The Role of Requirements and Specification in Product Line Engineering

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Change of paradigms in software & systems engineering

- Model based development
- Front Loading
  ◦ Stronger emphasis auf early phases: requirements, specification, architecture
  ◦ early quality assurance
- Domain engineering
  ◦ concentration auf domain specific aspects
  ◦ use cases
- Artefact orientation
  ◦ storage each artefact in data base
  ◦ relationship between views produce (tracing)
- Systematic generation of software
  ◦ target platforms
- Software- & system evolution - maintenance
- Product lines
  ◦ systematic reuse
Dimensions in the development of systems

- determining system boundaries
  - interfaces
  - embedding/context
- refinement
  - adding of properties - requirements completion
  - adding of details - design und implementation
  - change of granularity (time/action/interaction)
- decomposition
  - from system level to sub-system - component hierarchy
  - from integrated system to mechanics/electronics/software
- levels of abstraction
  - functionality
  - architecture
  - technical implementation
- perspective
  - domain specific
  - logical/functional
  - technical
- views
  - data
  - state
  - structure
  - time
  - process
  - interaction
- further dimension: variability
Success factors: adequate models and precise terminology

• Comprehensive system architecture
  ◦ function
  ◦ dependency between functions (feature interactions)
  ◦ modularity
  ◦ architecture
  ◦ context models: assumption/promise
  ◦ tracing

• compatibility
  ◦ substitutivity
  ◦ refinement
  ◦ interoperability

• artefact models: consistency in models
  ◦ product data modelling

• development process models
  ◦ roles
  ◦ artefacts
  ◦ processes

• quality models
  ◦ quality taxonomy + dependencies between quality attributes
Software & systems engineering (S&SE) and product lines engineering (PLE)

• Key notions in S&SE
  ◇ requirements
  ◇ architectures
  ◇ quality

• Key notions in PLE
  ◇ Feature modeling
  ◇ Domain engineering
  ◇ Variability

• PLE can be seen as the higher art of software & systems engineering (S&SE)
  ◇ introduces a new dimension into S&SE
  ◇ if classical S&SE disciplines are not mastered, then there is no way to master PLE
  ◇ PLE addresses the key notions in S&SE
Chapter 1: A system model
Cyber-Physical Systems:
Real World Awareness

Physical World

Embedded System

Cyberspace Services & Data

CPS

MMI

Physical MMI

Embedded System
What is a (discrete) system

A system has

• a **system boundary** that determines
  ◦ what is part of the systems and
  ◦ what lies outside (called its **context**)

• an **interface** (determined by the system boundary), which determines,
  ◦ what ways of interaction (**actions**) between the system and its context are possible (**static** or **syntactic interface**)
  ◦ which behaviour the system shows from view of the context (**interface behaviour, dynamic interface, interaction view**)

• an **internal structure**, given
  ◦ by its structuring in **sub-systems** (**architecture**)
  ◦ by its states and state transitions (**state view, state machines**)

• the structuring view, state view, and the interaction view use a **data model**

• the views may be documented by adequate **models**
Sets of typed channels

\[ I = \{ x_1 : T_1, x_2 : T_2, \ldots \} \]
\[ O = \{ y_1 : T'_1, y_2 : T'_2, \ldots \} \]

Syntactic interface

\[ (I \leadsto O) \]

Data stream of type \( T \)

\[ \text{STREAM}[T] = \{ \text{IN} \rightarrow T^* \} \]

Valuation of channel set \( Z \)

\[ \text{IH}[Z] = \{ Z \rightarrow \text{STREAM}[T] \} \]

Interface behaviour for syn. interface \((I \leadsto O)\)

\[ [I \leadsto O] = \{ \text{IH}[I] \rightarrow \varnothing (\text{IH}[O]) \} \]

Interface specification

\[ p : I \cup O \rightarrow IB \]

Represented as interface assertion \( S \)

Logical formula with channel names as variables for streams
Structuring concept for system model: architectural levels of abstraction

