Tutorial

F1

Evaluating Dependability Attributes of Component-Based Specifications

Ivica Crnkovic and Lars Grunske

Day: Sunday 20 May 2007, Full Day Tutorial

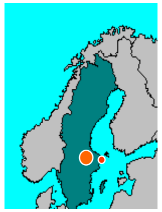
Venue: Ramsey

Evaluating Dependability Attributes of Component-Based Specifications

Ivica Crnkovic, Mälardalen University
Department of Computer Science and Electronics
Box 883, 721 23 Västerås, Sweden
http://www.idt.mdh.se/~icc, ivica.crnkovic@mdh.se

Lars Grunske, School of ITEE
ARC Centre for Complex Systems, University of Queensland
4072 Brisbane (St.Lucia), Australia
http://www.itee.uq.edu.au/~grunske/, grunske@itee.uq.edu.au

Presenter Introduction: Ivica Crnkovic



Mälardalen University, Vasteras (Västerås)

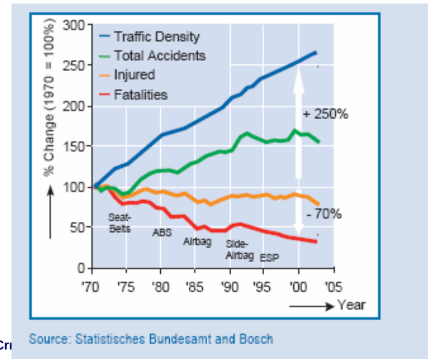
Prof. in Software Engineering
http://www.idt.mdh.se/~icc
ivica.crnkovic@mdh.se

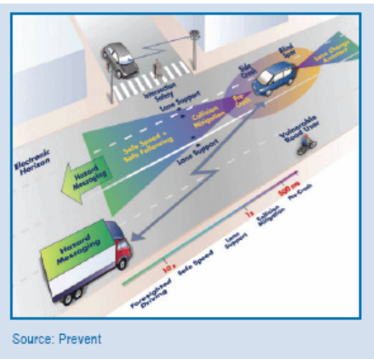
MÄLARDALEN UNIVERSITY

Presenter Introduction: Lars Grunske

Dr. rer. nat. Lars Grunske
Boeing Postdoctoral Research Fellow,
School of ITEE, University of Queensland,
4072 St Lucia, Brisbane, Australia
Webpage: http://www.itee.uq.edu.au/~grunske/

Dependable Systems





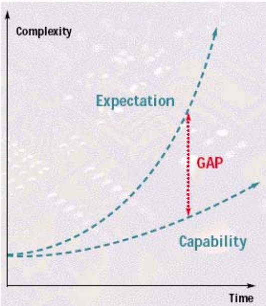
Outline of the Tutorial

1. Introduction
2. Basic concepts of dependable component-based systems and dependability
3. Overview of Component Models
4. Specification and composability of dependability properties
5. Overview of the State of the Art in Component-Based Dependability Evaluation Methods
6. Session Concluding remarks

Component-based software systems

Problems of software development

- The size & complexity of software increases rapidly
- Single products become part of product families
- Software is updated after deployment
- Demands of decreasing time to market
- Costs of software development increasing



Observations of the practice of software engineering

- About 80% of software development deals with changing (adaptation, improvement) of existing software
- Time to market is an important completive advantage:
  - Importance of incorporation of new innovations quickly
- System should be built to facilitate changes
  - Easy removal and addition of functionality
- Systems should be built to facilitate reuse
  - Easy integration of existing functions

Requirements:
- Provision of approach, technologies to facilitate
  Reuse, easy update and modification of software

Answer: Component-based Development

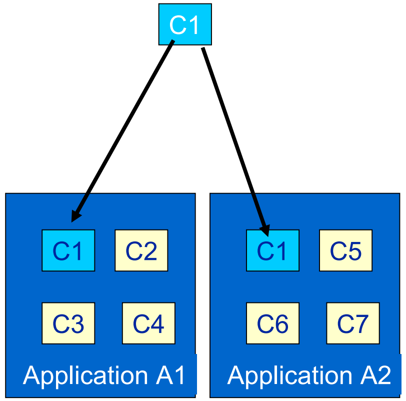
- Idea:
  - Separate development of components from development of systems
    - Build software systems from pre-existing components (like building cars from existing components)
    - Building components that can be reused in different applications

- **Component-based Software engineering** - supporting all aspects of activities in lifecyle of components and component-based systems

Main principles: (1) Reusability

- Reusing components in different systems

- The desire to reuse a component poses few technical constraints.
  - Similar systems architecture
  - Good documentation (component specification…)
  - a well-organized reuse process
  - ….



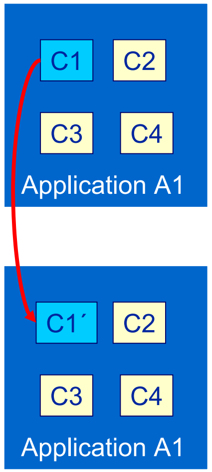
Main principles: (2) Substitutability

- Alternative implementations of a component may be used.

- The system should meet its requirements irrespective of which component is used.

- Substitution principles
  - Function level
  - Non-functional level

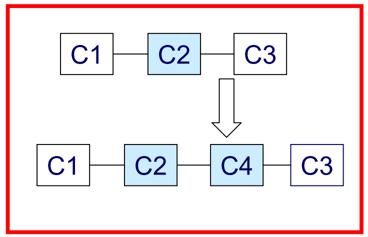
- Added technical challenges
  - Design-time: precise definition of interfaces & specification
  - Run-time: replacement mechanism

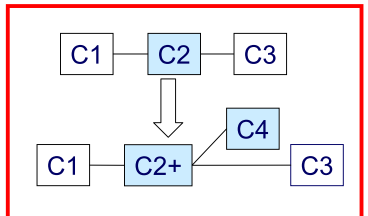


Main principles: (3) Extensibility

- Comes in two flavors:
  - Extending system functionality by adding components that are part of a system
  - Extending system functionality by increasing the functionality of individual components

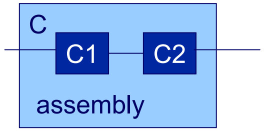
- Added technical challenges:
  - Design-time: extensible architecture
  - Run-time: mechanism for discovering new functionality





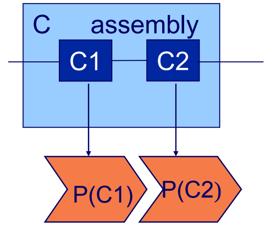
Main principles: (4) Composability

- Composition of components
  - P(c1 o c2) =P(c1) o P(c2) ??
  - Composition of functions
  - Composition of extra-functional properties
- Many challenges
  - How to reason about a system composed from components?
    - Different type of properties
    - Different principles of compositions



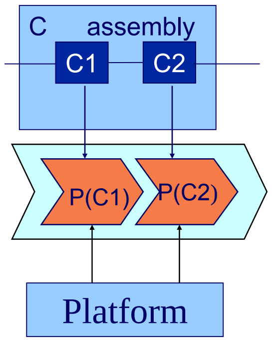
Compositional Reasoning

- Calculating properties of a system by combining properties of its constituents (components)
- Compositional reasoning: Function
  - If P(C) of program C is a function from input to output (pipe & filter) then the composition is modeled as a functional composition:
  - If $S = C_1 \text{ o } C_2$
    Then $P(S) = P(C_1) \text{ o } P(C_2)$



Predictable assembly

- Functional composition is not always possible
- Question with extra-functional properties
  - Example: dynamic memory usage M
  - If $S = C_1 \text{ o } C_2$
    then what is the composition $M(S) = M(C_1) \text{ o } M(C_2)$
- M is not defined only by properties $M(C_i)$, but also on properties of the platform “scheduling policy for example”
- Information supplied with C_1 is not enough



Predictable assembly = ability to predict properties of an assembly from properties of the involved components

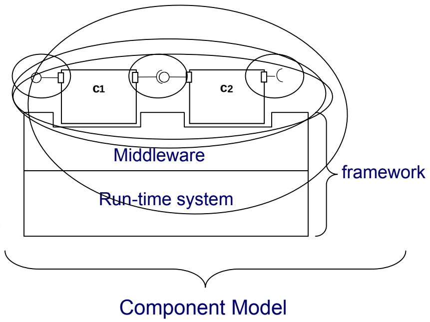
CBSE Terminology

To make the things easier we need first some definitions...

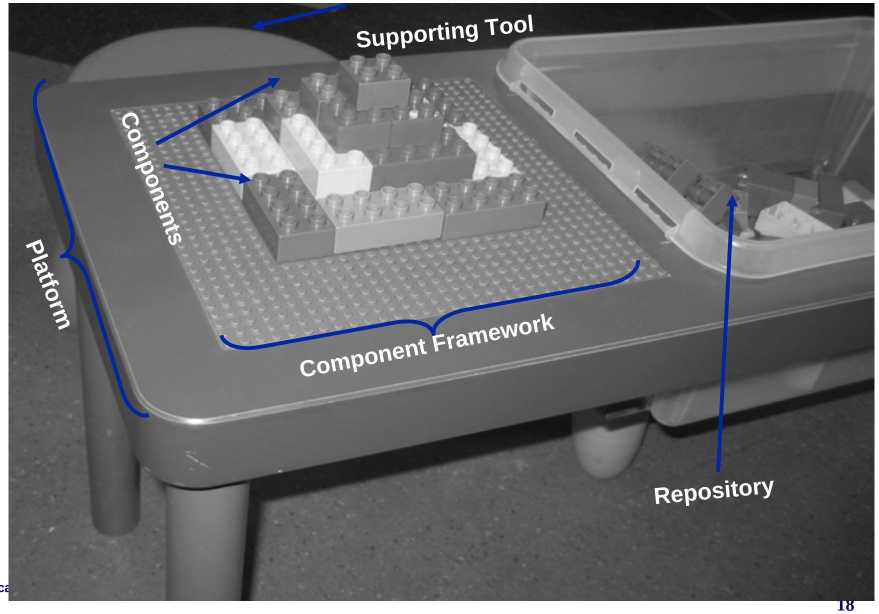
- Software Component
- Component-based systems
- Component specification
- Component composition
- Component and sytsems properties

Summary CBSE – basic definitions

- The basis is the **Component**
- Components can be **assembled** according to the rules specified by the **component model**
- Components are assembled through their **interfaces**
- A **Component Composition** is the process of assembling components to form an assembly, a larger component or an application
- Component are performing in the context of a **component framework**
- All parts conform to the **component model**
- A **component technology** is a concrete implementation of a component model



Component Technology



Software Component Definition (I)

Szyperski (Component Software beyond OO programming)

- A software component is
  - a unit of composition
  - with contractually specified interfaces
  - and explicit context dependencies only.
- A software component
  - can be deployed independently
  - it is subject to composition by third party.

Szyperski

Another definition

- A software component is a software element that
  - *confirms a component model*
  - can be independently deployed
  - composed without modification according to a composition standard.
- *A component model defines specific interaction and composition standards.*

G. Heineman, W. Councel, Component-based software engineering, putting the peaces together, Addoson Wesley, 2001

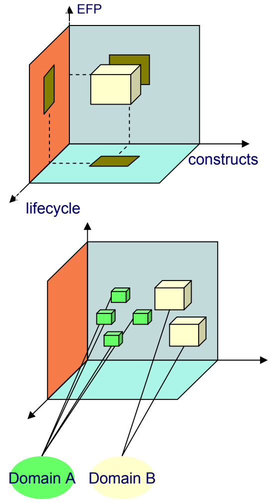
G. Heineman

Variety of component models

- The generalized definition allows different component models
  - In different domains there are different requirements and constraints
    - Different interactions (architectural styles)
    - Different extra-functional properties
    - Different integration and deployment policies

Component models classifications

- **Lifecycle.** *The lifecycle dimension identifies the support provided (explicitly or implicitly) by the component model, in certain points of a lifecycle of components or component-based systems*.
- **Constructs**. *The constructs dimension identifies (i) the component interface used for the interaction with other components and external environment, and (ii) the means of component binding and communication.*
- **Extra-Functional Properties**. *The extra-functional properties dimension identifies specifications and support that includes the provision of property values and means for their composition*.
- **Domains**. *This dimension shows in which application and business domains component models are used.*

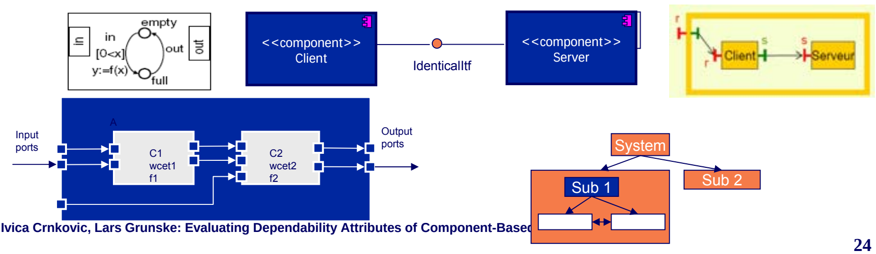


Classifications

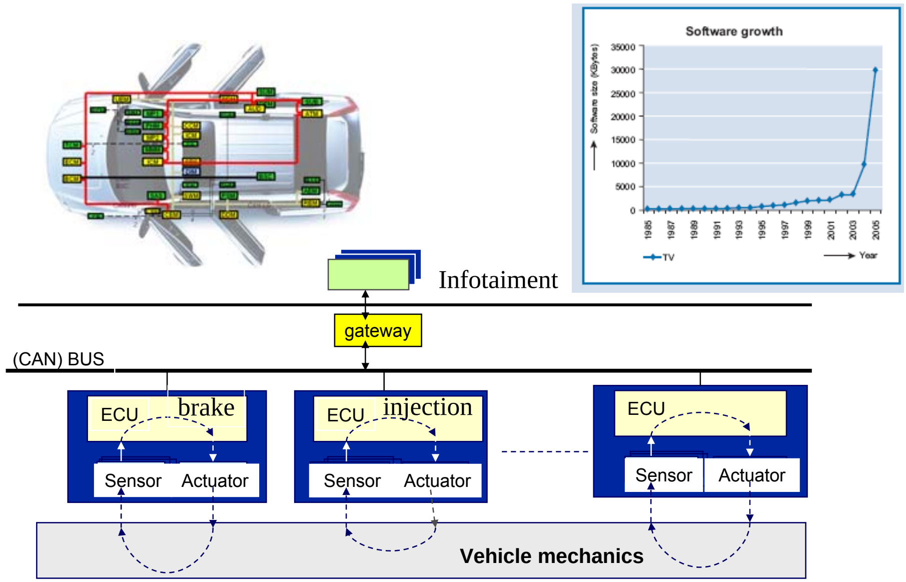
- **Lifecycle**
  - Modeling
  - Implementation
  - Packaging
  - Deployment
- **Constructs**
  - Interface types
  - Interface specification language
  - Interface Level (signature, contract-based, semantics)
  - Interaction
- **EFP**
  - General support for properties
  - Properties specification
  - Composition support
- **Domain**
  - Specific
  - General-purpose

Some of component models

- AUTOSAR
- BIP
- CCM
- Fractal
- KOALA
- EJB
- MS COM
- MS .NET
- OSGi
- PIN
- PECOS
- ROBOCOP
- RUBUS
- SaveCCM
- SOFA 2.0
- ….

