descriptions,
- Tools to support requirements management,
- Tools to develop dependable software (D-DST)
- Execution Environment to execute programs, to monitor and record the states of programs, and to respond to failures (D-RE)



Binding Development and Operation Phases

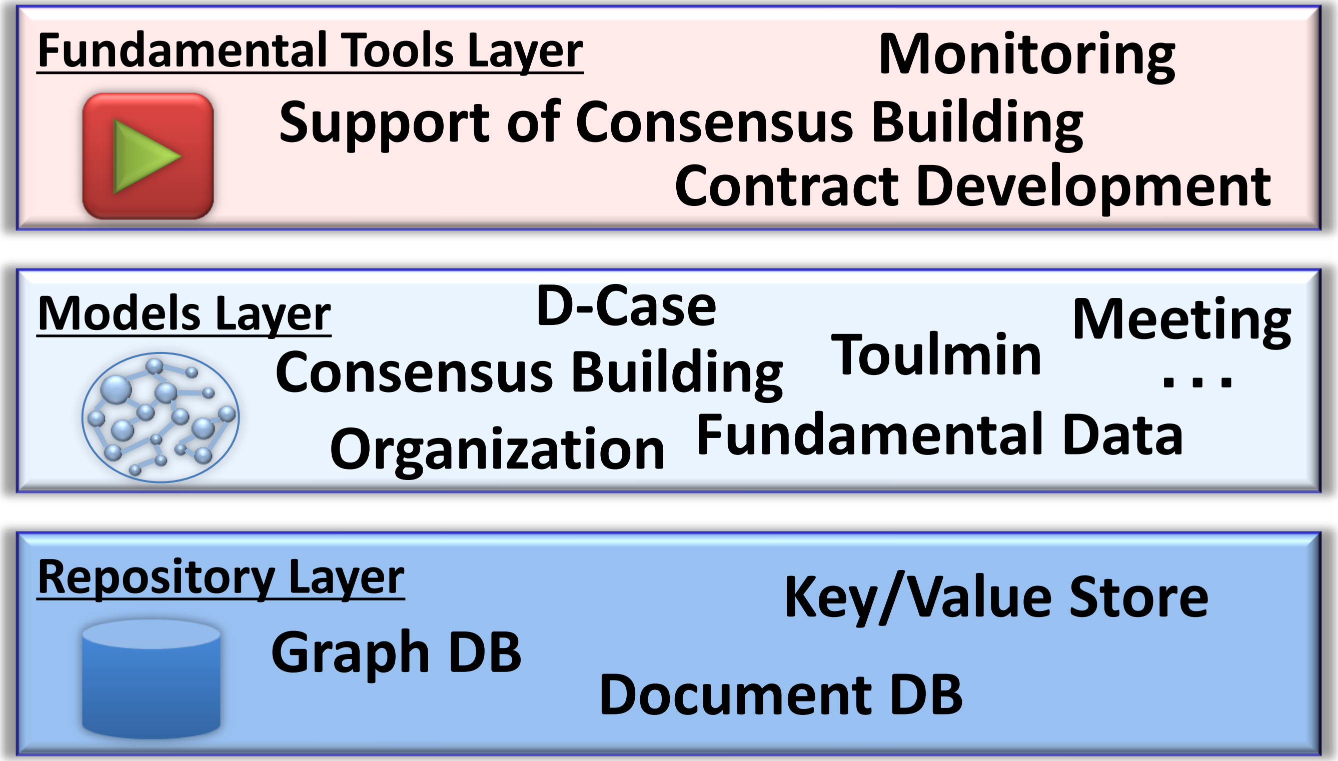
- Monitor Node in D-Case designates monitoring the operation and logging data
- D-Script describes responsive actions to be taken when operation shows a sign of failure or when operation fails
- D-Script Engine is designed to provide flexible yet secure man-machine interface in execution of D-Script

- D-ADD retains all the D-Case descriptions historically with the reasons why such decisions have been made
- D-ADD contributes, when a system is to be modified, to achieve stakeholders' agreement
- D-ADD contributes, when a system fails, to analyze causes of failures



D-ADD Consists of 3 Layers

- Fundamental Tools Layer which provide user interface tools
- Model Layer which associates agreement graph with D-Cases and evidences including documents and logged data
- Repository Layer which provides the storage



Fundamental Tools Layer

Monitoring

Support of Consensus Building

Contract Development

Models Layer

D-Case

Meeting

Consensus Building

Toulmin

. . .