<table>
<thead>
<tr>
<th>requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface assertion</td>
</tr>
<tr>
<td>R_1</td>
</tr>
<tr>
<td>R_2</td>
</tr>
<tr>
<td>R_n</td>
</tr>
</tbody>
</table>

context

functionality
functions and function hierarchy
context
modes

Component Architecture
sub-system interface behavior
connection structure

Technical and physical architecture

traces, links, relationships
Composition and decomposition of systems and functions

\[
F_1 \in IF[I_1 \triangleright O_1] \\
F_2 \in IF[I_2 \triangleright O_2] \\
Z_1 = O_1 \cap I_2 \\
Z_2 = O_2 \cap I_1 \\
I = I_1 \setminus Z_2 \cup I_2 \setminus Z_1 \\
O = O_1 \setminus Z_1 \cup O_2 \setminus Z_2 \\
F_1 \otimes F_2 \in IF[I \triangleright O], \\
(F_1 \otimes F_2) \cdot x = \{z|O: x = z|I \land z|O_1 \in F_1(z|I_1) \land z|O_2 \in F_2(z|I_2)\}\]
Modularity: composition rule for interface specifications

\[ \text{F1} \times \text{F2} \]

\begin{align*}
\text{F1} & \quad \text{in} \quad x_1, z_{21}: T \\
& \quad \text{out} \quad y_1, z_{12}: T \\
\text{S1} & \\
\text{F2} & \quad \text{in} \quad x_2, z_{12}: T \\
& \quad \text{out} \quad y_2, z_{21}: T \\
\text{S2} &
\end{align*}

\[ \exists z_{12}, z_{21}: \text{S1} \land \text{S2} \]
What is a function?

What is a component?

Sub-function $\neq$ Sub-system
Sub-types und projection auf interfaces

• A set $Z'$ of typed channels is called **subtype** of a set $Z$ of typed channels if:
  ◊ $Z'$ is sub-set of $Z$
  ◊ the data types of channel set $Z'$ are subsets of resp. data types of channel set $Z$

we write

$$Z' \text{ subtype } Z$$

• Let $x \in IH[Z]$ then the restriction

$$x|Z' \in IH[Z']$$

is called **projection** of $x$ onto $Z'$
Sub-types between interfaces

For syntactic interfaces \((I \rightarrow O)\) and \((I' \rightarrow O')\) where

\[I' \text{ subtype } I \text{ and } O' \text{ subtype } O\]

we call \((I' \rightarrow O')\) a sub-type of \((I \rightarrow O)\) and write:

\[(I' \rightarrow O') \text{ subtype } (I \rightarrow O)\]
Projection onto interfaces

Given

\[(I' \triangleright O') \text{ subtype } (I \triangleright O)\]

for \( F \in \text{IF}[I \triangleright O] \) the projection

\[F^\dagger (I' \triangleright O') \in \text{IF}[I' \triangleright O']\]

of \( F \) onto the syntactic interface \((I' \triangleright O')\) (for all \( x' \in \text{IH}[I'] \)) is given by:

\[F^\dagger (I' \triangleright O')(x') = \{ y | O' : \exists x \in \text{IH}[I] : x' = x|I' \land y \in F(x) \}\]

The projection is called \textit{faithful}, if for all \( x \in \text{dom}(F) \)

\[F(x)|O' = (F^\dagger (I' \triangleright O'))(x|I')\]
Composing functions
Combining Functions

Given two functions $F_1$ and $F_2$ in isolation

We want to combine them into a function $F_1 \oplus F_2$
Combining Functions

Their isolated combination

\[ F_1 \oplus F_2 \]

- \( I_1 \) to \( F_1 \)
- \( O_1 \) from \( F_1 \)
- \( I_2 \) to \( F_2 \)
- \( O_2 \) from \( F_2 \)
Combining Functions

If the two services $F_1$ and $F_2$ have feature interaction we typically get:

We explain the functional combination $F_1 \otimes F_2$ as a refinement step.
The steps of function combination

Given the isolated function $F_1$

We construct a refinement $F'_1$

And combine $F'_1$ with a refinement $F'_2$ of $F_2$
Function Hierarchy

\[ F_1, \ldots, n \rightarrow B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_n \]

\[ F_{1,2} \rightarrow B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_n \]

\[ F_{k,k+1} \rightarrow B_{k-1} \rightarrow B_k \rightarrow B_{k+1} \rightarrow \ldots \rightarrow B_n \]

\[ F_{n-1,n} \rightarrow B_{n-1} \rightarrow B_n \]

subservice relation

channels of mode types
A refined model of behavior

• A model of functional correctness - a qualitative model

\[ F_I : \text{IH}[I] \rightarrow \wp(\text{IH}[O]) \]