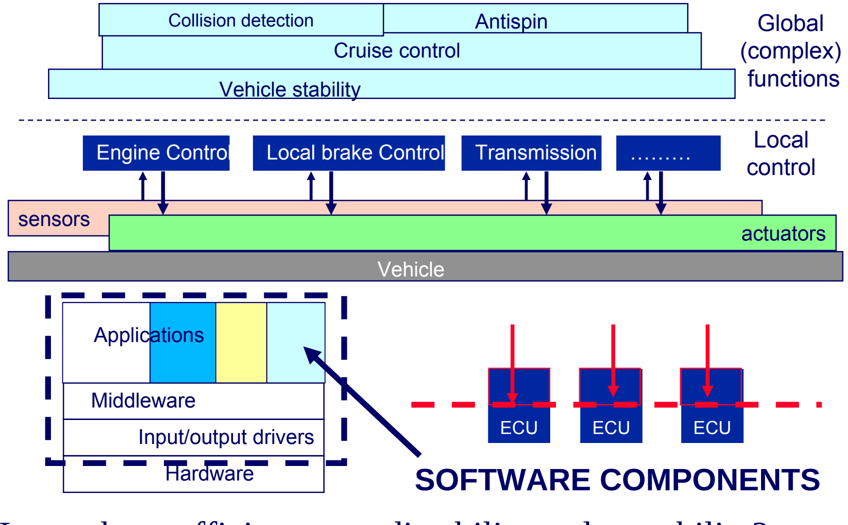


Example: Component-based embedded systems



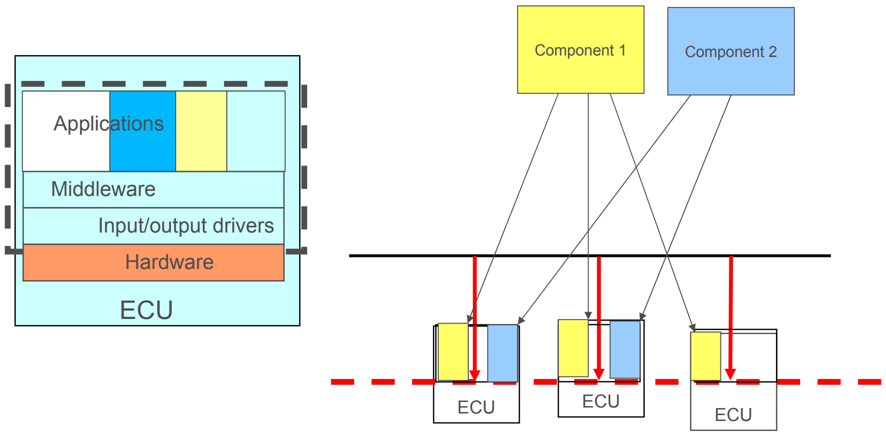
ECU – Electronic Control Unit

The architectural design challenge



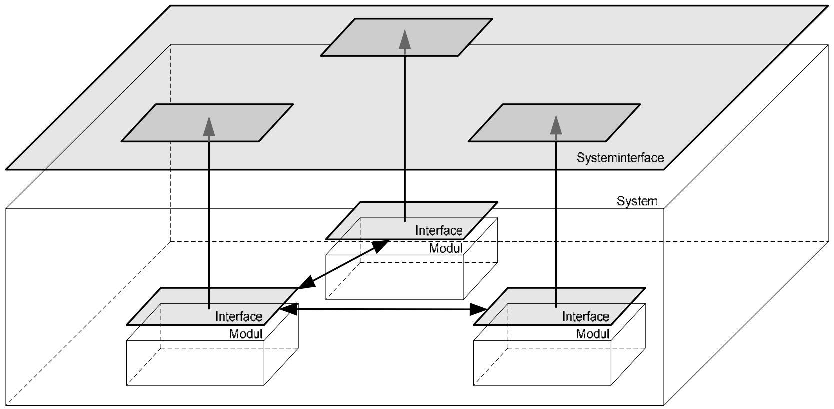
How to keep efficiency, predictability and reusability?

Distributed Software Components



Software Architecture and components

- Architecture Specification
  - Structure specification
  - Set of interface specification



Components and system properties

What are properties?

What are dependable systems?

Properties

- **Attribute/property**
  - "a construct whereby objects and individuals can be distinguished"
  - "a quality or trait belonging to an individual or thing"
    - A *required attribute/property* is expressed as a need or desire on an entity by some stakeholder.
    - An *exhibited attribute/property* is an attribute/property ascribed to an entity as a result of evaluating (for example measurement of) the entity.
- The need for properties is motivated by their explanatory roles they have to fill. They describe phenomena of interest – There are no "absolute" properties

Some example of properties

- Reusability, Configurability, Distributeability, Availability, Confidentiality, Integrity, Maintainability, Reliability, Safety, Security, Affordability, Accessibility, Administrability, Understandability, Generality, Operability, Simplicity, Mobility, Nomadicity, Hardware independence Software, independence, Accuracy, Footprint, Responsiveness, Scalability, Schedulability, Timeliness, CPU utilization, Latency, Transaction, Throughput, Concurrency, Efficiency, Flexibility, Changeability, Evolvability, Extensibility, Modifiability, Tailorability, Upgradeability, Expandability, Consistency, Adaptability, Composability, Interoperability, Openness, Heterogenity, Integrability, Audibility, Completeness, , Conciseness, Correctness, Testability, Traceability, Coherence, Analyzability, Modularity, ….

Kazman, R., L. Bass, G. Abowd, M. Webb,
"SAAM: A method for analyzing properties of software architectures,"
Proceedings of the 16th International Conference on Software Engineering, 1994.

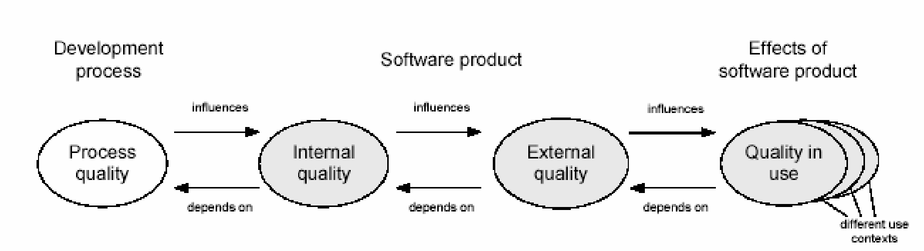
Kazman et al, Toward Deriving Software Architectures from Quality Attributes,
Technical Report CMU/SEI-94-TR-10, 1994.

McCall J., Richards P., Walters G., Factors in Software Quality, Vols I,II,III',
US Rome Air Development Center Reports, 1977.
Bosch, J., P. Molin, "Software Architecture Design: Evaluation and Transformation,"
Proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999.

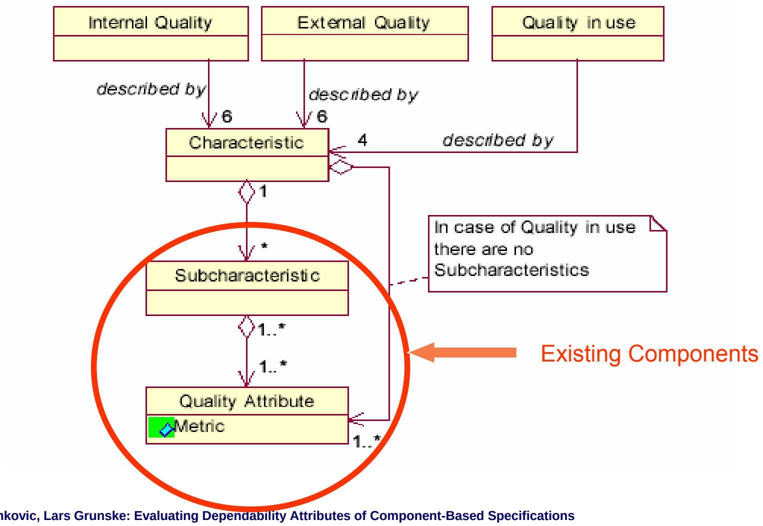
Classification of properties

- **Different classification**
  - Run-time properties
  - Life cycle properties
  - Run time
    - Reliability, safety, performance, robustness
  - Life cycle
    - Maintainability, portability, reusability,…
- **CBSE**
  - Component properties
  - System properties
    - Emerging properties

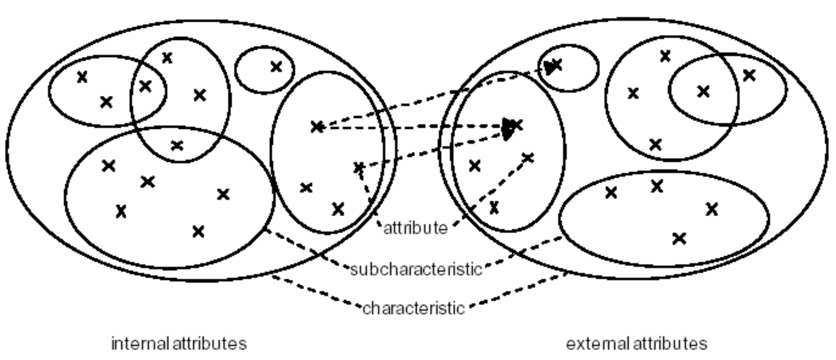
Quality model in ISO 9126-I



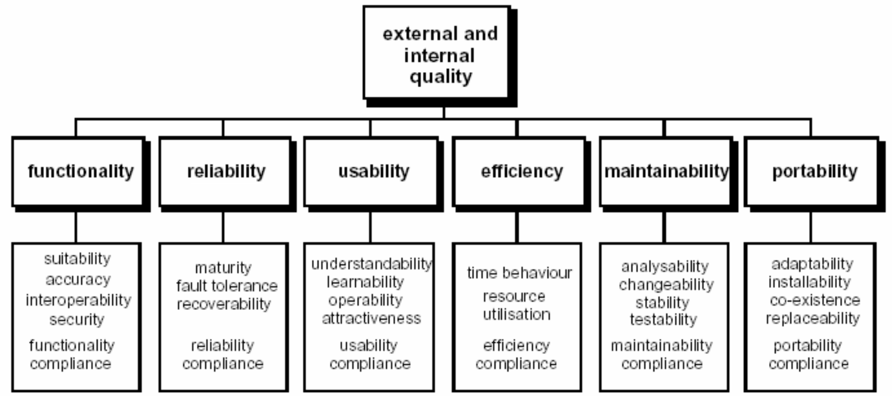
General Concepts of the ISO/IEC 9126-1



Quality characteristics, sub-characteristics and attributes



ISO/IEC 9126-1 quality attributes



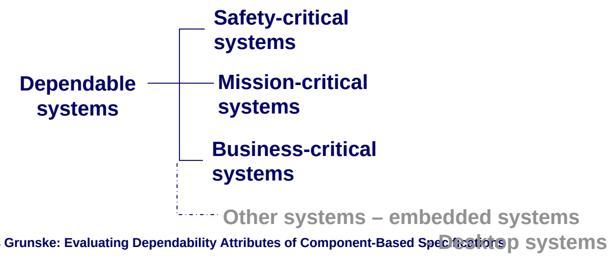
Other views – example: Dependability

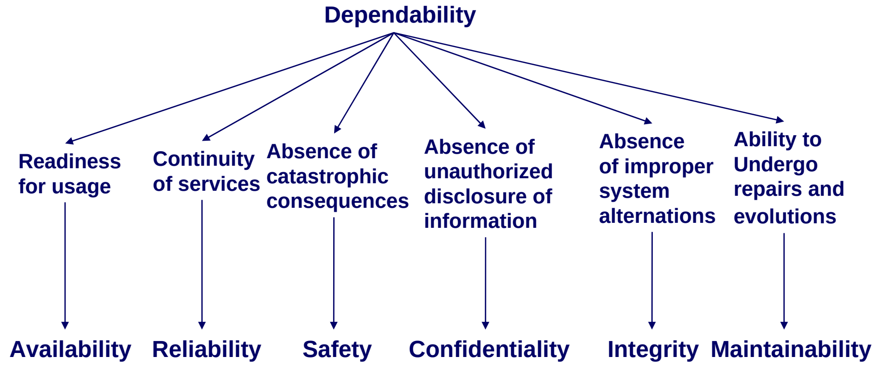
Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C., "*Basic concepts and taxonomy of dependable and secure computing*", IEEE Trans. Dependable Sec. Comput., Vol. 1, Issue 1, 2004

1. Ability of a system to deliver service that can justifiably be trusted
2. Ability of a system to avoid failures that are more frequent or more severe than is acceptable to user(s)

Related to

1. Trustworthiness (assurance that a system will perform as expected)
2. Survivability (capability to fulfill its mission in a timely manner)





Attributes of Dependability

Dependability Challenges

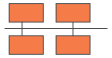
- **How can system quality attributes be accurately evaluated, from the specification of components properties which are determined with a certain (in)accuracy?**
- **Given the required system quality attributes, which properties are required from the components?**
- **To which extent, and under which constraints can the emerging system properties (i.e. the system properties non-existent on the component level) be derived from the component properties?**
- **Given a set of component properties, which system properties are predictable?**

Composition of properties

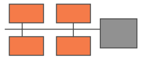
What do we need to know to predict system properties from component properties?

Given a set of component properties, which system properties are predictable?

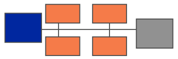
Component development (COTS type)
Known: Architectural Framework, component model
Unknown: system architecture, products, usage,..



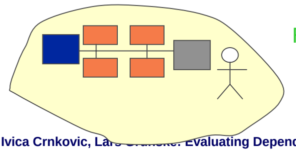
Product line
Known: domain, architectural framework, application skeleton, Variation (integration) points
Unknown: Final products



Open systems
Known: similar to PLA,
but integrators are not necessary known



Final product ready to use
(usage not necessary known)



Final product in use

What can we predict (or guarantee) about the system properties In each stage of development?

Properties Classification

1. ***Directly composable properties.*** A property of an assembly which is a function of, and only of the same property of the components involved.
2. ***Architecture-related properties.*** A property of an assembly which is a function of the same property of the components and of the software architecture.
3. ***Derived (emerging) properties.*** A property of an assembly which depends on several different properties of the components.
4. ***Usage-depended properties.*** A property of an assembly which is determined by its usage profile.
5. ***System context properties.*** A property which is determined by other properties and by the state of the system environment.