Organization

Fundamental Data

Repository Layer

Key/Value Store

Graph DB

Document DB

How can we apply Formal Methods in Software Lifecycle Processes for Open Systems?

Formal D-Case for Rigorous V&V (1)

- Verification and Validation (Barry Boehm):
  - Verification : *Are we building the system right?*

    w.r.t. given, specified criteria: spec, operational conditions, …
  - Validation: *Are we building the right system?*

    w.r.t. “Real World” : user needs, actual environment, …
- Our thesis on D-Case:

  A formal D-Case ≈ ⟨ a formal theory , a formal proof in it ⟩

  Cf. Y. Kinoshita and M. Takeyama, keynote speech to be given in *Safety-critical Software Symposium* 2013.
  - **Formal theory** codifies the agreed vocabulary and reasoning principles about the system, environment and processes.
  - **Formal proof** represents the verification argument of the agreed **formal claim** that specifies what is “right”

  Proof assistants, such as **Agda**, support its construction/checking

Formal D-Case for Rigorous V&V (2)

- Rigor in V&V enabled by **formal D-Case**:

  Rigor in Verification: Communication through a formal D-Case

  Rigor in Validation: Requirements for a formal D-Case

  Requirements for a positive answer to "*Are we verifying the system right w.r.t. the current best practice?*"

- But beware! We can *never* say we built *a 100% right system*.

  ∵ its rightness is w.r.t. the real world.

  Validation is inherently a vaguely defined action that ideally is never ending (**open systems viewpoint**).

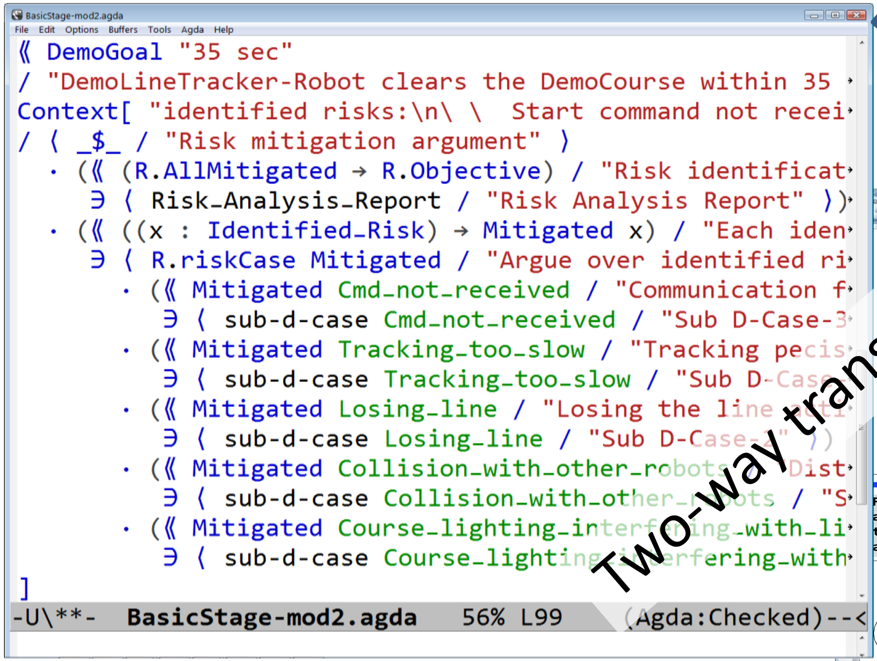
- Yet we can say something *definite* about the conformance to the requirements of best practices.

  Proof assistants, such as **Agda**, help here, too.

