Resilient Systems - Predictability and Evolution

Ivica Crnkovic
Mälardalen University, Sweden

www.idt.mdh.se/~icc
ivica/crnkovic@mdh.se
What is resilience?

• In ecology\(^1\)
  – the capacity of an *ecosystem* to respond to a perturbation or disturbance by *resisting damage* and *recovering* quickly

• In System Engineering
  – Resilience is the ability of organizational, hardware and software systems to mitigate the severity and likelihood of failures or losses, to adapt to changing conditions, and to respond appropriately after the fact.

\(^1\)Resilience (ecology - Wikipedia)
Characteristics of resilient systems

• Resilience has been defined in two ways in ecological literature\(^2\):
  
  – ‘engineering resilience’ : as the time required for an ecosystem to return to an equilibrium or steady-state following a perturbation
  
  – as "the capacity of a system to absorb disturbance and reorganize while .... still retain essentially the same function".

---

Characteristics of resilient systems

Critical aspects of resilience:

1. **LatITUDE**: the maximum amount a system can be changed before losing its ability to recover.

2. **Resistance**: the ease or difficulty of changing the system; how “resistant” it is to being changed.

3. **Precariousness**: how close the current state of the system is to a limit or “threshold.”

4. **Panarchy**: the degree to which a certain hierarchical level of an ecosystem is influenced by other levels.
Resilience vs. dependability vs. robustness?

- ** Dependability ** Ability of a system to deliver service that can justifiably be trusted.

  (Ability of a system to avoid failures that are more frequent or more severe than is acceptable to user(s))

- ** Survivability ** is the ability to remain alive or continue to exist.

- ** Robustness ** is the persistence of a system’s characteristic behavior under perturbations or conditions of uncertainty.

- ** Resilience ** is the persistence of the avoidance of failures that are unexpectedly frequent or severe, when facing change [Laprie]
Resilience vs. robustness
Resilience vs. dependability vs. robustness?

Highly controlled
Operates in a narrow band
Predefined states ("modes")
Top-down design
Challenge: predict all states caused by the environment

A broad spectrum of possible equilibrium state
Not necessary all states are predicted
Adaptive and evolving systems
impact of the system on the environment
Challenge:
  Adaptation
  Optimal performance in different states
  Minimize unwanted impact on the environment
Do we care about impact on the environment?

**ABB vision:** “Power and productivity for a better world”

“….., we help our customers to use electrical power efficiently, to increase industrial productivity and to lower environmental impact in a sustainable way.”
Resilient system development challenges

Resilience is an extra-functional (non-functional) property

• A part of the design:
  – System adaptability
  – System evolvability
  – Impact on external environment
    • (direct & indirect)
    • (short & long term)
Extended development process

Development process – top-down vs. Bottom-up

• Modern, sustainable and resilient systems:
  – Complex, distributed, highly dynamic, highly adaptive
  – Adjustable to the changing environment and to the users

• Top-down principle is not enough!
  – It would be a problem with evolution and flexibility

• Late compositions, reconfigurations, on-line verifications, on-line validations
Challenges - System boundaries and bottom-up

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

Product line
Known: domain, architectural framework, application skeleton, variation points
Unknown: Final products

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

Final products ready to use
(usage not necessary known)

Final product in use

Challenges:
• What can we predict (or guarantee) about the system run-time properties?

Indirect impact of the product
Challenges of compositions

Component = <{Interfaces}, {Properties}>  ( C= <I,P>)

Composition:  \[ A = \langle C_1 \oplus C_2 \rangle \Rightarrow I = \langle I_1 \oplus I_2 \rangle \land P = \langle P_1 \oplus P_2 \rangle \]

- Interface composition (BINDING): \[ I(C) = I(C_1) \oplus I(C_2) \]
- Property composition: \[ P_i(C) = P_i(C_1) \oplus P_i(C_2) \]

\[ P(C) = P_1(C_1) \oplus P_2(C_2) \]

Is it possible (i.e. predict) to calculate P from P1 and P2?
Is it possible to predict a quality attribute of a system from quality attributes of components?

Ivica Crnkovic, Magnus Larsson, Otto Preiss, Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes, Architecting Dependable Systems III,, p pp. 257 – 278, Springer, LNCS 3549, Editor(s): R. de Lemos et al. (Eds.);, 2005

Resilient systems: Challenges of compositions

\[ P(C) = P_1(C_1) \oplus P_2(C_2) \]

• Which properties are important for resilience?

• Can we predict the compositions of these properties?
Some definition first...