1. Definition: *A directly composable property of an assembly is a function of, and only of the same property of the components.*

$$P = \text{attribute}, \quad A = \text{assembly}, \quad c = \text{component}$$
$$A = \{c_i : 1 \le i \le n\}$$
$$P(A) = f(P(c_1), P(c_2), \dots, P(c_n))$$

- Consequence: to derive (predict) an assembly property it is not necessary to know anything about the system(s)

Example

- “Physical characteristics”
  - Static memory

$$M(A) = \sum_{i=1}^{n} M(c_i)$$
$$M = \text{memory size}, A = \text{assembly}, c_i = \text{components}$$

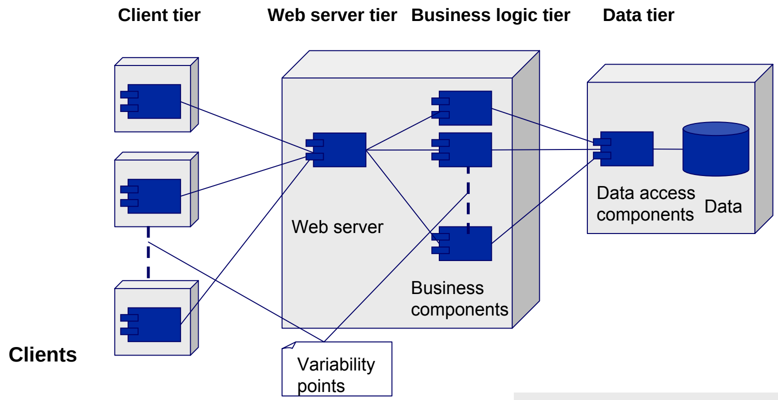
  - (the “function” can be much more complicated)
  - (the functions are determined by different factors, such as technologies)

2. Definition: *An architecture-related property of an assembly is a function of the same property of the components and of the software architecture.*

$$A = \{c_i : 1 \le i \le n\}$$
$$P(A) = f(P(c_1), P(c_2), \ldots, P(c_n), SA)$$
$$SA = \text{software architecture}$$

- *Consequence: System/assembly architecture must be known*
  - *Ok when building systems of particular class*
    - *(product-line architectures)*

Example (J2ee or .NET distributed systems)



Yan L., Gorton I., Liu A., and Chen S., "Evaluating the scalability of enterprise javabeans technology", In Proceedings of 9th Asia-Pacific Software Engineer-ing Conference, IEEE, 2002.

$$T/N = ax + b\frac{x}{y} + cy$$

T/N = execution time per transaction

x = number of clients; y = number of components

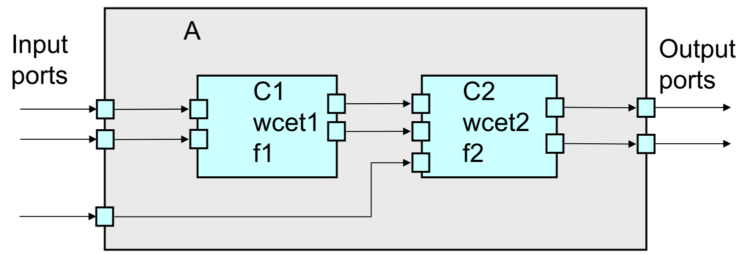
a, b, c = proportional factors for a particular implementation

3. Definition: *A derived property of an assembly is a property that depends on several different properties of the components*.

$$A = \{c_i : 1 \le i \le n\}$$
$$P(A) = f\begin{pmatrix} P_1(c_1), P_1(c_2), \cdots, P_1(c_n), \\ P_2(c_1), P_2(c_2), \cdots, P_2(c_n), \\ \vdots \\ P_k(c_1), P_k(c_2), \cdots, P_k(c_n) \end{pmatrix}$$
$$P = \text{assembly attribute}$$
$$P_1 \ldots P_k = \text{component attributes}$$

- Consequence: we must know different properties and their relations (might be quite complex)

Example



end-to-end deadline is a function of different component properties, such as **worst case execution time** (WCET) and **execution period.**

$$L^{n+1}(c_i) = c_i.wcet + B(c_i) + \sum_{\forall c_j \in hp(c_i)} \left\lceil \frac{L^n(c_i)}{c_j.T} \right\rceil c_j.wcet$$

4. Definition: *A Usage-dependent property of an assembly is a property which is determined by its usage profile.*

$$P(A,U_k) = f(P(c_i, U'_{i,k})): \ i,k \in N$$
$$P = \text{attribute for a particular usage profile}$$
$$U_k = \text{assembly usage profile}$$
$$U'_{i,k} = \text{component usage profile}$$

Consequence: It is not enough to know which system will be built. It must be known how the system will be used

Example: Reliability

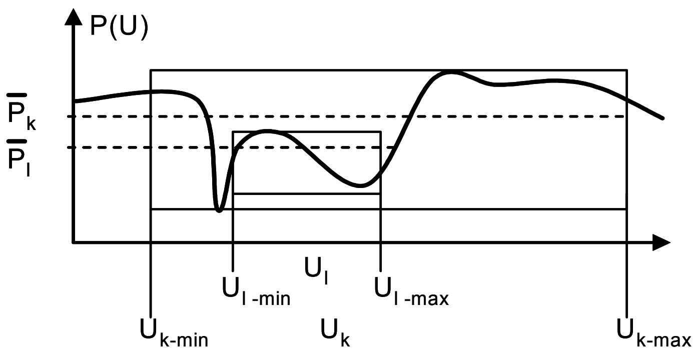
- the probability that a system will perform its intended function during a specified period of time under stated conditions.
- Mean time between failure
- How to calculate reliability for Software System?
  - Start from from a usage profile
  - Identify probability of the execution of components
  - Find out (measure) reliability of components
  - Calculate reliability of the system

Ralf H. Reussner, Heinz W. Schmidt, Iman H. Poernomo, Reliability prediction for component-based software architectures The Journal of Systems and Software 66 (2003) 241–252

Claes Wohlin, Per Runeson: Certification of Software Components,IEEE Trans. Software Eng. 20(6): 494-499 (1994)

Can we predict reliability using existing usage profiles?
Reuse problem:

mapping system usage profile to component usage profile
When the known (measured) properties values can be reused?



$$U_l \subseteq U_k \Rightarrow P_{k-\min}(A,U_k) \le P_l(A,U_l) \le P_{k-\max}(A,U_k)$$

5. Definition: *A System Environment Context property is a property which is determined by other properties and by context of the system environment.*

$$P_k(S,U_k,E_l) = f(P_k(c_i,U'_{i,k}),E_l); \quad i,k,l \in N$$
$$U_k = \text{System usage profile;}$$
$$E_l = \text{Environment context}$$
$$S = \text{System}$$
$$U'_{i,k} = \text{Component usage profile}$$

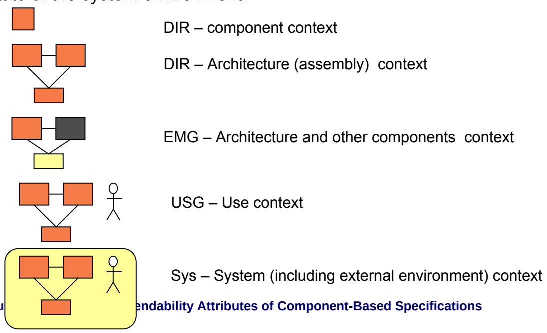
- **Consequence:** ***It is not sufficient to know the systems and their usage, it is necessary to know particular systems and the context in which they are being performed***

Example

- safety property
  - related to the potential catastrophe
  - the same property may have different degrees of safety even for the same usage profile.

Summary - Classification

1. ***(DIR) - Directly composable properties.*** A property of an assembly which is a function of, and only of the same property of the components involved.
2. ***(ART) - Architecture-related properties.*** A property of an assembly which is a function of the same property of the components and of the software architecture.
3. ***(EMG) - Derived (emerging) properties****.* A property of an assembly which depends on several different properties of the components.
4. ***(USG) - Usage-depended properties****.* A property of an assembly which is determined by its usage profile.
5. ***(SYS) - System context properties****.* A property which is determined by other properties and by the state of the system environment.



Conclusion

- Most of the emerging properties are impossible (or difficult) predict from pure composition reasoning
- Different analysis methods of the systems are applied

A General Framework for Model-Based Quality Evaluation of Component-Based Systems

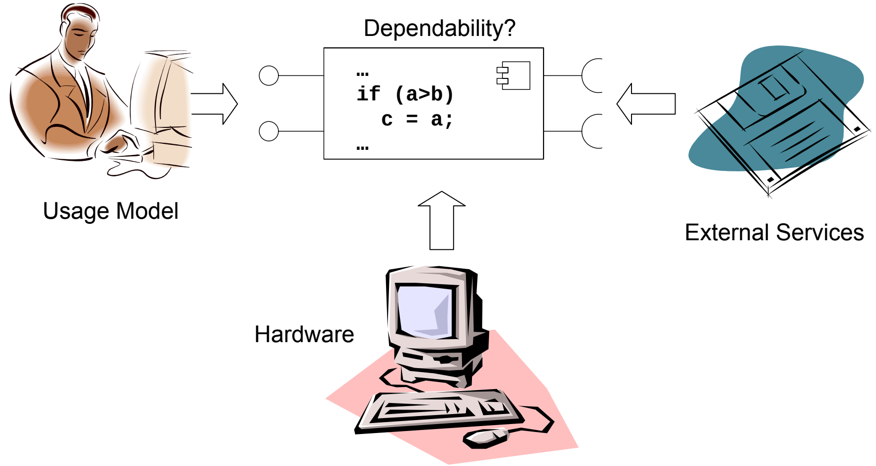
Encapsulated Evaluation Models
Operational Profiles
Composition Algorithms
Analysis Algorithms

A General Framework for Model-Based Quality Evaluation of CB Systems

- **Encapsulated Evaluation Models**
  - Independent from the deployment and the environment of a component
  - Similar to datasheets of electrical elements
  - Why?
    - Components are not self-contained and require external services
    - Components depend on the deployment environment
  - Examples:
    - WCET ← hardware platform
    - Reliability ← reliability of the external services
    - Performance ← frequency the environment calls services

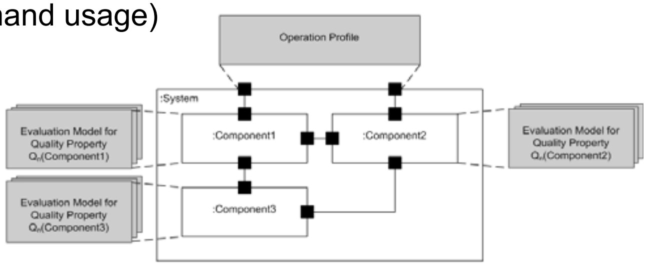
A General Framework for Model-Based Quality Evaluation

- **Motivation: Encapsulated Evaluation Models**



A General Framework for Model-Based Quality Evaluation of CB Systems

- **Operational Profile**
  - Operational/usage profile *OP* describes the usage of the component-based system
  - Example
    - Performance attributes depend on the number of requests per second from the system's users
    - Reliability depends on the operational mode (continuous vs. on demand usage)

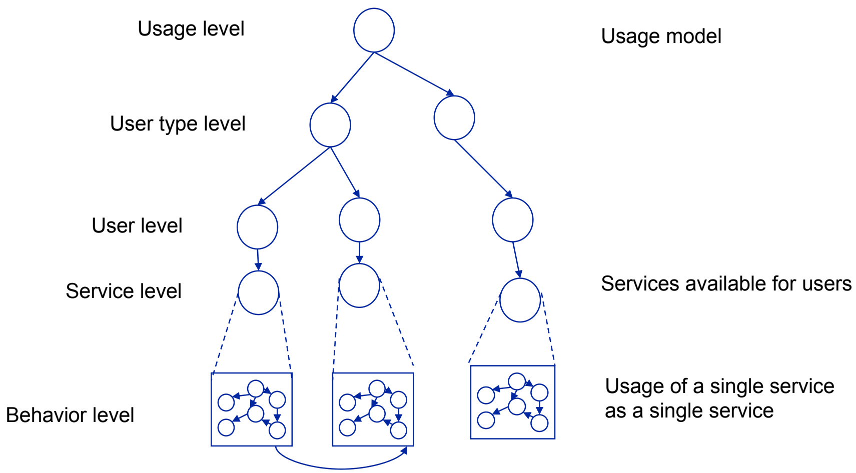


Operational Profile: Usage modeling and usage profile

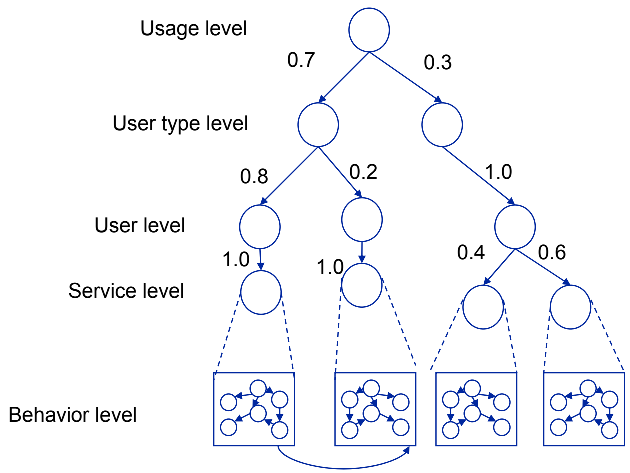
- Intended to model external view of the use of the component
- Component reuse – also reuse of usage model
- Use of Markov chains (FSM + probability of transition between states)
  - Problem – for complex systems Markov chains become very large

  - Attempt to solve the complexity by introduction of State Hierarchy Model [Claes Wohlin & Per Runesson 1994]

Operational Profile: State Hierarchy Model

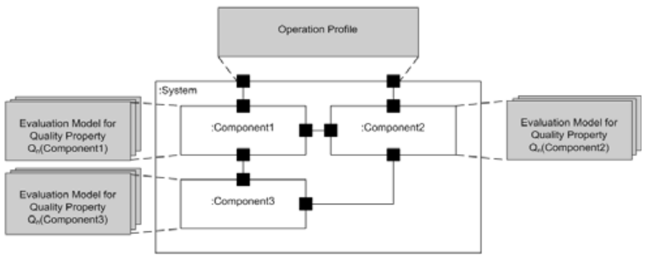


Operational Profile: Probabilities of Usage



A General Framework for Model-Based Quality Evaluation of CB Systems

- **Composition Algorithm**
  - Construction of a quality evaluation model for a hierarchical design specification
- **Analysis Algorithm**
  - “Extract” relevant measures of certain dependability attributes (eg. hazard probabilities)



Safety

*German :-)

Safety Terminology (1)

- **(Accident).** *An accident is an undesired event that causes loss or impairment of human life or health, material, environment or other goods*

- **(Hazard).** *A hazard is a state of a system and its environment in which the occurrence of an accident only depends on factors which are not under control of the system.*

Safety Terminology (2)

- **(Failure).** *A failure is any behavior of a component or system, which deviates from the specified behavior, although the environment conditions do not violate their specification.*
  - tl timing failure of a service (expected event or service is delivered after the defined deadline has expired - reaction too late)
  - te timing failure of a service (event or service is delivered before it was expected -reaction too early)
  - v incorrect result of requested service (wrong data or service result - value)
  - c accomplish an unexpected service (unexpected event or service - commission)
  - o unavailable service (no event or service is delivered when it is expected - omission)
- **(Fault).** *A fault is a state or constitution of a component that deviates from the specification and that can potentially lead to a failure.*

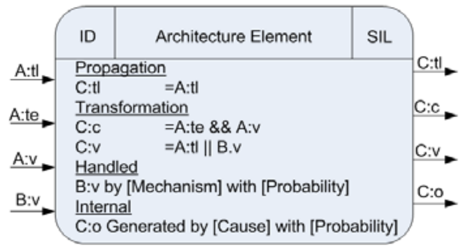
Safety Terminology (3)

- **(Risk).** *Risk is the severity combined with the probability of a hazard.*
- **(Acceptable Risk).** *Acceptable risk is the level of risk that has deliberately been defined to be supportable by the society, usually based on an agreed acceptance criterion*
  - ALARP
  - MEM
  - GAMAB

- **(Safety).** *Safety is freedom from unacceptable risks*
- **(Safety Requirements).** *A safety requirement is a (more or less formal) description of a hazard combined with the tolerable probability of this hazard.*
  - Hazard Spec. +THP/THR