D-Case/Agda (“**D-Case in Agda**” Verification Tool)

supports checking/construction of formal D-Cases

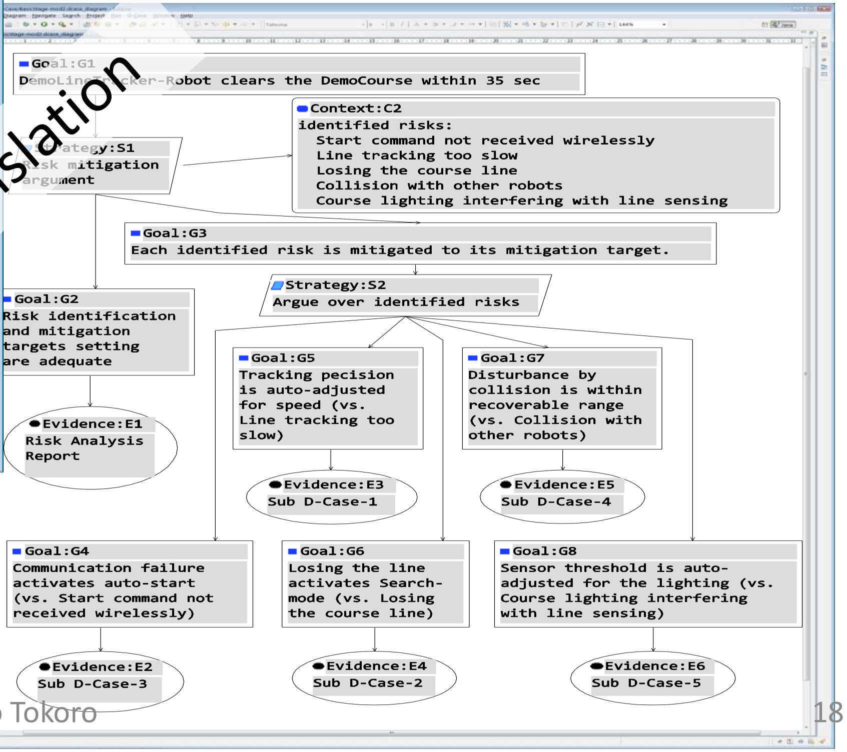
Checking, construction, generation of D-Cases
as formal proofs using Agda proof assistant



```
《 DemoGoal "35 sec"
/ "DemoLineTracker-Robot clears the DemoCourse within 35
Context[ "identified risks:\n\ \  Start command not recei
/ ⟨ _$_ / "Risk mitigation argument" ⟩
  • (《 (R.AllMitigated → R.Objective) / "Risk identificat
     ∋ ⟨ Risk_Analysis_Report / "Risk Analysis Report" ⟩)
  • (《 ((x : Identified_Risk) → Mitigated x) / "Each iden
     ∋ ⟨ R.riskCase Mitigated / "Argue over identified ri
       • (《 Mitigated Cmd_not_received / "Communication f
          ∋ ⟨ sub-d-case Cmd_not_received / "Sub D-Case-3
       • (《 Mitigated Tracking_too_slow / "Tracking pecis
          ∋ ⟨ sub-d-case Tracking_too_slow / "Sub D-Case-
       • (《 Mitigated Losing_line / "Losing the line acti
          ∋ ⟨ sub-d-case Losing_line / "Sub D-Case-2" ⟩)
       • (《 Mitigated Collision_with_other_robots / "Dist
          ∋ ⟨ sub-d-case Collision_with_other_robots / "S
       • (《 Mitigated Course_lighting_interfering_with_li
          ∋ ⟨ sub-d-case Course_lighting_interfering_with
]
-U\**-   BasicStage-mod2.agda    56% L99     (Agda:Checked)--<
```

Two-way translation

Graphical edit,
domain-expert review
using D-Case Editor



Goal:G1
DemoLineTracker-Robot clears the DemoCourse within 35 sec

Context:C2
identified risks:
Start command not received wirelessly
Line tracking too slow
Losing the course line
Collision with other robots
Course lighting interfering with line sensing

Strategy:S1
Risk mitigation argument

Goal:G3
Each identified risk is mitigated to its mitigation target.