• A model of probability - quantitative model

\[ F_N : \text{IH}[I] \rightarrow \{ (\mu, X) : X \subseteq \text{IH}[O] \} \]

where \((\mu, X)\) denotes a probability distribution \(\mu\) over set \(X\)
Chapter 2: PLE & RE
The role of requirements in product lines engineering (PLE)

• Products in product lines should be selected by choosing from requirements
  ◊ Requirements driven PLE
  ◊ Which artifacts for choosing RE

• Two levels of RE in PLE
  ◊ RE for a PLE (RE4PLE)
  ◊ RE for a product instance (PI) for a PLE (RE4PI@PLE)

• Key questions
  ◊ Adequate structuring of system requirements for PLE
  ◊ Structuring functionality

• What we do not consider
  ◊ Project requirements
  ◊ Constraints
Variability and System Functionality

- System requirements
  - Structuring system requirements
  - Consistency
- System functionality
  - System hierarchies: hierarchies of system sub-functions
  - Interdependencies, dependencies (mutual dependencies)
- System architecture
  - Interface specification
  - Composition
  - Interaction
- Connecting and linking
  - Mutual compatibility (interdependencies of views)
  - Completeness (self containedness)
  - Refinement (correctness)
  - Traceability (tracing)
- Variability
  - Tracing, linking
Variability in PLE

• Variability in the functionality and functional requirements
  ◊ Variability in the set of functions
    • different choice of functions
    • managing dependencies
  ◊ Variability within functions
    • Variability in MMI
      – logical
      – physical
    • Variability w.r.t. context
      – sensors
      – actuators
      – controlled system

• Variability in non-functional requirements
  ◊ Product requirements
    • Quality
    • Implementation concepts
IEEE Requirements Quality Attributes and its role for PLE

• An SRS should be
  a) Correct
  b) Unambiguous
  c) Complete
  d) Consistent
  e) Ranked for importance/stability
  f) Verifiable
  g) Modifiable
  h) Traceable

• Which of these requirements quality attributes are of particular importance for PLE
  ◊ RE4PLE
  ◊ RE4PI@PLE

• How does variability influence the requirements quality attributes

• An RE4PLE should be
  i) Modifiable/Extendable
  j) Generically Consistent
  k) Include variability
  l) Generically Verifiable
  m) Traceable

• An RE4PLE should be
  ◊ generated schematically out of the generic requirements of RE4PLE
How well is RE understood?

State of the art in RE:
- Standardized processes?
- Standardized artifacts?
- Standardized roles?
- Standardized theory?
- Standardized methods?
- Standardized tools?
- Standardized terminology?
- Model-based RE?
- Best practice?

Out of the 10 most success critical factors of S&SE 5 are related to RE!
Classifying Requirements

Classical (following Glinz)
• Functional requirements:
  Functionality and behavior:
  Functions, Data, Stimuli, Reactions, Behavior
• Attribute
  ◇ Performance requirements:
    Time and space bounds: Timing, Speed, Volume, Throughput
  ◇ Specific quality requirements “-ilities”: Reliability, Usability, Security, Availability, Portability, Maintainability, ...
• Constraint:
  Physical, Legal, Cultural, Environmental, Design&Implementation, Interface, ...