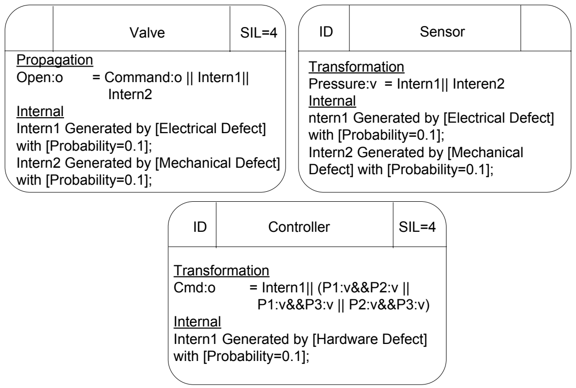
Failure Propagation and Transformation Notation (FPTN)

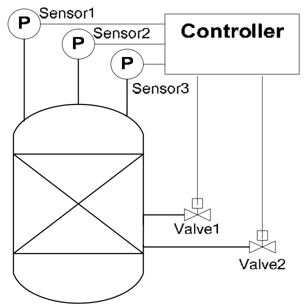
- Failure Propagation and Transformation Notation (FPTN)
  - Introduced by Fenelon, McDermid, Nicholson, Pumfrey
  - Benefits
    - Failure categorization (reaction too late(tl), reaction too early(te), value failure(v), commission(c) and omission(o))
    - First modular safety evaluation model
  - Weaknesses
    - No process support
    - No tool support
    - Event-based



Failure Propagation and Transformation Notation
(FPTN) Example

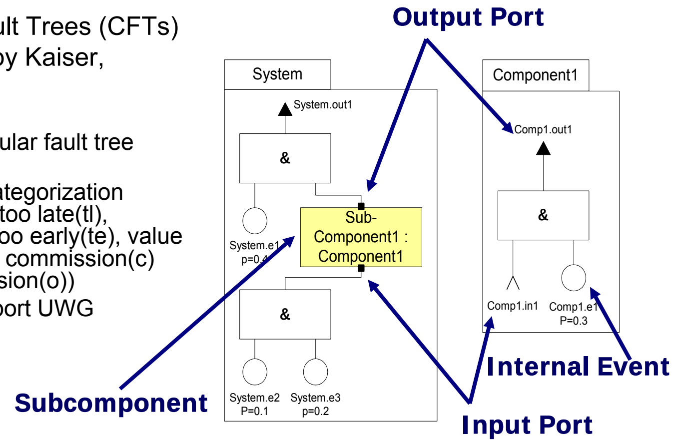
- Steam Boiler Example





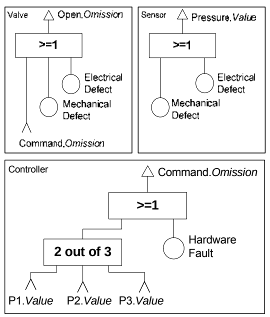
Component Fault Trees (CFTs)

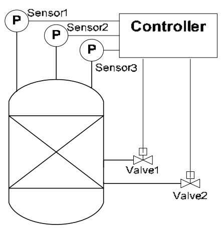
- Component Fault Trees (CFTs)
  - Introduced by Kaiser, Grunske
  - Benefits
    - First modular fault tree model
    - Failure categorization (reaction too late(tl), reaction too early(te), value failure(v), commission(c) and omission(o))
    - Tool support UWG

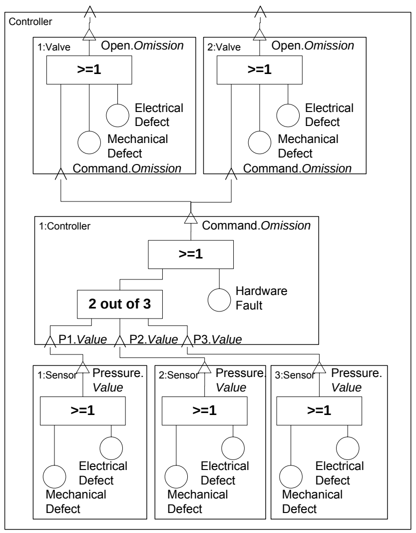


FT Component corresponds to Technical Component. Components have (Failure-)Ports.

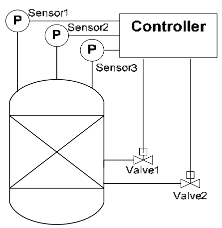
CFT Example

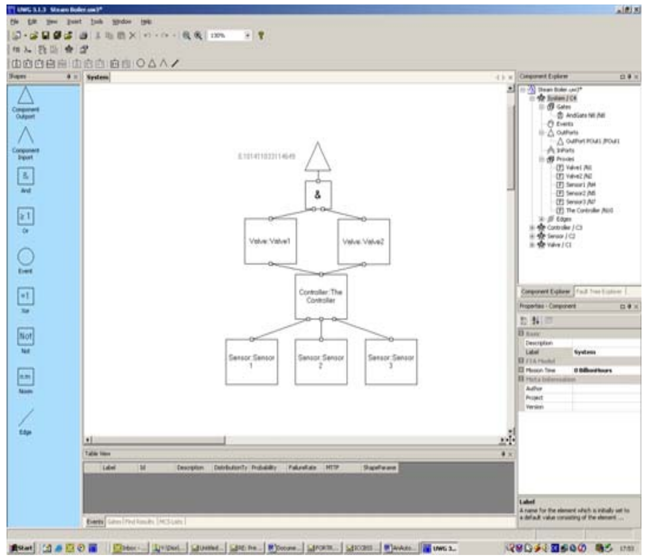






Analysis of the Top-Level CFT: The UWG3 Tool

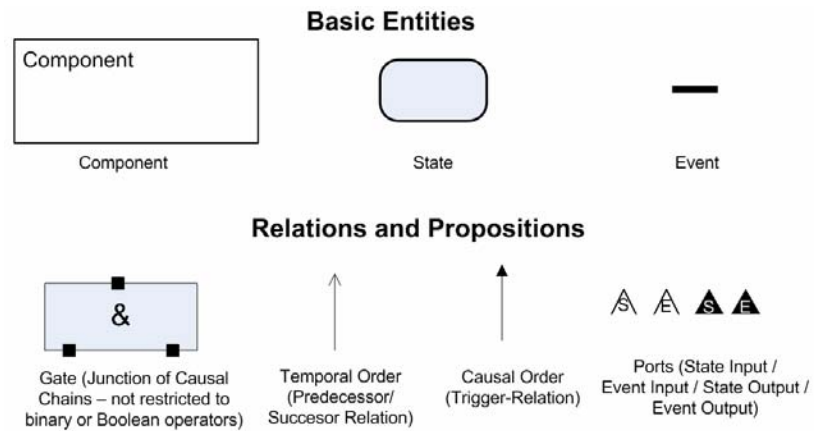




State Event Fault Trees (SEFT)

- State Event Fault Trees (SEFT)
  - Introduced by Kaiser, Gramlich, Grunske, Papadopoulos
  - Benefits
    - Automatic generation of system-level SEFT
    - State-event based semantic
    - Tool support (www.essarel.de)
  - Weaknesses
    - Complex Evaluation
    - For real world application only simulation-based results achievable

State Event Fault Trees - Syntax

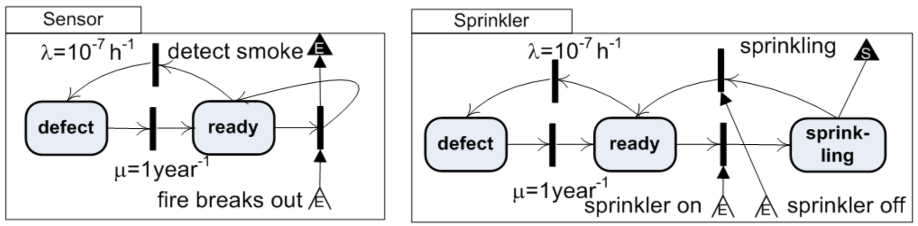


State Event Fault Trees Semantics/ Tool Support

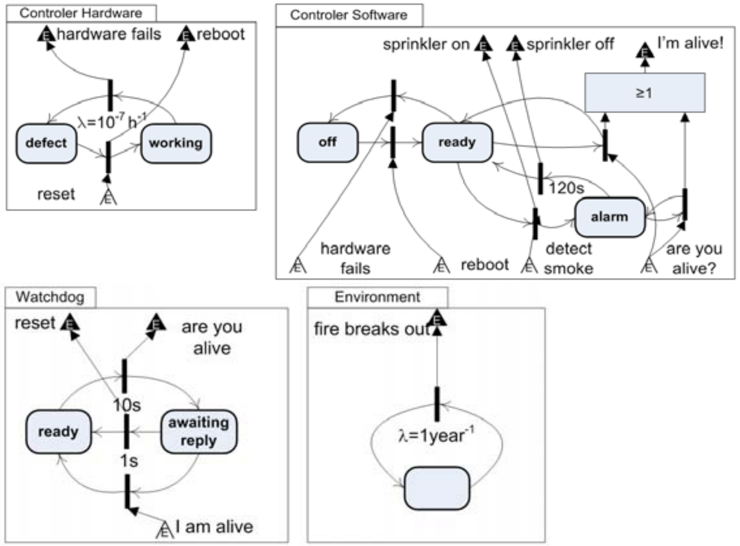
- Semantics (transformational)
  - Deterministic and Stochastic Petri Nets (DSPNs)
  - Used also for probability evaluation
- Tool Support
  - ESSaRel (Embedded Systems Safety and Reliability Analyser) Project www.essarel.de
  - Translation to DSPNs
  - Analysing via TimeNET 3.0 http://pdv.cs.tu-berlin.de/~timenet/
- Model-based safety evaluation
  - Based on HiP-HOPS and CFT safety evaluation process
    - Generation and Connection of SEFTs

State Event Fault Trees Example (1)

- Fire alarm system
  - Controller unit (hardware +software), smoke sensor, sprinkler, watchdog

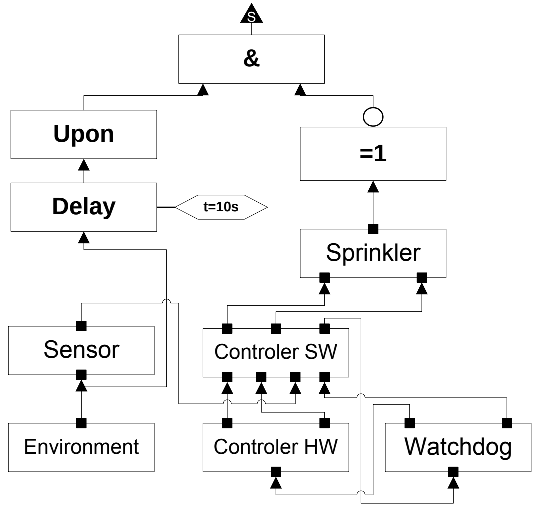


State Event Fault Trees Example (2)



State Event Fault Trees Example (3)

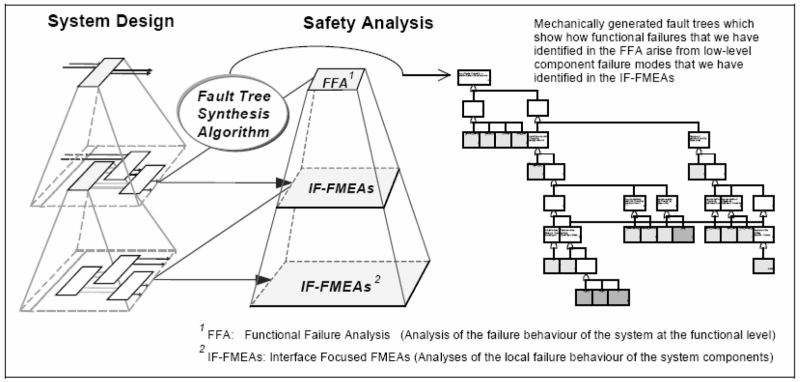
- Hazard Description
  - Fire breaks out and the sprinkler is not turned on within 10s



HiPHOPS

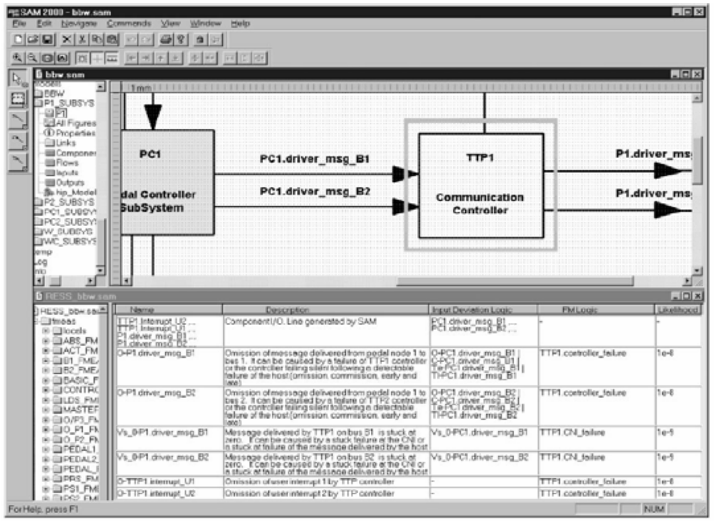
- Tabular Failure Annotations and HIP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies)
  - Introduced by Papadopoulos and McDermid in cooperation with Daimler Chrysler
  - Benefits
    - Automatic generation of system-level fault trees
    - Automatic generation of FMEA tables
    - Tool support/ Matlab Simulink
  - Weaknesses
    - Tabular failure annotations
    - Event-based

HIPHOPS Example (1)



From *Papadopoulos Y.*, McDermid J. A., Sasse R., Heiner G., Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure, Int. J. of Reliability Engineering and System Safety, 71(3):229-247, Elsevier Science, 2001.

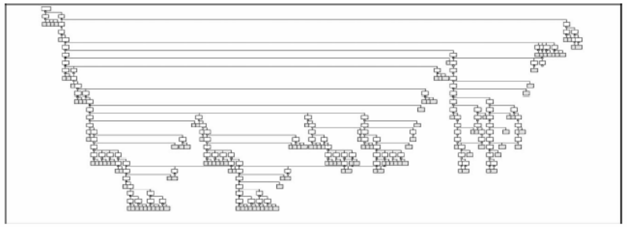
HIPHOPS Example (2)



From *Papadopoulos Y.*, McDermid J. A., Sasse R., Heiner G., Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure, Int. J. of Reliability Engineering and System Safety, 71(3):229-247, Elsevier Science, 2001.

HIPHOPS Example (3)

- Generation of traditional fault trees
  - Fault Tree+



From *Papadopoulos Y.*, McDermid J. A., Sasse R., Heiner G., Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure, Int. J. of Reliability Engineering and System Safety, 71(3):229-247, Elsevier Science, 2001.