Goal:G2
Risk identification and mitigation targets setting are adequate

Strategy:S2
Argue over identified risks

Evidence:E1
Risk Analysis Report

Goal:G5
Tracking pecision is auto-adjusted for speed (vs. Line tracking too slow)

Goal:G7
Disturbance by collision is within recoverable range (vs. Collision with other robots)

Evidence:E3
Sub D-Case-1

Evidence:E5
Sub D-Case-4

Goal:G4
Communication failure activates auto-start (vs. Start command not received wirelessly)

Goal:G6
Losing the line activates Search-mode (vs. Losing the course line)

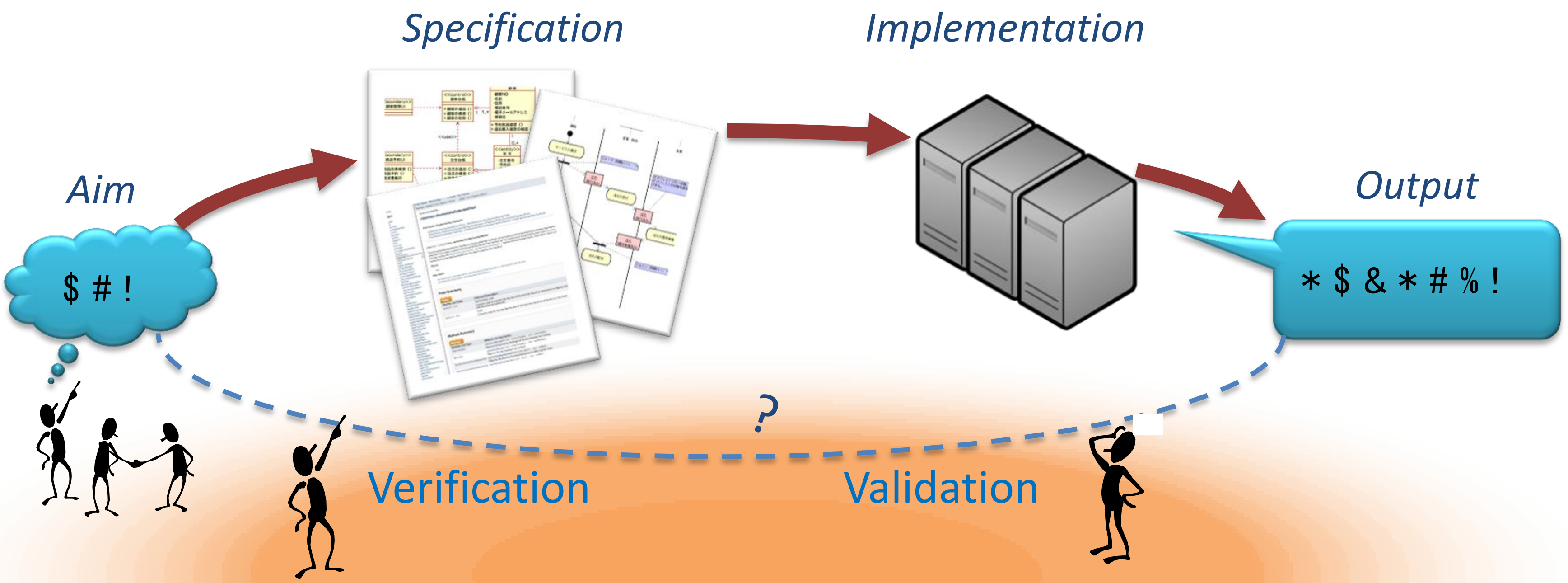
Goal:G8
Sensor threshold is auto-adjusted for the lighting (vs. Course lighting interfering with line sensing)

Evidence:E2
Sub D-Case-3

Evidence:E4
Sub D-Case-2

Evidence:E6
Sub D-Case-5

Applying Formal Methods to Open Systems



Aim

$ # !

Specification

Implementation

Output

* $ & * # % !

?