Advanced - rethinking classifications
• Behavioral
  ◇ Qualitative:
    functional correctness
  ◇ Quantitative:
    • Performance requirement:
      Time and space bounds: Timing, Speed, Volume, Throughput
    • Reliability, Security, Availability,
• Representational
  ◇ Usability, Portability, Maintainability, ...
  ◇ Constraint:
    Physical, Legal, Cultural, Environmental, Design&Implementation, Interface, ...
Project risks in RE

• Invalid requirements
  ◇ Incorrect requirements
  ◇ Superficial requirements
• Incomplete, imprecise requirements
• Implementation biased requirements
• Instability of requirements
  ◇ Instable requirements are the perhaps most significant risk in RE
  ◇ Flexibility versus instability
    • controlled instability
• Methodological deficiencies in RE
  ◇ Unstructured documentation
  ◇ Fuzzy formulation
  ◇ Insufficient traceability
Requirements engineering (RE) for PLE

- Concepts to achieve variability
  - Parameterization
  - Flexibility of composition
  - Schematic choice
  - Points of variations
  - Adding pre-/postprocessing

- Componentware
- Frameworks
- PLA

- Variability for which aspect of requirements needed
  - What is the appropriate level for variability
  - Which categories of requirements are addressed by variability

- Which concepts of variability for which aspect
Chapter 3: A logic based formalization of tracing
Levels of abstractions in S&SE

• Requirements

• System views

  ◊ Functionality
    • Functions and function hierarchy
    • Context
      Physical
      MMI
    • Modes

  ◊ Sub-systems: Component Architecture
    • sub-system interface behavior
    • connection structure

  ◊ Technical and physical architecture
    • MMI
    • Software
    • Hardware
    • Software/hardware codesign
How to capture semantics: logics

Basic approach:
• develop theories in terms of
  ◊ mathematics and
  ◊ logics
• that capture essential concepts in software & systems engineering
• reflect the terminology
• prove or disprove ideas and concepts
• validate/verify methods
• can be the basis of automation
  ◊ tool support
• support views
  ◊ abstraction
  ◊ structuring
• are powerful enough to support comprehensive system modelling
On the theory of modelling

- Logic as a basis and an instrument of engineering
  - system properties as logical propositions
  - usage of logical terminology
    - consistency
    - completeness
    - independence
    - redundancy
    - ...
- Reductionistic presentation of phenomena and concepts of systems & software engineering by logic and mathematics
  - Occam’s Razor (lex parsimoniae): a theory is in its structure and inner relationships as simple as possible to construct
- Comprehensive capturing and representation of all relevant system properties and their logical mutual dependencies
System views and viewpoints - a strictly logical approach

- A viewpoint is given by a stakeholder or in terms of a specific aspect
  - defines a perspective onto the system
  - Example: user viewpoint
- A viewpoint leads to a view
  - a view represents a particular abstraction of a system
  - serves a certain purpose
  - Example: functional view
- Views can be represented by models and logics
  - each model/view can be represented by logical assertions
  - the models provide the model theory as a basis for the logical theory
- having logical assertions representing views we can
  - analyse views by analysing set of logical assertions
  - relate views in terms of relating set of logical assertions
    - consistency
    - completeness
    - refinement
Fundamental views onto systems

• Interface view - system functionality
  ◊ structured in function hierarchy

• Architecture view - system design
  ◊ structured in component hierarchy

• State view
  ◊ structured in state hierarchy
  ◊ abstraction in operational states

• Process view
  ◊ structured in trace hierarchy
System level: requirement specification

The assertions $A_i$ describe the requirements by the system interface behaviour.
Example: System level requirements specification

A transmission component **FairMIX**

```
FairMIX

<table>
<thead>
<tr>
<th>in</th>
<th>x, z: T</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y: T</td>
</tr>
</tbody>
</table>

∀ m ∈ T: {m}#x + {m}#z = {m}#y
∧ ∀ t ∈ IN, m ∈ T: {m}#x↓t + {m}#z↓t ≥ {m}#y↓(t+n)
```
Architectural view: decomposition into components

- architecture specification (given by component interface specifications $C_i$)

$$C = \bigwedge \{C_i : 1 \leq i \leq k\}$$
Logics: three levels of system specification

• System level requirements (we concentrate on functional requirements)
  ◊ a list of requirements in terms of system properties

• System level functional specification
  ◊ a decomposition of the system functionality into a hierarchy of sub-functions
  ◊ a specification of the sub-functions by properties
  ◊ feature interactions specified via a mode concept

• Architecture specification
  ◊ a decomposition of the system into a sub-systems (components)
  ◊ their connections via their sub-interfaces
  ◊ interface specification by interface properties
Three levels of specification: three examples

• System level requirements (functional requirements)
  ◊ “the car must not accelerate its speed without users control”