Safety Evaluation Techniques & Generic Framework

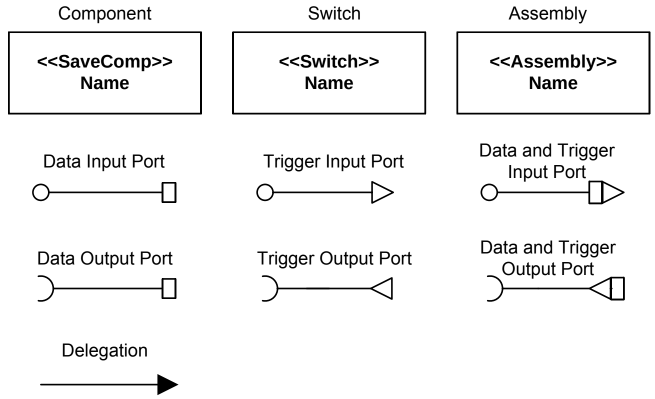
Method & Reference	Encapsulated Evaluation Model	Operational Profile	Composition Algorithm	Evaluation Algorithm
Fenelon, McDemid, Nicholson, Pumfrey [13,14]	FPTN modules that describe the propagation and transformation of failure for one component	Not considered	Hierarchical composition of the FPTN modules + wiring input and output failure ports	Determination of the probabilities of the top level failure modes, manual, not tool supported
Papadopoulos, McDermid, Heiner, Sasse [28]	Tabular failure annotations of (Matlab/Simulink) components, extension of FPTN modules	Not considered, but possible handling of input failures generated by the environment	Automatic generation of system-level fault-trees and FMEA-tables	Minimal cutset analysis and determination of the probabilities of the top level fault tree nodes, automatical, with commercial fault tree tools
Grunske, Kaiser, Liggesmeyer, Mäckel [11,10,12]	Component Fault Trees (CFT), modular and hierarchical decomposable fault trees where the interfaces are described by typed input and output failure ports	Not considered, but possible handling of input failures generated by the environment	Model-based construction of hierarchical CFTs based on the architecture of the system (wiring input and output failure ports based on the system's failure flow)	Determination of hazard probabilities (CFTs + annotated hazard conditions) + automatical and tool-supported with BDD algorithms (UWG3-www.essarel.de)
Kaiser Gramlich, Grunske, Papadopoulos [15,16]	State/event-based fault trees (SEFT), semantic based on stochastic Petri nets	Not considered, but possible handling of input failures generated by the environment	Model-based construction of hierarchical SEFTs according to the architecture of the system, based on failure propagation and port wiring	Determination of hazard probabilities (SEFTs + annotated hazard conditions) by simulation of a stochastic Petri Net(ESSaRel www.essarel.de)

Safety Evaluation Case Study

Safety Evaluation of a Computer Assisted Braking System with SaveCCM

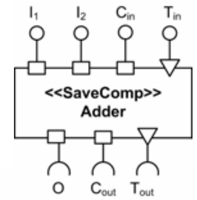
SaveCCM

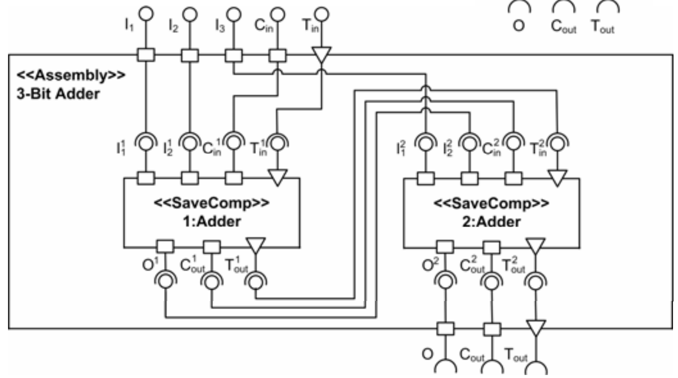
- SaveCCM is a architecture description language for embedded control applications in automotive (vehicular) systems.



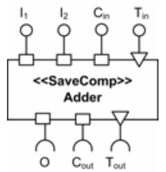
SaveCCM Syntax

- SaveCCM benefits for safety eval.
  - Stongly Encapsulated Interfaces
  - Hierarchical (De)Composition

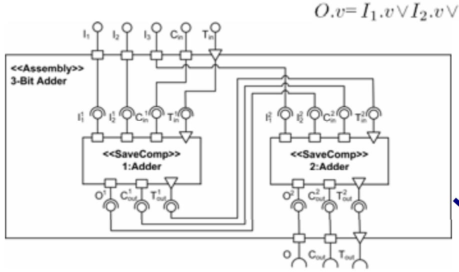




SaveCCM vs. FPM



$$O.v = I_1.v \vee I_2.v \vee C_{in}.v \vee InternalFault \vee T_{in}.te \vee T_{in}.tl \vee T_{in}.o \vee T_{in}.c$$



$$O.v = I_1^2.v \vee (I_1^1.v \vee I_2^1.v \vee C_{in}^1.v \vee InternalFault^1 \vee T_{in}^1.te \vee T_{in}^1.tl \vee T_{in}^1.o \vee T_{in}^1.c)$$
$$\vee C_{in}^2.v \vee InternalFault^2 \vee T_{in}^2.te \vee T_{in}^2.tl \vee T_{in}^2.o \vee T_{in}^2.c.$$

Failure Modes of Components

- Assumption
  - Components exchange information (services, messages, etc.) only via ports
- Derivation from expected information is called a *failure*
- For each service / message that some component produces or consumes, different failure modes can be assigned, e.g.
  - Value failure
  - Timing failure (too early / too late)
  - Omission failure (service not delivered when requested)
  - Commission failure (undesired service provided)
- As ports in structural models designate information / service propagation (failure propagation)

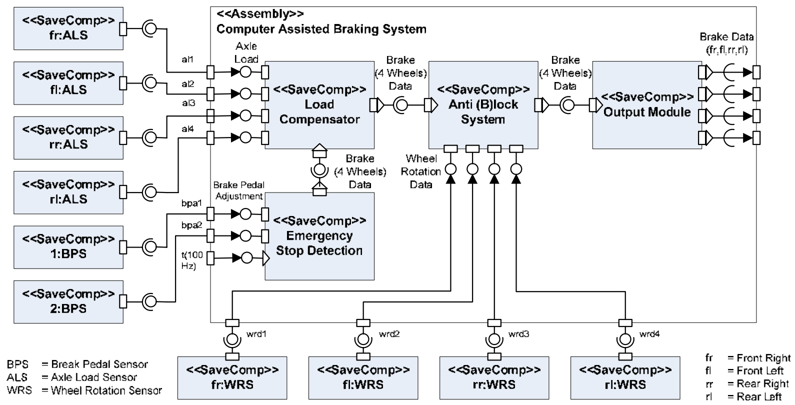
Safety Evaluation Process

Safety Evaluation Steps

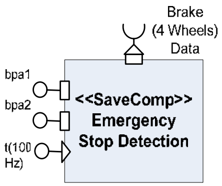
1. Generate an encapsulated failure propagation model for each SaveCCM *Component* and *Switch*.
2. Identify the relations between system output failures and hazards.
3. Construct an encapsulated failure propagation model for each SaveCCM *Assembly*.
4. Calculate the output failure probabilities of the system-level *Assembly* and accordingly the hazard probabilities of the system.
5. Compare the calculated hazard probabilities with the tolerable hazard probabilities.

Case Study

- Computer Assisted Braking System



Step 1



SHARD

SaveCCM Component Emergency Stop Detection	Output Failure Flow Brake (4 Wheels) Data (B4WD)				
	B4WD.v	B4WD.te	B4WD.tl	B4WD.o	B4WD.c
bpa1.v	×				
bpa2.v	×				
t(100Hz).te		×			
t(100Hz).tl			×		
t(100Hz).o			×	×	
t(100Hz).c		×			×
Internal timing problem (Int1)		×	×	×	×
Wrong interpretation of the input data(Int2)	×				
Hardware failure of the underlying control unit(Int3)		×	×	×	×
Memory problem in the data segment of the underlying control unit(Int4)	×				

Generation of a FPM

$B4WD.v = (bpa1.v \wedge bpa1.v) \vee Int2 \vee Int4$

$B4WD.te = t(100Hz).te \vee t(100Hz).c \vee Int1 \vee Int3$

$B4WD.tl = t(100Hz).tl \vee t(100Hz).o \vee Int1 \vee Int3$

$B4WD.o = t(100Hz).o \vee Int1 \vee Int3$

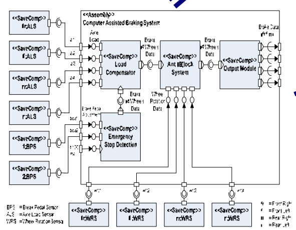
$B4WD.c = t(100Hz).c \vee Int1 \vee Int3$

Repeat this
for all components

Step 2

Hazard ID	Effect Description	Risk Class	Tolerable Hazard Rate (THR) (per hour)
H1	Complete lack of braking	Catastrophic	10^{-8}
H2	Lock up (1–4 wheels)	Catastrophic	10^{-8}
H3	Unexpected application/release of the brakes	Catastrophic	10^{-8}
H4	Braking response not proportional to demand	Catastrophic	10^{-8}
H5	Tardy/slow response	Major	10^{-7}
H6	Uneven braking (pressures vary "wildly" in response to constant demand)	Major	10^{-7}
H7	Unequal braking (1-3 wheels brake less or more than required)	Major	10^{-7}

PHI,PHA



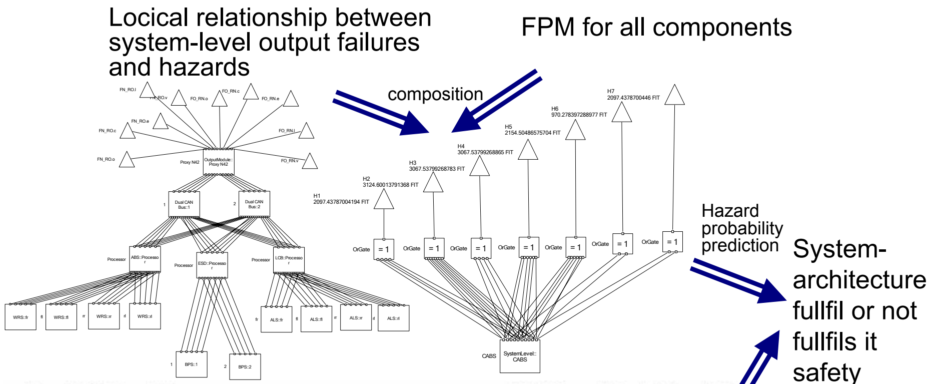
SHARD

System Output	Hazardous Failure Modes H1	H2	H3	H4	H5	H6	H7
BrakeData.o	×	×		×	×		
BrakeData.c		×	×	×		×	
BrakeData.te							
BrakeData.tl		×			×		
BrakeData.v	×	×	×	×		×	×

Generation of OF2H

Logical relationship between system-level output failures and hazards

Step 3-5

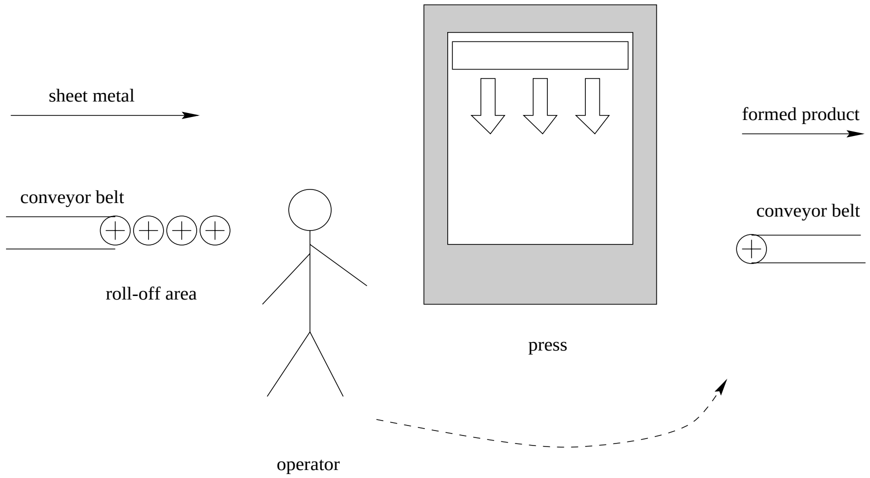


Hazard ID	Effect Description	Risk Class	Tolerable Hazard Rate (THR) (per hour)
H1	Complete lack of braking	Catastrophic	10^{-8}
H2	Lock up (1–4 wheels)	Catastrophic	10^{-8}
H3	Unexpected application/release of the brakes	Catastrophic	10^{-8}
H4	Braking response not proportional to demand	Catastrophic	10^{-8}
H5	Tardy/slow response	Major	10^{-7}
H6	Uneven braking (pressures vary "wildly" in response to constant demand)	Major	10^{-7}
H7	Unequal braking (1-3 wheels brake less or more than required)	Major	10^{-7}

Safety Evaluation Exercise

Industrial Metal Press

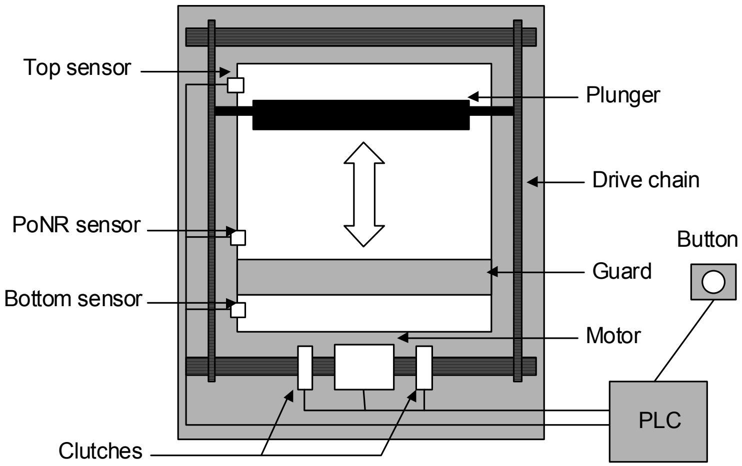
Industrial Press: operational concept



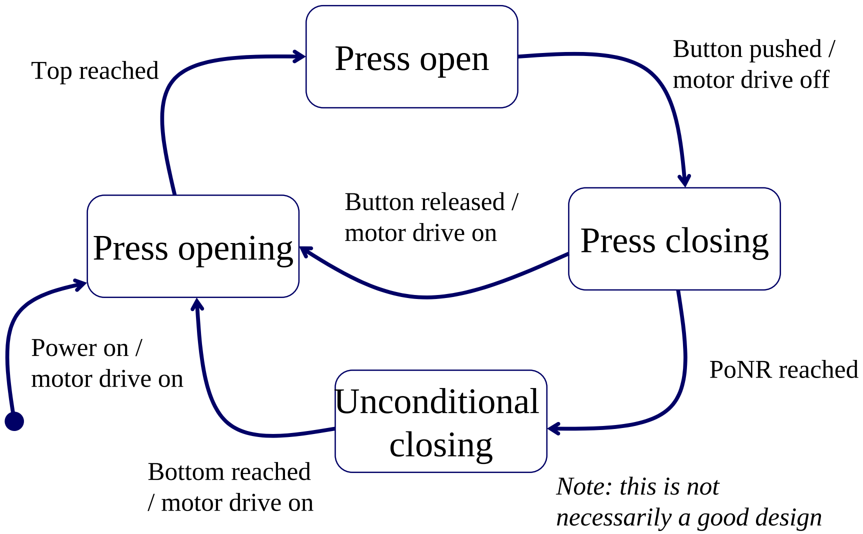
Industrial Press: system-level view

- Press main functions:
  - Raise plunger to top (open the press)
  - Release plunger (close the press)
  - Abort operation (stop closing & reopen the press)
- System-level requirements/operational concept:
  - Upon start-up, press will open fully
  - If button is pushed while press is fully open, press will start to close
  - Upon closing, press will automatically reopen
  - If safe to do so, closing can be aborted by releasing the button
    - Safe = above Point of No Return (PoNR)

Industrial Press: Press physical architecture



Industrial Press: Control Logic



Questions

- What are the safety requirements?
- What are the system hazards?
- What are the tolerable hazard rates?
- What are the relations between system failures and system hazards?
- What are the encapsulated evalution models?
- What are the component failure probabilities?
- Is the system safe?
- How could the system be improved?