- **Component**
- **Assembly** – a set of components
- **System**
- **System Usage**
- **System context**
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-dependend 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.
Extra Functional Properties – composition types (I)

1. *Directly composable properties.* A property of an assembly is a function of, and only of, the same property of the components involved.

   \[ P(A) = f(P(C_1), \ldots P(C_i), \ldots, P(C_n)) \]

   – Example: Static memory

2. *Architecture-related properties.* A property of an assembly is a function of the same property of the components and of the software architecture.

   \[ P(A) = f(SA, \ldots P(C_i)\ldots), \ i=1\ldots n \]

   – SA = software architecture

   – Example: Performance in relation to a number of clients and servers
3 **Derived properties.** A property of an assembly depends on several different properties of the components.

- \( P(A) = f(SA, ...Pi(C_j)...), \ i=1...m, \ j=1...n \)
- \( Pi = \text{component properties} \)
- \( C_j = \text{components} \)

Example

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

fixed priority scheduling

\[
L^{n+1}(c_i) = c_i \cdot wcet + B(c_i) + \sum_{c_j \in P(c_i)} \left[ \frac{L^n(c_j)}{c_j \cdot T} \right] \cdot \text{wcet}
\]
Extra Functional Properties – composition types (III)

4. **Usage-dependent properties.** A property of an assembly is determined by its usage profile.
   - \( P(A,U) = f(SA, \ldots Pi(C_j,U), \ldots), i=1\ldots m, j=1\ldots n \)
   - \( U = \) Usage profile
   
   - **Example:** Reliability, Availability

5. **System environment context properties.** A property is determined by other properties and by the state of the system environment.
   - \( P(S,U,X) = f(SA, \ldots Pi(C_j,U,X), \ldots), i=1\ldots m, j=1\ldots n \)
   - \( S = \) system, \( X = \) system context
   
   - **Example:** Safety, Security
   - **Resilience??** (challenge: context definition)
How can we assess the resilience?
Inspiration: Quality model in ISO 9126-I

Example

having source code reviews” (a Software development process quality) influences the source code in that “the number of not initialized variables” (an internal quality attribute of a software product) is minimized. This positively influences the reliability, of the system (an external quality attribute of a software product).
General Concepts of the ISO/IEC 9126-1
Resilience property model
(in a spirit of ISO/IEC 9126-1)

Questions:
which are subcharacteristics?
which measuring attributes?
How to design for resilience?
*How to analyze resilience?*
Analysis Process

Related to Software Architecture

Change stimuli

Elicit Architectural Concerns

Stakeholders’ concerns (subcharacteristics)

Analyze Implication of change stimuli

Potential architectural requirements

Propose architectural solutions

Analyze Implications to the subcharacteristics
Example: Software Evolvability Model

- Literature survey and analysis of the literature
- Industrial case studies

Evolvability

Subcharacteristics

- Analyzability
- Architectural Integrity
- Changeability
- Portability
- Extensibility
- Testability
- Domain-specific attributes

Evolvability is refined to Subcharacteristics
Subcharacteristics are measured by Metrics
Metrics are related to QoS

Software Architecture Evolvability Analysis (AREA) Process

Elicit Architectural Concerns

Stakeholders' Concerns

Analyze Implications of Change Stimuli

Potential Architectural Requirements

Propose Architectural Solutions

Candidate Architectural Solutions

Assess Architectural Solutions

Legend

Information Flow

Common Activity

Method-specific Activity

Qualitative Architecture Evolvability
Analysis Method

Phase 1: Analyze the implications of change stimuli on software architecture
Step 1.1: Identify potential requirements on the software architecture
Step 1.2: Prioritize potential requirements on the software architecture

Phase 2: Analyze and prepare the software architecture to accommodate change stimuli and potential future changes
Step 2.1: Extract architectural constructs related to the respective identified requirement
Step 2.2: Identify refactoring components for each identified requirement
Step 2.3: Identify and assess potential refactoring solutions from technical and business perspectives
Step 2.4: Define test cases

Phase 3: Finalize the evaluation
Step 3.1: Analyze and present evaluation results
Case Study: ABB Robotic Control System

A conceptual view of the software architecture
Architectural concerns

• Analyzability
• Architectural Integrity
• Changeability
• Portability
• Extensibility
• Testability
• Domain-specific attributes
  – Performance, real-time
Phase 1: Change Stimuli and Architectural Requirements

Change Stimuli

- Time-to-market requirements
- Increased ease and flexibility of distributed development of diverse application variants.

Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Transform the monolithic architecture to modular architecture.</td>
</tr>
<tr>
<td>R2</td>
<td>Reduced architecture complexity.</td>
</tr>
<tr>
<td>R3</td>
<td>Enable distributed development of extensions with minimum dependency.</td>
</tr>
<tr>
<td>R4</td>
<td>Portability.</td>
</tr>
<tr>
<td>R5</td>
<td>Impact on product development process.</td>
</tr>
<tr>
<td>R6</td>
<td>Minimized software code size and runtime footprint.</td>
</tr>
</tbody>
</table>

R1, R2, R3 prioritized
Phase 2: Architectural analysis

- Extract architectural constructs related to the respective identified issues
- Identify refactoring components for each identified issue

Refactoring components – dynamic component binding
Phase 3: finalize the analysis

Example: Implications of the IPC component refactoring on evolvability subcharacteristics (+ positive impact, - negative impact)

<table>
<thead>
<tr>
<th>Subcharacteristics</th>
<th>IPC Component Refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzability</td>
<td>– due to less possibility of static analysis since definitions are defined dynamically</td>
</tr>
<tr>
<td>Architectural Integrity</td>
<td>+ due to documentation of specific requirements, architectural solutions and consequences</td>
</tr>
<tr>
<td>Changeability</td>
<td>+ due to the dynamism which makes it easier to introduce and deploy new slots</td>
</tr>
<tr>
<td>Portability</td>
<td>+ due to improved abstraction of Application Programming Interfaces (APIs) for IPC</td>
</tr>
<tr>
<td>Extensibility</td>
<td>+ due to encapsulation of IPC facilities and dynamic deployment</td>
</tr>
<tr>
<td>Testability</td>
<td>No impact</td>
</tr>
<tr>
<td>Domain-specific attribute</td>
<td>+ resource limitation issue is handled through dynamic IPC connection</td>
</tr>
<tr>
<td></td>
<td>– due to introduced dynamism, the system performance could be slightly reduced</td>
</tr>
</tbody>
</table>
Quantitative Architecture Evolvability Analysis Method

Using **Analytic Hierarchy Process (AHP)** for comparison of choices

**Phase 1: Analyze the implications of change stimuli on software architecture**
- Step 1.1: Elicit stakeholders’ views on evolvability subcharacteristics
- Step 1.2: Extract stakeholders’ prioritization and preferences of evolvability subcharacteristics

**Phase 2: Analyze and prepare the software architecture to accommodate change stimuli and potential future changes**
- Step 2.1: Identify candidate architectural solutions
- Step 2.2: Assess candidate architectural solutions’ impacts on evolvability subcharacteristics

**Phase 3: Finalize the evaluation**
- Step 3.1: Analyze and present evaluation results
Analytic Hierarchy Process (AHP)

\[ [m_{ij}] \text{ nxn matrices} \quad \text{– comparison elements (subcharacteristics)} \]

\[ m_{ij} \text{ – comparison values between two elements} \quad (1 \ldots 9 : 1) \]
\( \text{(results – binary relations between the elements)} \)

Calculation and value normalization -> normalized significance of the values \( W_i \)
\( (W_1+W_2+\ldots W_n = 1) \)
Case study: Ericsson

- mobile network architecture with respect to the evolvability of a logical node at Ericsson.
- Logical Node:
  - handle control signaling for and keep track of user equipment such as mobiles using a certain type of radio access.
Phase 1: requirements identification

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Subcharacteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1. The ISSU solution should be easy to understand for the development organization.</td>
<td>Analyzability</td>
</tr>
<tr>
<td>R2. Many kinds of application changes shall be possible without special upgrade code, e.g., backward compatible interfaces.</td>
<td>Changeability</td>
</tr>
<tr>
<td>R3. Enable introduction of ISSU solution without limitations on extending existing features or adding new ones.</td>
<td>Extensibility</td>
</tr>
<tr>
<td>R4. Enable change of operating system or hardware.</td>
<td>Portability</td>
</tr>
<tr>
<td>R5. Enable the ease to test and debug parts of the system individually, and extract test data from the system.</td>
<td>Testability</td>
</tr>
<tr>
<td>R6. The ISSU total time, capacity impact during ISSU and normal execution should be within specified values.</td>
<td>Capacity</td>
</tr>
<tr>
<td>R7. Critical components need to have redundancy during ISSU. R8. The impact of software or hardware failure during upgrade should be limited.</td>
<td>Availability</td>
</tr>
</tbody>
</table>
Preferences of evolvability subcharacteristics (1)