• System level functional specification
  ◊ “the acc (adaptive cruise control) accelerates the car up to the speed selected by the user, provided no obstacles are recognized in front”

• Architecture specification
  ◊ “the radar signal based sensor measures the distance in m/s to the car in front and sends it to the acc monitor every 100 ms”
System level functional requirements

- The system interface behaviour $F$ is specified by the system requirements specification

$$A = \{A_i: 1 \leq i \leq n\}$$

where the $A_i$ are interface assertions

<table>
<thead>
<tr>
<th>Function</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>...</td>
<td>Yes</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>$A_2$</td>
<td>...</td>
<td>No</td>
<td>medium</td>
<td></td>
</tr>
<tr>
<td>$A_n$</td>
<td>...</td>
<td>no</td>
<td>low</td>
<td></td>
</tr>
</tbody>
</table>
Functional view: functional decomposition

• The system interface behaviour \( F \) as specified by the system requirements specification
  \[ A = \{ A_i : 1 \leq i \leq n \} \]
is structured
  ◆ into a set of sub-interfaces for sub-functions \( F_1, \ldots, F_k \)
  ◆ that are specified independently by introducing a number of mode channels to capture their feature interactions
  ◆ each \( F_i \) sub-function is described by
    • a syntactic interface and
    • an interface assertion \( B_i \) such that
  \[ \wedge \{ B_i : 1 \leq i \leq k \} \Rightarrow A \]
Architecture: decomposition into components

• Composition $F = F_1 \otimes F_2 \otimes F_3$

$\begin{align*}
F_1: & \ x_1 : T_1 \\
& \ y_6 : T'_6 \quad x_6 : T_6 \\
& \ y_8 : T'_8 \quad x_8 : T_8
\end{align*}$

$\begin{align*}
F_2: & \ x_2 : T_2 \\
& \ y_7 : T'_7 \quad x_7 : T_7 \\
& \ y_8 : T'_8 \quad x_8 : T_8
\end{align*}$

$\begin{align*}
F_3: & \ y_6 : T'_6 \quad x_6 : T_6 \\
& \ y_7 : T'_7 \quad x_7 : T_7 \\
& \ x_5 : T_5 \quad y_5 : T'_5
\end{align*}$

$\begin{align*}
F: & \ x_1 : T_1 \\
& \ y_4 : T'_4 \quad x_4 : T_4 \\
& \ x_5 : T_5 \quad y_5 : T'_5
\end{align*}$
Variability in function hierarchies

- The functionality multi-functional system can be specified by a function hierarchy
  - a modular specification of sub-functions is achieved by a mode concept
- Variability can be introduced by modes
- Variability in function hierarchies leads to a kind of interpreted FODA trees:
  - FODA trees with behavioral specifications
  - FODA trees with dependencies
**Modes for modularity**

- A mode concept is an abstract state model for a system, such that for each sub-function
  - a projection of the mode space determines the behavior of the functions
  - use cases can be selected for the different modes
  - the dependencies between the sub-functions are captured via the modes
- Modes for PLE
  - a general mode concept with variability is needed
  - for a product instance a consistent mode space is derived
  - which modes are required can be seen by the selected functions
Three levels of Specification

• Requirements - system level
  ◦ List of requirements - functional system property
  ◦ Example: „The activation of safety relevant functions by the system is always double checked for plausibility.“

• Functional specification - system level
  ◦ decomposition of System functionality in hierarchy of (sub-)functions
  ◦ Specification of (sub-)functions
  ◦ Specification of dependencies (feature interactions) between (sub-)functions based on a mode concept
  ◦ Example: „Thrust reversal may only be activated, if the plane is on the ground.“

• Architecture specification - component level
  ◦ decomposition a Systems in sub-systems (component)
  ◦ relationship to data flow diagram
  ◦ interface specification of component
  ◦ Example: “The weight sensor indicates that the plane is on the ground.“
Relational view: Tracing

<table>
<thead>
<tr>
<th>Function</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>...</td>
<td>Yes</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>A_2</td>
<td>...</td>
<td>No</td>
<td>medium</td>
<td></td>
</tr>
<tr>
<td>A_n</td>
<td>...</td>
<td>no</td>
<td>low</td>
<td></td>
</tr>
</tbody>
</table>
Relating logical views

Let $p$ be a properties and $R$ be a set of properties
a subset $R' \subseteq R$ is called guarantor for $p$ in $R$ if
\[ \land R' \Rightarrow p \]

• Classifying guarantors
  ◊ A guarantor $R'$ for $p$ is called minimal, if every strict subset of $R'$ is not a guarantor
  ◊ a minimal guarantor is called unique if there does not exist a different minimal guarantor.