Answers

☺hidden☺

Open Problems and Future Work

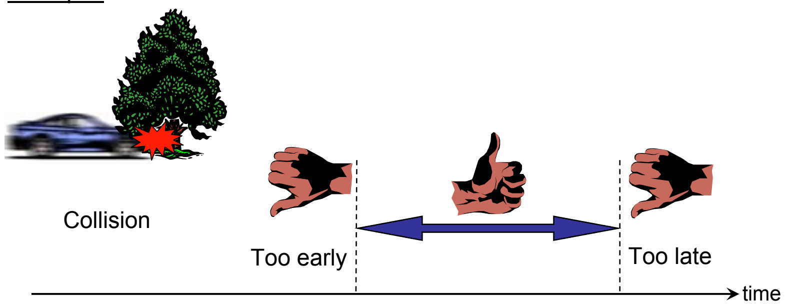
- How can we determine the probability of an internal software defect or fault?
  - Empirical data
  - Measurement-based models
  - It is hard to determine the resulting failure modes for a given fault
- Effort for the COTS component vendors to produce the failure propagation models
  - All stakeholders must use compatible models / failure categories
  - Reuse potential promises pay-off

Performance, Realtime

Real-time systems

RT Systems : Correct result at the right time

Example:



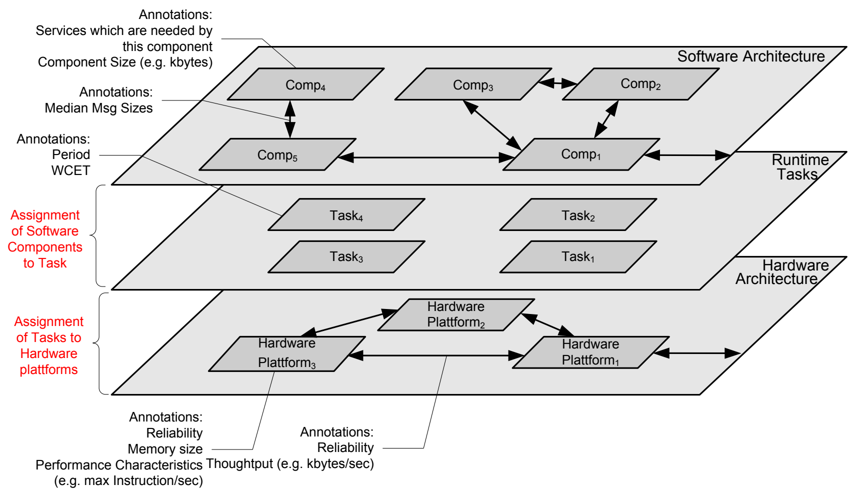
An air bag must not be inflated too late, nor too early!

What are Real-Time Systems

- A Real-Time System has a number of **Tasks** (Programs)
  A Task has a set of properties
  - **Deadline (D)**
  - **Period / Minimum inerarrival time (T / MINT)**
  - **Worst/Best Case Execution Time (WCET / BCET)**
  - **Transactions with an end-to-end deadline (E2ED)**
- A real-time system has a **scheduler** that uses a **scheduling policy** to assign a task a fraction of the processor time

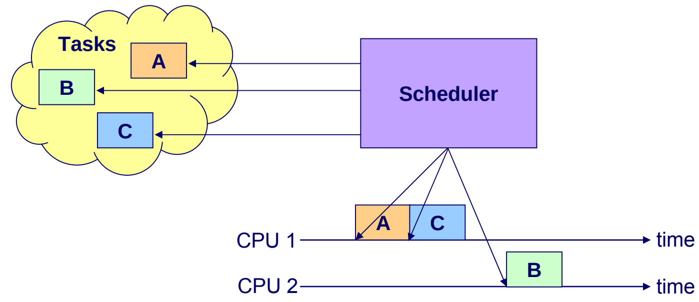
E2ED

Scheduling Analysis



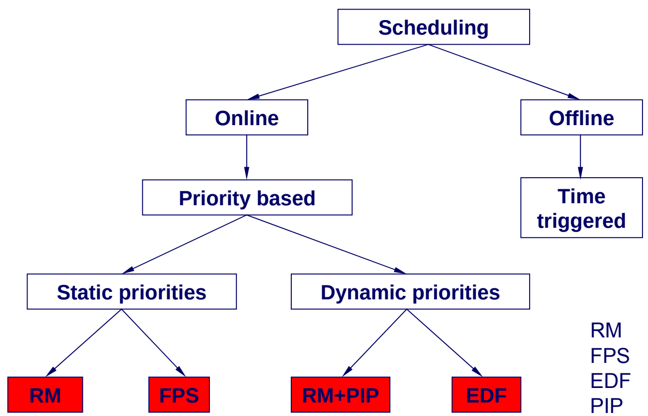
Scheduling

Example:
"Run task A at time 3 on CPU 1"
"Run task B after task A on CPU 2"



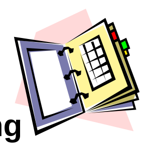
Used to meet the demands in a best possible way

Simple classification of scheduling algorithms



RM Rate Monotonic
FPS Fixed Priority Scheduling
EDF Earliest Deadline First
PIP Priority Inheritance Protoc

Offline scheduling



Also known as static or pre-run-time scheduling

Static schedule (time table) created before we start the system

Run-time dispatching: just follows the generated time table

Properties (compared to online scheduling)

(+) Allows more complex task models

(+) More difficult scheduling problems

(−) Less flexible

Analysis

"proof by construction"

Online vs offline scheduling

Online scheduling

- (+) flexible
- (+) relatively simple analysis

- (-) difficult to cope with complex constraints
- (-) less deterministic

Offline scheduling

- (+) deterministic
- (+) simplier to test and verify
- (+) handles complex constraints

- (-) new schedule must be generated if we add a new function
- (-) it could take a long time to produce a schedule

Online vs. offline scheduling

When to use each of the methods?

Offline scheduling

High demands on timing and functional verification, testability and determinism

Safety-critical applications, e.g., control system for Boeing 777

Online scheduling

Demands on flexibility, meny non-periodic activities

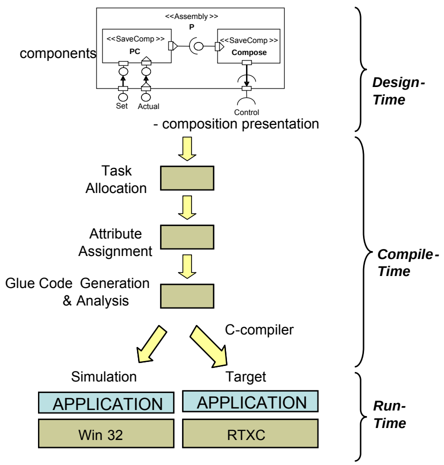
Example: multimedia applications, webservers,..

Combination of both

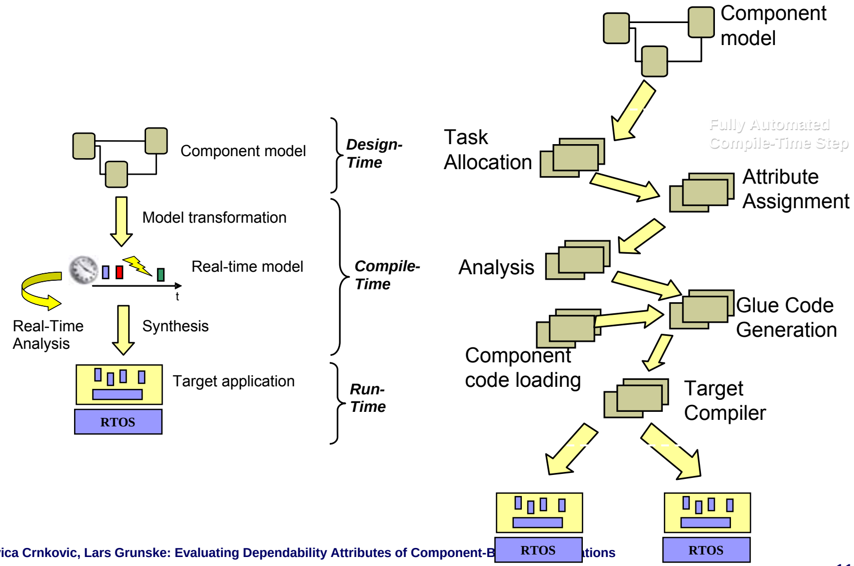
Combined offline and online scheduling

The time critical parts scheduled offline and non-critical parts online

From component model to RT execution model

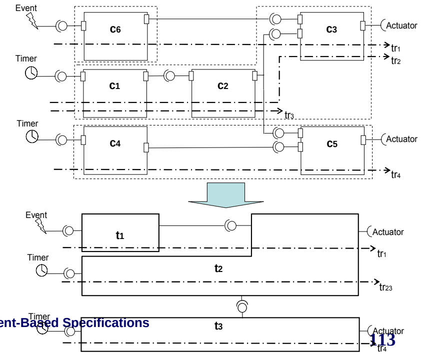


From component model to RT execution model

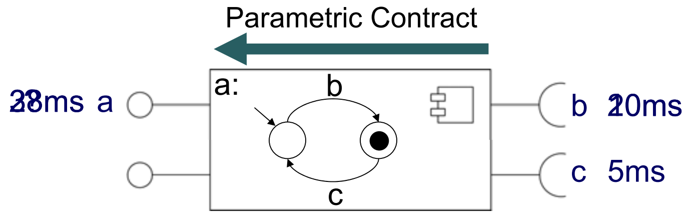


Allocating components to real-time tasks

- Today one-to-one allocation is commonly used
  - Not efficient in terms of cpu-overhead and stack usage
  - However, highly analyzable
- How can the mapping between components and tasks be analyzable and efficient?
- Infeasible to calculate due to the many different possible mappings in a large system
- Limitations
  - Only pipe-and-filter architectures
  - No advanced real-time constraints



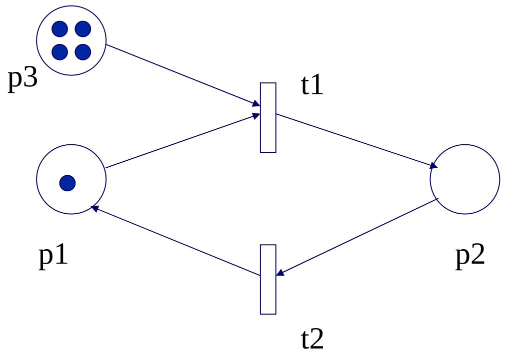
Parametric Contracts



- Lifting the Design-by-Contract Principle to Software Components
- Linking the provided and required services of the same component
- Specified by the QML+ Service Effect Automata

Stochastic Petri Nets

- Petri Nets
  - Places,
  - Transition,
  - Token
- Petri nets are extended by associating time with the firing of transitions, resulting in timed Petri nets.
  - A special case of timed Petri nets are stochastic Petri nets (SPN) where the firing times are determined by random variables.
  - exponentially distributed firing times
- Generalized SPN (GSPN)
  - Transition with zero firing times



Performance Evaluation Techniques & Generic Framework

Method & Reference	Encapsulated Evaluation Model	Operational Profile	Composition Algorithm	Evaluation Algorithm
Wu, McMullan, Woodside [24,25]	Layered queueing networks (LQN), which provide a hierarchical black-box view of the performance of a component	Modelled as input queues	Automatic generation of a layered performance models from component sub-models, tool supported	Traditional evaluation of the top-level LQN
Firus, Becker, Reussner [23]	Parametric performance contracts, similar approach as the reliability evaluation described in [21]	Service effect automata, similar to [21]	Hierarchical composition of the parametric contracts and service effect automata	Calculation of the time consumption of possible call sequences incl. loops and choices, not explained in detail
Bertolino, Mirandola [22]	Performance annotations conform to OMG's SPT profile [37]	Weighted use cases with call probabilities	Construction of a formal model (queueing network) based on the performance annotations, deployment architecture and resource usage, supported by XMI-transformations	Response times are calculated with standard queueing network analysis algorithms and tools

Availability, Reliability, Maintainability

Very simple model for terminating batch sequel systems [late 70ies]

- *Comp* is the set of components that can be called.
- q_i is the probability that the component C_i will be called and r_i is the binary reliability of the component C_i (ether the component will produce the correct output or not).
- The reliability of the system can be determined as follows:

$$R = \sum_{\forall C_i \in Comp} q_i r_i$$

- The problems of this model are obvious

Reliability Evaluation Techniques & Generic Framework

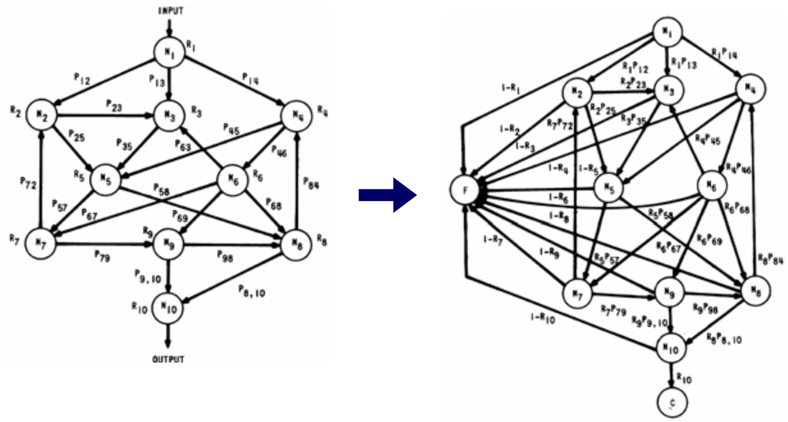
Method & Reference	Encapsulated Evaluation Model	Operational Profile	Composition Algorithm	Evaluation Algorithm
Hamlet, Mason, Woit [17]	Reliability measures, independent from the operational profile of components, profile mappings are used to obtain the reliability measures in the deployment context	Operational profiles at the system level (also know as trail profiles)	Composition of the evaluation models based on the system-control flow	The system reliability is calculated based on traditional reliability equations (extended by equations for conditional cases and loops)
Yacoub, Ammar [18]	Dynamic reliability metrics	Description of the operation profile with sequence diagrams	Generation of component dependency graphs, that describe probabilistic call sequences as Markov models	Assessment of the reliability-based risk of a component by traversing the component dependency graph
Reussner, Poernomo, Schmidt [35,36,21]	Parametric contracts, a generalisation of design-by-contract principle based on the Quality of Service Modelling Language" (QML) [32]	Service effect automata, that describe the call probabilities of services, these service effect automata are also used to describe the environment of a single component	Composition of service effect automata + identification of the accepted language (traces) of the composed service effect automaton	For each trace the reliability of a service can be determined with traditional methods, the final reliability is the sum of the individual trace reliabilities

