ICSEA 2012, November 21.

The Seventh International Conference on Software Engineering Advances

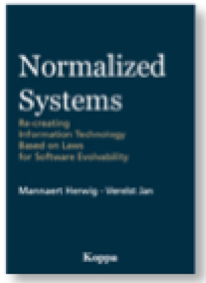
Normalized Systems: Towards Designing Evolvable Modular Structures

Prof. dr. Herwig Mannaert
Normalized Systems Institute
University of Antwerp

Universiteit Antwerpen

About Normalized Systems

- A theoretical framework to gain insight into the behaviour of *modular structures under change*, and aiming at the design of *evolvable modular structures*
  - Initial scope: *Modular Structures in Software Architectures*
  - Based on modularity instead of software techmologies
    - ➔ Completely independent of any framework, programming language,
    - Has shown to be able to deal with the challenge of increasing complexity
      - E.g. hardware, Internet, space industry…
  - Grounded in systems theoretic concepts
  - Publications: book, >40 (journals + conference proceedings), (invited) lectures at different universities …
  - Education: undergraduate, postgraduate…



AN INCONVENIENT TRUTH

The Dream: Doug Mc Ilroy

"expect families of routines to be constructed on *rational principles* so that families fit together as **building blocks"**

uit: McIlroy, *Mass Produced Software Components*,
1968 NATO Conference on Software Engineering, Garmisch, Germany.

The Reality: Manny Lehman

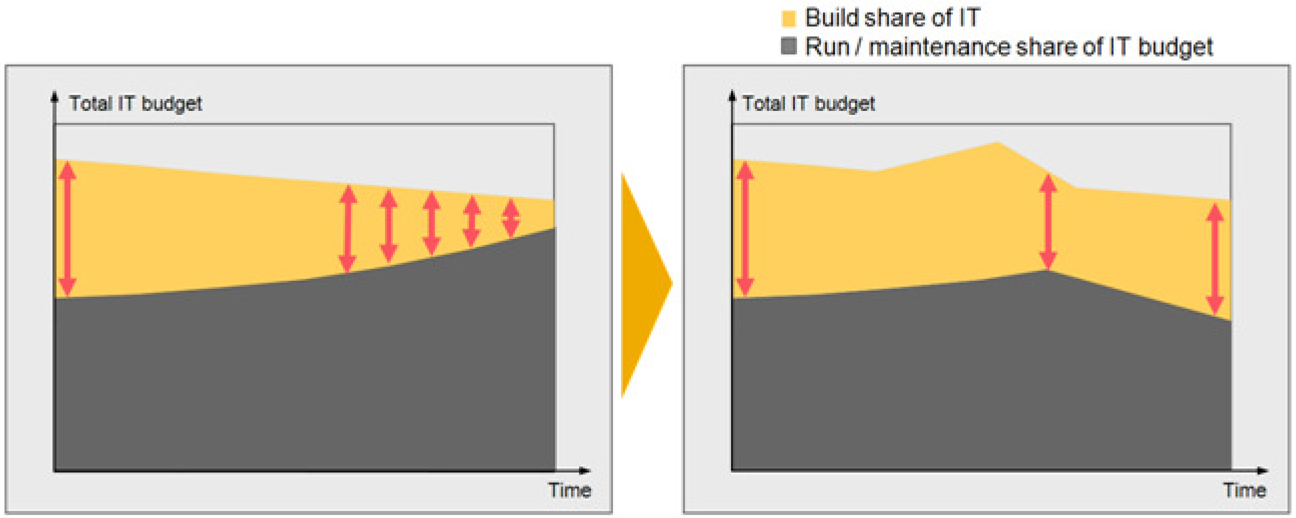
The Law of Increasing Complexity
Manny Lehman

"As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."

Proceedings of the IEEE, vol. 68, nr. 9, september 1980, pp. 1068.