• Classifying properties
  ◊ A property $q \in R$ is called weak guarantor for $p$ in $R$ if it occurs in some minimal guarantor of $p$ in $R$
  ◊ A property $q \in R$ is called strong guarantor for $p$ in $R$ if it occurs in every guarantor of $p$ in $R$

◊ Cf. Primimplikanten a la Quine
Three levels of system description in logic

• system level requirements
  \[ A = \land \{A_i : 1 \leq i \leq r\} \]

• functional specification at system level - functionality
  \[ B = \land \{B_i : 1 \leq i \leq n\} \]

• architecture specification (given by component interface specifications)
  \[ C = \land \{C_k : 1 \leq k \leq m\} \]

• correctness
  functional Specification correct in relation auf requirements:
  \[ B \Rightarrow A \]
  architecture (let \( m_1, \ldots, m_n \) be mode channels):
  \[ C \Rightarrow \exists m_1, \ldots, m_j : B \]
Applying projections - functional slicing

Identifying sub-functions - functional slicing sub-interface behaviour

Given some behaviour

\[ F \in [I \rightarrow O] \text{ with interface assertion } B \]

a set of behaviours

\[ F_k \in [I'_k \rightarrow O'_k] \text{ with interface assertion } B_k \]

with

\[ [I_k \rightarrow O_k] \text{ subtype } [I'_k \rightarrow O'_k] \]

\[ I = \cup \{ I_k : 1 \leq k \leq m \}, \quad O \cup \{ O_k : 1 \leq k \leq m \} \]

is called a functional decomposition if

\[ F = \otimes \{ F_k : 1 \leq k \leq m \} \text{ with interface assertion } B = \wedge B_k \]
Architecture

• Composition \( F = C_1 \otimes C_2 \otimes C_3 \)
**Relationship: req spec to function spec - tracing**

<table>
<thead>
<tr>
<th></th>
<th>system level reqs</th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>...</th>
<th>...</th>
<th>A_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub-function reqs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Red:** B_i is strong guarantor of A_j

**Yellow:** B_i is weak guarantor of A_j

**Green:** B_i is not a weak guarantor of A_j


<table>
<thead>
<tr>
<th>(C_1)</th>
<th>(\ldots)</th>
<th>(C_2)</th>
<th>(C_3)</th>
<th>(C_4)</th>
<th>(C_5)</th>
<th>(C_6)</th>
<th>(C_7)</th>
<th>(C_8)</th>
<th>(C_9)</th>
<th>(\ldots)</th>
<th>(C_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_1)</td>
<td>(A_2)</td>
<td>(A_3)</td>
<td>(A_4)</td>
<td>(A_5)</td>
<td>(A_6)</td>
<td>(A_7)</td>
<td>(A_8)</td>
<td>(A_9)</td>
<td>(\ldots)</td>
<td>(A_k)</td>
<td></td>
</tr>
</tbody>
</table>

**Relationship: architecture to requirements**

- **Red**: \(C_i\) is strong guarantor of \(A_j\)
- **Yellow**: \(C_i\) is weak guarantor of \(A_j\)
- **Green**: \(C_i\) is not a weak guarantor of \(A_j\)
Concluding remarks

• Understand S&SE first then work on PLE
  ◊ RE
    • structuring functionality
  ◊ architecture
  ◊ tracing

• Dominant role of RE also for PLE
  ◊ RE-driven PLE

• On-going work
  ◊ a more structured view on RE
  ◊ generalize results in S&SE to PLE
  ◊ an artifact model for RE including variability
  ◊ relating function hierarchies to FODA trees
  ◊ exploit the logical views for PLE