User Oriented Software Reliability Model [Cheung 80]

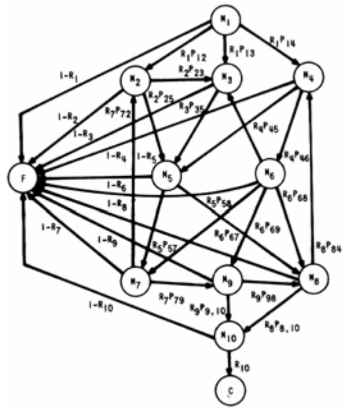
- Assumptions:
  - The operation profile of the system is defined by the probabilities of the transfer of control between component
  - This control transfer follows Markov-properties
  - System has exactly one start and one end-component
- Notation
  - R_i reliability of component N_i
  - P_{ij} probability of correct control transfer from component N_i to component N_j

$$Q = \begin{array}{c} \\ N_1 \\ \vdots \\ N_i \\ \vdots \\ N_{n-1} \\ N_n \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_2 & \cdots & N_j & \cdots & N_n \end{array} \\ \begin{bmatrix} 0 & R_1P_{12} & \cdots & R_1P_{1j} & \cdots & R_1P_{1n} \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & R_iP_{i2} & \cdots & R_iP_{ij} & \cdots & R_iP_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & R_{n-1}P_{(n-1)2} & \cdots & R_{n-1}P_{(n-1)j} & \cdots & R_{n-1}P_{(n-1)n} \\ 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \end{array} .$$

User Oriented Software Reliability Model [Cheung 80]



User Oriented Software Reliability Model [Cheung 80]



- P^n is the the nth power matrix of P
- Consequently, $P^n(i,j)$ is the probability of reaching state N_j from the starting state N_i within n steps
- Reliability of the system $R=P^n(N_1,C)$

$$\hat{P} = \begin{array}{c|cccccccc} & C & F & N_1 & N_2 & \cdots & N_j & \cdots & N_n \\ \hline C & 1 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ F & 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ N_1 & 0 & 1-R_1 & 0 & R_1P_{12} & \cdots & R_1P_{1j} & \cdots & R_1P_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ N_i & 0 & 1-R_i & 0 & R_iP_{i2} & \cdots & R_iP_{ij} & \cdots & R_iP_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ N_{n-1} & 0 & 1-R_{n-1} & 0 & R_{n-1}P_{(n-1)2} & \cdots & R_{n-1}P_{(n-1)j} & \cdots & R_{n-1}P_{(n-1)n} \\ N_n & R_n & 1-R_n & 0 & 0 & \cdots & 0 & \cdots & 0 \end{array}$$

User Oriented Software Reliability Model [Cheung 80]

- Probability calculation trick: Let S be an n by n matrix such that

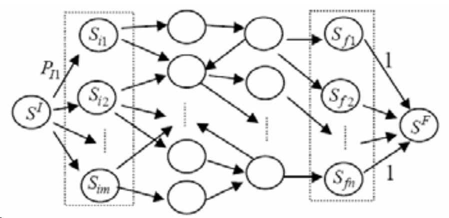
$$S = I + Q + Q^2 + Q^3 + \cdots = \sum_{k=0}^{\infty} Q^k.$$

$$W = I - Q$$

- [Cheung 80] shows that $S = W^{-1} = (I-Q)^{-1}$ and as a result the reliability of a system can be calculated as follows: $R=S(1,n)R_n$

Architecture-Based Software Reliability Model (1) [Wang et al. 99]

- Based on the [Cheung 80] model
- Extension
  - Multiple entry points & multiple exit point
    - Realistic operational profile
  - Extension for Architectural Styles

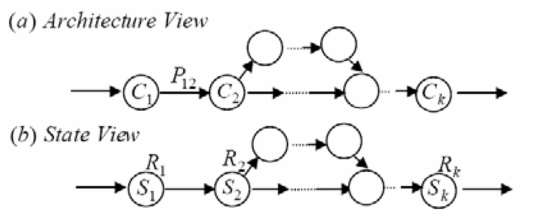


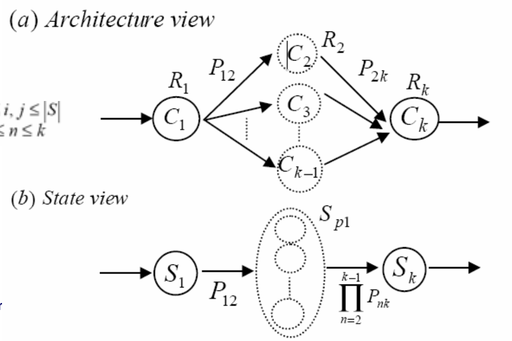
$$M = \begin{array}{c|cccccc} & S_1 & S_2 & .. & S_i & .. & S_{n-1} & S_n \\ \hline S_1 & 0 & R_1P_{12} & .. & R_1P_{1i} & .. & R_1P_{1(n-1)} & R_1P_{1n} \\ S_2 & R_2P_{21} & 0 & .. & R_2P_{2i} & .. & R_2P_{2(n-1)} & R_2P_{2n} \\ \vdots & \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ S_i & R_iP_{i1} & R_iP_{i1} & .. & 0 & .. & R_iP_{i(n-1)} & R_iP_{in} \\ \vdots & \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ S_{n-1} & R_{n-1}P_{(n-1)1} & R_{n-1}P_{(n-1)2} & .. & R_{n-1}P_{(n-1)i} & .. & 0 & R_{n-1}P_{(n-1)n} \\ S_n & R_nP_{n1} & R_nP_{n2} & .. & R_nP_{ni} & .. & R_nP_{n(n-1)} & 0 \end{array}$$

Architecture-Based Software Reliability Model (2) [Wang et al. 99]

- Batch-sequential/ pipeline style
  - Analysis: identical to [Cheung 80]
- Parallel/ Pipe-filter style
  - Analysis

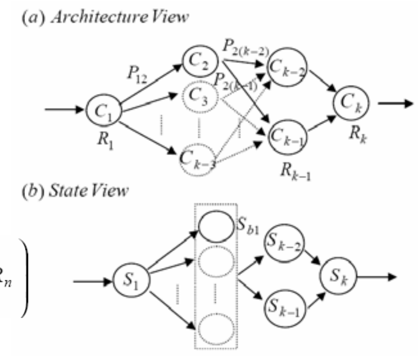
$$M(i,j) = \prod_{C_n \text{ in } S_i} R_n P_{nj},\ S_i \in S_p\ ,\ \text{for } 1 \le i, j \le |S| \text{ and } 1 \le n \le k$$

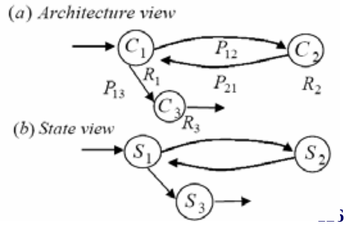




Architecture-Based Software Reliability Model (3) [Wang et al. 99]

- Fault Tolerance
  - Primary component C_2 and a set of backup components
  - Analysis: Reliability (by Induction) $R_2 + \sum_{n=3}^{k-3}\left(\left(\prod_{m=2}^{n-1}(1-R_m)\right)R_n\right)$ times transition probability
  - Assumption: Independent Failure
- Call- Return
  - Analysis: identical to [Cheung 80]
  - Problem: Loop





Architecture-Based Software Reliability Model [Wang et al. 99] Example

	S_1	S_2	S_{b1}	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
S_1	0	1	0	0	0	0	0	0	0	0
S_2	0	0	.981018	0	0	0	0	0	0	.000982
S_{b1}	0	0	0	.2267	.6682	0	.1039	0	0	0
S_5	0	.048	0	0	.951	0	.104	0	0	.001
S_6	0	0	.4222	0	0	.1	0	.4132	.1	.061
S_7	0	0	0	0	.99	0	0	0	0	0
S_8	0	0	0	0	1	0	0	0	0	0
S_9	0	0	0	0	.01	0	0	0	.99	0
S_{10}	0	0	0	0	.8999	0	0	0	0	0
S_{11}	0	0	0	0	0	0	0	0	0	0

$n = 10$

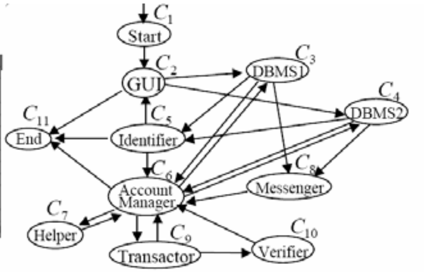
$|I - M| = 0.106059$

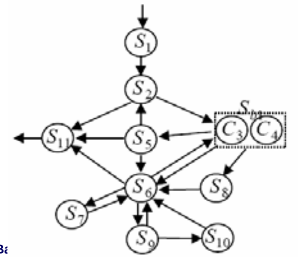
$|E| = -0.059289$

$$T(1, \{S_{11}\}) = (-1)^{n+1} \frac{|E|}{|I-M|} = 0.559$$

The overall system reliability R is obtained as:

$R = T(1, \{S_{11}\}) \times R_{11} = 0.56$



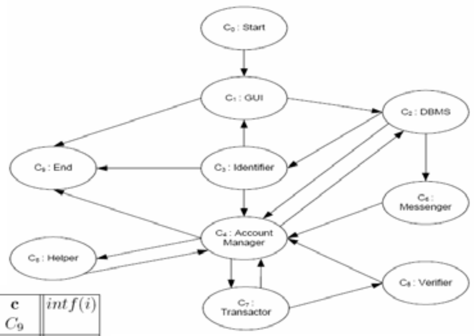


Architecture-Based Software Reliability Model with Error-Propagation [Cortellessa,Grassi 07]

- Based on [Cheung 80] and [Wang et al.99]
- Each component has two reliability metrics
  - Internal failure probability *intf()*
  - Error propagation probability *ep()*

Architecture-Based Software Reliability Model with Error-Propagation [Cortellessa,Grassi 07] Example

- Results are more realistic
- Component Selection is more accurate



	Q C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	c C_9	$intf(i)$
C_0	0	1	0	0	0	0	0	0	0	0	0
C_1	0	0	0.999	0	0	0	0	0	0	0.001	0.018
C_2	0	0	0	0.227	0.669	0	0.104	0	0	0	0.035
C_3	0	0.048	0	0	0.951	0	0	0	0	0.001	0
C_4	0	0	0.4239	0	0	0.1	0	0.4149	0	0.0612	0.004
C_5	0	0	0	0	1	0	0	0	0	0	0.01
C_6	0	0	0	0	1	0	0	0	0	0	0
C_7	0	0	0	0	0.01	0	0	0	0.99	0	0
C_8	0	0	0	0	1	0	0	0	0	0	0.1001
C_9	0	0	0	0	0	0	0	0	0	1	0

$\forall i, ep(i)$	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Rel	0.4745	0.8261	0.8989	0.9399	0.9494	0.9617	0.9710	0.9784	0.9848	0.9906

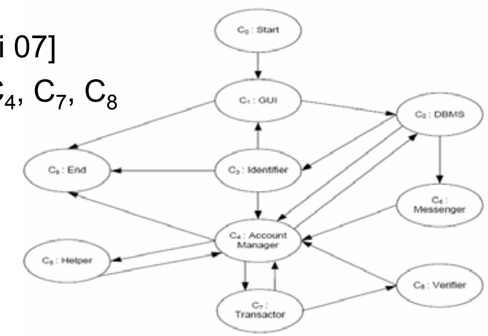
Sensitivity Analysis

- Find the component C_i with the most influence on the system reliability
- Identical to identifying architecture optimisation points, like
  - Bottleneck (Performance)
  - Single Point of Failure (Safety)
- With respect to the component reliability [Cheung 80], [Wang et al. 99], [Cortellessa, Grassi 07] : $\frac{\partial Rel}{\partial intf(i)}$
- With respect to the error propagation probability [Cortellessa, Grassi 07]: $\frac{\partial Rel}{\partial ep(i)}$

Sensitivity Analysis – Example

Source [Cortellessa, Grassi 07]

Sensitive Component C_2, C_4, C_7, C_8



	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
$ep(i)$	1	1	1	1	1	1	1	1
$\frac{\partial Rel}{\partial ep(i)}$	-0.0199	-1.7830	-0.4360	-4.2732	-0.4246	-0.2001	-1.6031	-1.5853
$intf(i)$	0.018	0.035	0	0.004	0.01	0	0	0.1001
$\frac{\partial Rel}{\partial intf(i)}$	-0.5051	-2.1502	-0.4705	-3.8948	-0.3864	-0.2159	-1.4442	-1.5870

Reliability Evaluation for Service Oriented Architectures

- Based on the [Kubat 89] model (formulation in the SOA domain is still pending)
- Notation:
  - *K* describes a set of services of a system
  - r_k is the service call arrival rate (Operational Profile)
- Solution:

$$\lambda_S = \sum_{k=1}^{K} r_k [1 - R(k)],$$

  - R(k) is calculated traditionally based on the number of visits for each component and the component reliabilities when the task is called.
  - The architecture is a DTMC with transition probabilities p_{ij} between components

Further Models and Readings

- Classification of [Goseva-Popstojanova, Trivedi 01]
- State based models
  - Reliability Prediction and Sensitivity Analysis Based on Software Architecture [Gokhale et al. 02] [Gokhale, Trivedy 98]
  - Software Dependabilty [Kanoun, Sabourin 87]
  - Laprie model for dynamic failure behaivior [Laprie84] [Laprie, Kanoun 92]
  - Littlewood model [Littlewood 1979]
- Path based model (eg. [Yacoub et al. 99])
- Additive models (eg. [Xie, Wohlin 95])

Open Problems and Future Work

- How can we determine the probability of an internal software defect or fault?
  - Empirical data
  - Measurement-based models
  - It is hard to determine the resulting failure modes for a given fault
- How can we determine the transition probabilities
- What are the limitations and assumptions of these models

SHARPE: Symbolic-Hierarchical Automated Reliability and Performance Evaluator

- Robin A. Sahner &Kishor S. Trivedi
- Evaluation Backend for multiple Input Models

Model Type	Dependability	Performance	Performability
Fault tree (FT)	X		
Multistate fault tree	X		
RBD	X		
Reliability graph (RG)	X		
Markov chain	X	X	X
Semi-Markov chain	X	X	X
MRGP	X	X	X
GSPN	X	X	X
Stochastic reward net	X	X	X
PFQN		X	
MPFQN		X	
Task graph		X	
Phased-Mission systems	X		

```
* aircraft flight control system

bind
mIRS .000015
mPRS .000019
mSA .000037
mCS .00048
end

ftree aircraft
basic IRS exp(mIRS)
basic PRS exp(mPRS)
basic CS exp(mCS)
basic SA exp(mSA)
kofn IRS23 2,3, IRS
kofn PRS23 2,3, PRS
kofn CS34 3,4, CS
kofn SAS23 2,3, SA
or TOP IRS23 PRS23 CS34 SAS23
end

format 8

expr mean(aircraft)
eval (aircraft) 1000 10000 1000
expr value(10;aircraft) end
```