Preferences on evolvability subcharacteristics provided by a software architect

<table>
<thead>
<tr>
<th></th>
<th>Analyzability</th>
<th>Integrity</th>
<th>Changeability</th>
<th>Extensibility</th>
<th>Portability</th>
<th>Testability</th>
<th>Availability</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzability</td>
<td>1</td>
<td>3</td>
<td>1/3</td>
<td>1/3</td>
<td>5</td>
<td>1</td>
<td>1/4</td>
<td>1/3</td>
</tr>
<tr>
<td>Integrity</td>
<td>1/3</td>
<td>1</td>
<td>1/6</td>
<td>1/6</td>
<td>2</td>
<td>1/3</td>
<td>1/8</td>
<td>1/7</td>
</tr>
<tr>
<td>Changeability</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1/3</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>Extensibility</td>
<td>3</td>
<td>6</td>
<td>1/2</td>
<td>1</td>
<td>3</td>
<td>1/3</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>Portability</td>
<td>1/5</td>
<td>1/2</td>
<td>1/5</td>
<td>1/3</td>
<td>1</td>
<td>1/7</td>
<td>1/9</td>
<td>1/8</td>
</tr>
<tr>
<td>Testability</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1/3</td>
<td>1/2</td>
</tr>
<tr>
<td>Availability</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Capacity</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Preferences | 0.077 | 0.030 | 0.135 | 0.110 | 0.023 | 0.158 | 0.249 | 0.219 |
## Preferences of evolvability subcharacteristics (2)

<table>
<thead>
<tr>
<th>Stakeholders</th>
<th>Analyzability</th>
<th>Integrity</th>
<th>Changeability</th>
<th>Extensibility</th>
<th>Portability</th>
<th>Testability</th>
<th>Availability</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architects</td>
<td>0.073</td>
<td>0.038</td>
<td>0.105</td>
<td>0.090</td>
<td>0.039</td>
<td>0.128</td>
<td>0.322</td>
<td>0.205</td>
</tr>
<tr>
<td>Designers</td>
<td>0.105</td>
<td>0.125</td>
<td>0.103</td>
<td>0.108</td>
<td>0.042</td>
<td>0.154</td>
<td>0.322</td>
<td>0.041</td>
</tr>
<tr>
<td>System owner</td>
<td>0.061</td>
<td>0.189</td>
<td>0.111</td>
<td>0.108</td>
<td>0.023</td>
<td>0.112</td>
<td>0.350</td>
<td>0.046</td>
</tr>
<tr>
<td>Aggregated</td>
<td>0.080</td>
<td>0.117</td>
<td>0.106</td>
<td>0.102</td>
<td>0.035</td>
<td>0.131</td>
<td>0.331</td>
<td>0.098</td>
</tr>
</tbody>
</table>

### Consistency ratios for stakeholders

<table>
<thead>
<tr>
<th>Stakeholders</th>
<th>Architect A</th>
<th>Architect B</th>
<th>Architect C</th>
<th>Designers</th>
<th>System Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
<td>0.061</td>
<td>0.109</td>
<td>0.039</td>
<td>0.088</td>
<td>0.046</td>
</tr>
<tr>
<td>Ratio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CR < 0.1 – the consistent selections between stakeholders
Phase 2: provide alternative architectural solutions

<table>
<thead>
<tr>
<th></th>
<th>Alt 1: Slot by slot concept</th>
<th>Alt 2: Zone concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzability</td>
<td>0.667</td>
<td>0.333</td>
</tr>
<tr>
<td>Integrity</td>
<td>0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>Changeability</td>
<td>0.250</td>
<td>0.750</td>
</tr>
<tr>
<td>Extensibility</td>
<td>0.333</td>
<td>0.667</td>
</tr>
<tr>
<td>Portability</td>
<td>0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>Testability</td>
<td>0.333</td>
<td>0.667</td>
</tr>
<tr>
<td>Availability</td>
<td>0.750</td>
<td>0.250</td>
</tr>
<tr>
<td>Capacity</td>
<td>0.800</td>
<td>0.200</td>
</tr>
</tbody>
</table>

Select alternative: \( W_{Alt} = \sum_{i=1}^{k} (PQ_i \times PA_{m,q_i}) \)

\( W_{Alt1} = 0.080 \times 0.667 + 0.117 \times 0.500 + 0.106 \times 0.250 + 0.102 \times 0.333 + 0.035 \times 0.500 + 0.131 \times 0.333 + 0.331 \times 0.750 + 0.098 \times 0.800 = 0.560 \)

\( W_{Alt2} = 0.440 \)
Summary - Questions

• What is the importance of resilience
  – Is resilience really of interest for our industrial partners?

• How to include resilience aspects in the development process?
  – In certification?
  – In real-time systems?
  – Component-based approach?
  – Model-driven approach?

• How to verify resilient systems?
  – Can we test all possible scenarios?

• Adaptability
  – Where to implement the adaptability?
    • Source code level?
    • Architectural level?