Verification

Validation

Repetitive Application of Formal D-Case to an open system for More Rigorous in Verification and in Validation

User Interfaces for Formal Methods are getting more important

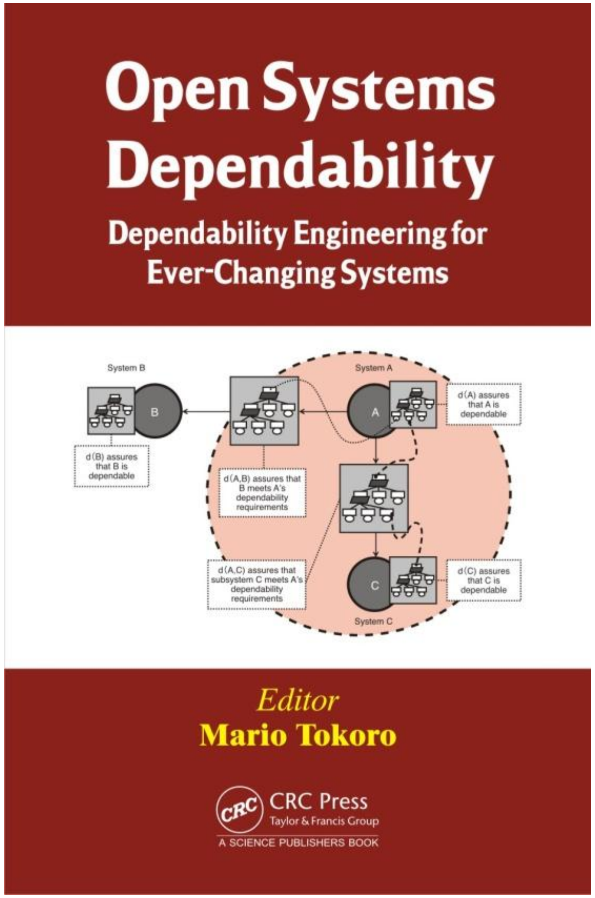
About the DEOS Project

Dependability Engineering for Open Systems

- A project under Japan Science and Technology Agency (JST)
- Roughly $60M in total over 7.5 years started in 2006
- 5 teams selected in 2006 and 4 teams in 2008
- 17 professors, 8 researchers from national laboratories, more than 40 post-docs, and many graduate students have worked together.
- R&D Center (DEOSC) was established in 2007 for supporting development, integrating codes developed by the teams, and promoting the use.

Current Status

- The DEOS process is defined
- Prototype Architectures have been demonstrated
- D-Cases are described for a few systems and more
- A book is published from CRC Press in October
- Other books are being written, e.g., on D-Case, D-ADD, D-Script…
- We are promoting international standardization (see next pages).



Open Systems Dependability

Dependability Engineering for Ever-Changing Systems

Editor

Mario Tokoro

CRC Press
Taylor & Francis Group
A SCIENCE PUBLISHERS BOOK

System B
B
d(B) assures that B is dependable
System A
A
d(A) assures that A is dependable
d(A,B) assures that B meets A's dependability requirements
d(A,C) assures that subsystem C meets A's dependability requirements
C
d(C) assures that C is dependable
System C

Standardization (1)

Purpose

- Sharing the concepts of Open Systems Dependability
- Provide guidelines for IT systems for social infrastructures
- Achieving common use of tools

We have been active in IEC and ISO standardization efforts

- IEC TC56 (Dependability)
  - The concept of Open Systems Dependability was submitted as NWIP to IEC TC56 in September 2012
  - Participating as experts: IEC60300-1(Dependability management), IEC62741(Dependability case), and IEC62628(Guidance on software aspects of dependability)
- ISO/IEC JTC1/SC7 (Software and systems engineering)
  - Standards for methodology of consensus building, achieving accountability and process
  - ISO/IEC15026 Systems and Software Assurance (Co-editor)

Standardization (2)

Purpose

- To contribute to Users

We have been working with The Open Group:

- To contribute to TOGAF 9.1 and/or TOGAF Next Generation wrt Dependability
  - ✓ Importance of *Change Management* (the notion of Open Systems Dependability) for achieving Dependability
  - ✓ Necessity of Integrating *Development* and *Operation* into *a single iterative process* (the DEOS Process)
  - ✓ Importance of *Stakeholders' Agreement* and *Accountability Achievement* through an assurance case (D-Case with RTES "Dependability Through Assuredness") and its history
  - ✓ D-Case Tools, D-ADD Implementations, etc.

We much appreciate your support and participation.

For more information, please send e-mail to
mario.tokoro@csl.sony.co.jp

Thank you

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

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

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