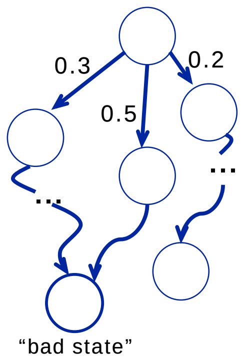
input

```
mean(aircraft):    2.21322439e+03
----------------------------------------

     system aircraft
        t              F(t)

  1.00000000 e+03 1.62926398 e-01
  2.00000000 e+03 5.15384469 e-01
  3.00000000 e+03 7.70884601 e-01
  4.00000000 e+03 9.02767485 e-01
  5.00000000 e+03 9.61222540 e-01
  6.00000000 e+03 9.85111991 e-01
  7.00000000 e+03 9.94421933 e-01
  8.00000000 e+03 9.97944094 e-01
  9.00000000 e+03 9.99250932 e-01
  1.00000000 e+04 9.99729373 e-01
----------------------------------------

value(10;aircraft):  1.02381366e-06
```

output

Probabilistic Model Checking

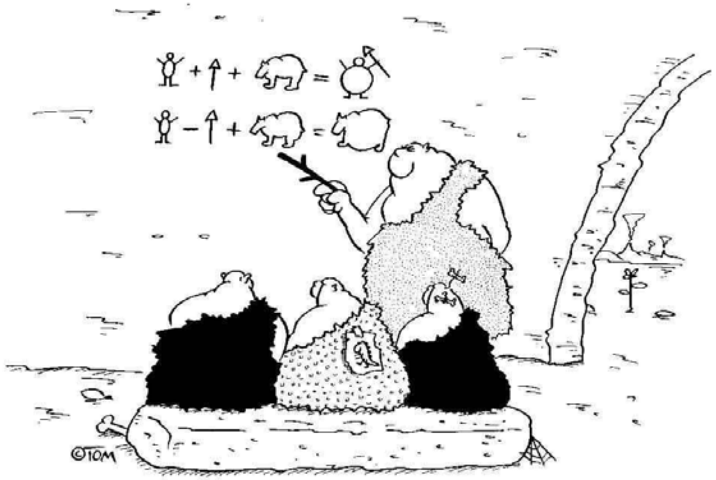
- Probabilistic model checking question:
  - *What's the probability of reaching bad state?*
- Model
  - CTMC, DTMC, GSPN, …
- Property Specification
  - CSL (Continuous Stochastic Logic)
  - PCTL (Probabilistic Computation Tree Logic)
- Model Checker
  - PRISM,
  - ETMCC
  - VESTA
- Problems: State Explosion, Limited Support of Counter Examples



Application of Dependability Evaluation Techniques

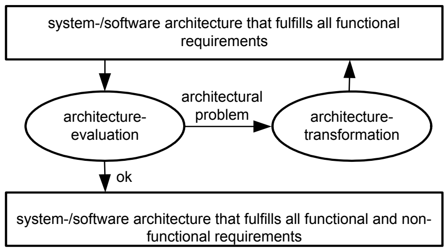
Dependability Optimisation

How Can Quantitative Architecture Evaluation be USED in practices



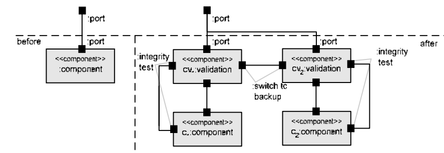
Background Dependability Optimisation: Simple Solution

- Goal: Quality improvement by architecture transformation
- Solution:
  - Evaluation algorithms to determine the quality of the architecture (eg. Component Fault Trees (CFTs) → safety)
  - Transformation operators:
    - Improve the non-functional properties
    - Preserve the functional properties
  - Search with Backtracking

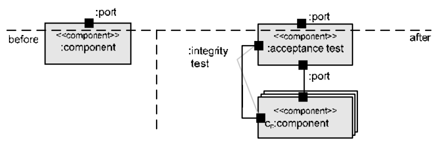


Architecture Transformation: Quality improving transformation operators

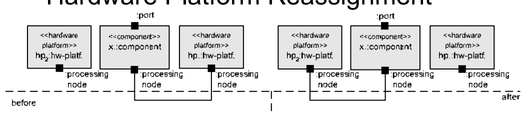
- Two-Channel-Redundancy



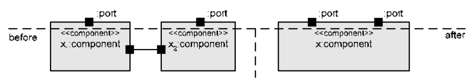
- Recovery Block



- Hardware Platform Reassignment



- Process Fusion



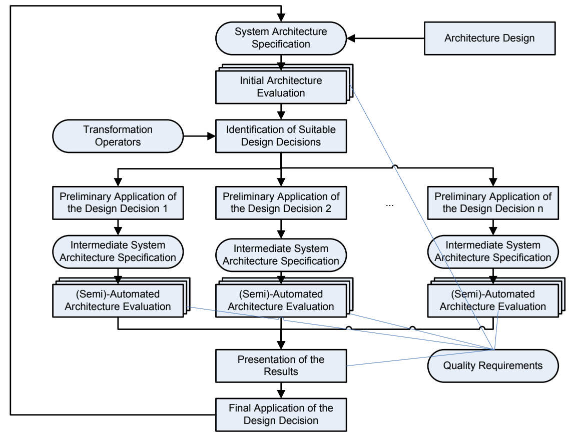
- Further transformation operators /Viking-Plop 2003/
  - Multi-Cannel-Redundancy with Voting
  - Protected-Single-Channel
  - Hardware Platform Substitution
  - Hardware Platform Reassignment
  - Actuation-Monitor
  - Integrity Check
  - Watchdog

All Problems Solved???

- How to improve dependability aspects early in the system development lifecycle?
  - Rigorous assessment, evaluation and analysis of design specifications (architecture specifications)
    - because the earlier a quality problem can be identified, the better and more cost effectively this problem can be fixed.
  - Dependability Improving Action →Early in the development process
  - Problem: Dependability requirements conflicting with each other.
    - Trade-Offs
- Motivation
  - The fulfilment of dependability requirements is very important for the success of a software project.

Trade-off Analysis Method

TAFES Framework (Trade-off Analysis For Embedded Systems)



General Introduction to Multiobjective Optimisation Problems

- Multiobjective Optimization Problem
  - Find a solution *x* which is an element of the solution space *X*, satisfies a set of constrains $g_i(x)$ and optimizes a vector function $f(x)= [f_1(x),f_2(x),f_3(x),\ldots,f_n(x)]$ whose elements represent the objective functions.
- Pareto Optimal Solutions
  - Set of non-dominated solutions
    - a solution x_1 is dominated by another solution x_2 if x_2 matches or exceeds x_1 in all objectives.

Multiobjective Optimisation Problem for Our Problem

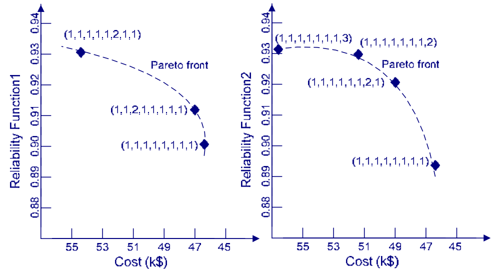
- Problem Definition:
  - Find a solution *x* (an architecture design) which is an element of the solution space *X* (set of all possible design solutions), satisfies a set of constrains $g_i(x)$ (economic and engineering constrains) and optimizes a vector function $f(x)= [f_1(x),f_2(x),f_3(x),\ldots,f_n(x)]$ whose elements represent the objective functions (fulfillment of dependability requirements).

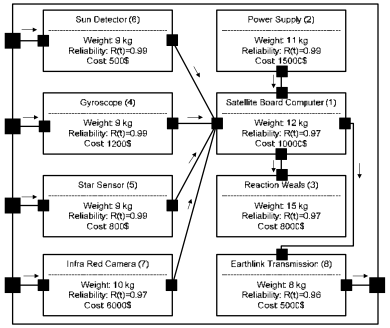
Multiobjective Optimization and Architecture Trade-Off Analysis

- Simple Solution
  - Evolutionary Algorithms
    - Mutation operators → Architecture refactorings
    - Ranking procedure→ Quantitative architecture evaluations

Example (Multiobjective Optimization)

- DLR's BIRD (Bi-spectral InfraRed Detector)
  - Two critical functions
    - Function 1: Attitude Control Function (ACF) intended to control the satellite's position and rotation. →needed components (1,2,3,4,5,6)
    - Function 2: Collection of infrared sensor data and the transmission of the data to the ground station. →needed components (1,2,7,8)
  - Evaluation (Cost Weight, Reliability [RBDs])





Limitation of the Approach

- Dependability Optimization for Conflicting Quality Objectives
  - Multiobjective Optimization
    - Currently based on Evolutionary Algorithms
    - Future Tasks: Tabu-Search, Memetic algorithms, Swarm-based optimisations (Particle Swarms)
    - Empirical Validation
- General Framework for Model-Driven Quality Evaluation of Component-Based Systems
  - Safety, Performance, Reliability
  - Validation and Experiments for other Quality Attributes
- Still a long way ahead!!!!

Conlusion

- CBD is an attractive approach
- CBD main concern is ability of composition
- Dependability includes attributes that are either not directly composable or composable when system characteristics are known
- Instead of composability, analysis of systems are used
- CBD make the analysis easier since the analysis elements are on higher abstraction level comparing non-component based systems.
- There exits many dependability analysis – they can be applied on CB systems

References

- General
  - Grunske L., Early Quality Prediction of Component-Based Systems - A Generic Framework, Journal of Systems and Software, Elsevier, Volume 80, Issue 5, May 2007, pp. 678–686
- Performance
  - Becker S., Grunske L., Mirandola R. Overhage S., Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective, In R. Reussner, J. Stafford, and C. Szyperski, editors, Architecting Systems with Trustworthy Components, volume 3938 of LNCS, Springer, 2006 pp 169–192.
- Architecture Optimisation
  - Grunske L., Identifying "Good" Architectural Design Alternatives with Multi-Objective Optimization Strategies, International Conference on Software Engineering (ICSE), Emerging Results, Shanghai, ACM 1-59593-085-X/06/0005, 20-28 May 2006, pp. 849-852

References

- Safety
  - Grunske L., Kaiser B., Papadopoulos Y., Model-Driven Safety Evaluation with State-Event-Based Component Failure Annotations, Eighth International ACM SIGSOFT Symposium on Component-based Software Engineering (CBSE 2005), St Luis, Missouri, May 14-15, Lecture Notes in Computer Science Volume 3489, Springer 2005, pp. 33-48
  - Grunske L., Kaiser B., Automatic Generation of Analyzable Failure Propagation Models from Component-Level Failure Annotations, Fifth International Conference on Quality Software, Melbourne, Sep 19 -20, IEEE Computer Society, 2005, pp. 117-123
  - Grunske L., Kaiser B., and Reussner R.H., Specification and Evaluation of Safety Properties in a Component-based Software Engineering Process, book chapter, in Embedded System Development with Components, Lecture Notes in Computer Science vol. 3778, ISBN: 3-540-30644-7, Springer, 2005, pp. 249 - 274
  - Grunske L. and Kaiser B., An Automated Dependability Analysis Method for COTS-Based Systems, 4th International Conference on COTS-Based Software Systems, ICCBSS 2005, Lecture Notes in Computer Science Volume 3412, Springer, Feb 2005, pp 178-190
  - Grunske L., Annotation of Component Specifications with Modular Analysis Models for Safety Properties, Proceedings of the 1st International Workshop on Component Engineering Methodology, Erfurt (WCEM 03), September 22, 2003, pp. 31-41
  - Grunske L., Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM, In proceedings of the conference Quality of Software Architectures (QoSA 2006), volume 4214 of LNCS, Springer, 2006, pp 199–213.
  - Fenelon, P., McDermid, J., Nicholson, M., Pumfrey., D.J.: Towards integrated safety analysis and design. ACM Computing Reviews, 2 (1994) 21–32
  - Kaiser, B., Liggesmeyer, P., M¨ackel, O.: A new component concept for fault trees. In: Proceedings of the 8th AustralianWorkshop on Safety Critical Systems and Software (SCS'03), Adelaide (2003) 37–46
  - Papadopoulos, Y., McDermid, J.A., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. Int. Journal of Reliability Engineering and System Safety 71 (2001) 229–247
  - Papadopoulos, Y., Maruhn, M.: Model-based synthesis of fault trees from matlab-simulink models. In: 2001 International Conference on Dependable Systems and Networks (DSN 2001) (formerly: FTCS), 1-4 July 2001, Göteborg, Sweden, Proceedings, IEEE Computer Society (2001) 77–82
  - Wallace, M.: Modular architectural representation and analysis of fault propagation and transformation. Electr. Notes Theor. Comput. Sci. 141 (2005) 53–71

References

- Reliability
  - R.C. Cheung, A user-oriented software reliability model, IEEE Trans. Software Eng. 6 (2) (1980) 118–125
  - V. Cortellessa V. Grassi, A modeling approach to analyze the impact of error propagation on reliability of component-based systems, CBSE 2007, to appear
  - S. Gokhale, W.E. Wong, K. Trivedi, J.R. Horgan, An analytical approach to architecture based software reliability prediction, in: Proceedings of the Third International Computer Performance and Dependability Symposium (IPDS'98), 1998, pp. 13–22.
  - S. Gokhale, K. Trivedi, Reliability Prediction and Sensitivity Analysis Based on Software Architecture, Proc. of 13th International Symposium on Software Reliability Engineering (ISSRE'02), 2002.
  - K. Goseva-Popstojanova, K.S. Trivedi, Architecture-based approach to reliability assessment of software systems, Performance Evaluation 45 (2–3) (2001) 179–204.
  - D. Hamlet, D. Woit, D. Mason, Theory of software reliability based on components, in: Proceedings of the International Conference on Software Engineering, Toronto, Canada, 2001, pp. 361–370.
  - P. Kubat, Assessing reliability of modular software, Oper. Res. Lett. 8 (35–41) (1989).
  - K. Kanoun, T. Sabourin, Software dependability of a telephone switching system, in: Proceedings of the 17th International Symposium on Fault-tolerant Computing (FTCS'17), 1987, pp. 236–241.
  - S. Krishnamurthy, A.P. Mathur, On the estimation of reliability of a software system using reliabilities of its components, in: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE'97), 1997, pp. 146–155.
  - J.C. Laprie, Dependability evaluation of software systems in operation, IEEE Trans. Software Eng. 10 (6) (1984) 701–714.
  - J.C. Laprie, K. Kanoun, X-ware reliability and availability modeling, IEEE Trans. Software Eng. 18 (2) (1992) 130–147.
  - B. Littlewood, Software reliability model for modular program structure, IEEE Trans. Reliability 28 (3) (1979) 241–246.
  - J.D. Musa, A. Iannino, K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.
  - J.D. Musa, Operational profiles in software reliability engineering, IEEE Software 10 (2) (1993) 14–32.
  - R.H. Reussner, H.W. Schmidt, I.H. Poernomo, Reliability prediction for component-based software architectures, Journal of Systems and Software, no. 66, 2003, pp. 241-252.
  - W.Wang,Y.Wu,M. Chen, An architecture-based software reliability model, in: Proceedings of the Pacific Rim International Symposium on Dependable Computing, 1999, pp. 143–150.
  - M. Xie, C. Wohlin, An additive reliability model for the analysis of modular software failure data, in: Proceedings of the Sixth International Symposium on Software Reliability Engineering (ISSRE'95), 1995, pp. 188–194.
  - S. Yacoub, B. Cukic, H. Ammar, Scenario-based reliability analysis of component-based software, in: Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99), 1999, pp. 22–31.