An Indication: IT Maintenance

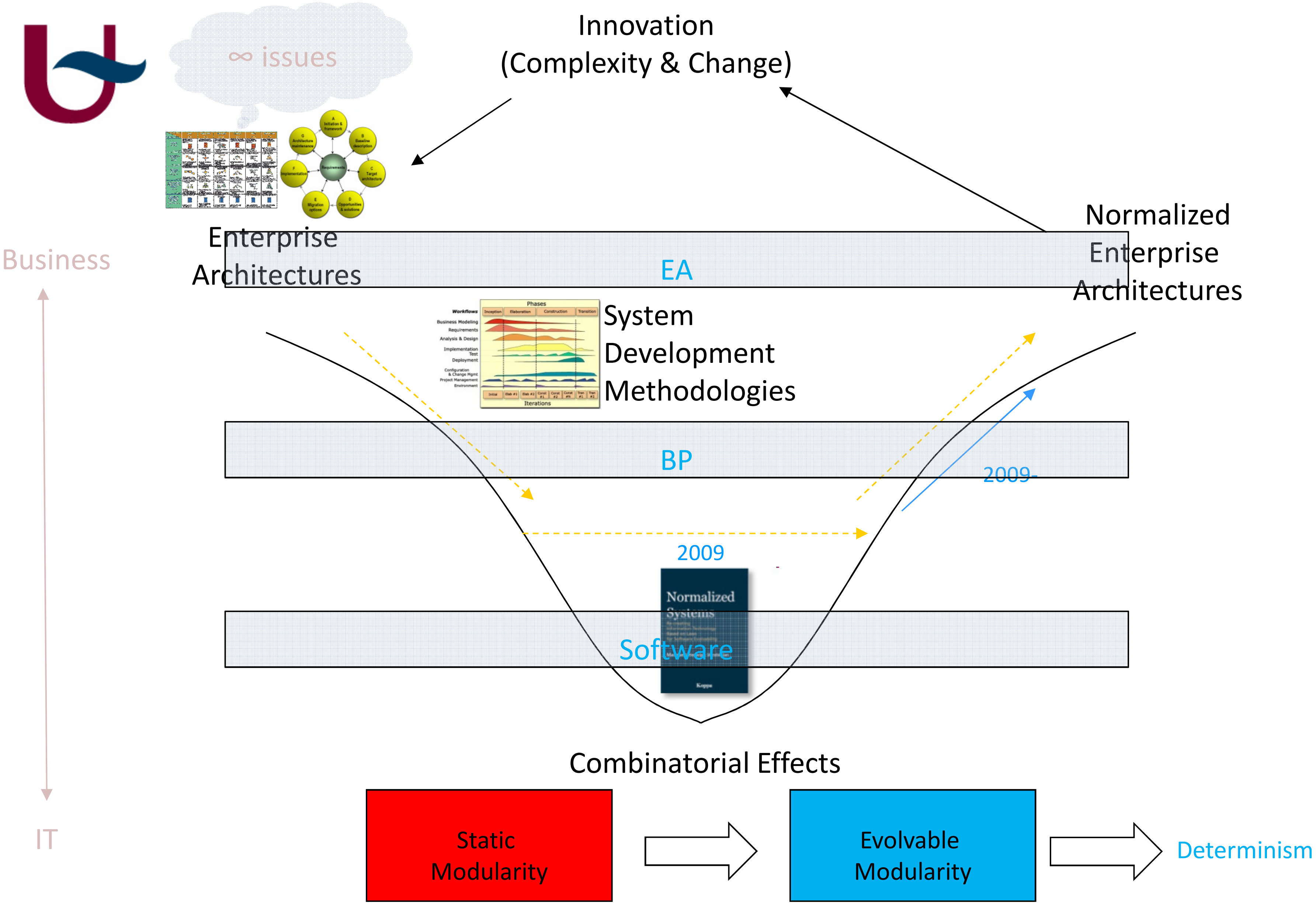
Continuing IT build as before will yield more and more complexity and thus increase build and maintenance costs. In times of cost reduction and frozen IT budgets this will lead to a dead lock in innovation and functionality enhancement.



Do-Nothing-Alternative

Restructuring supported by Enterprise Architecture

To achieve an improved and improving build / run cost ratio an initial invest is inevitable, because the restructuring of the IT landscape needs concepts and projects for complexity reduction and the setup of new, cost efficient and flexible solutions.

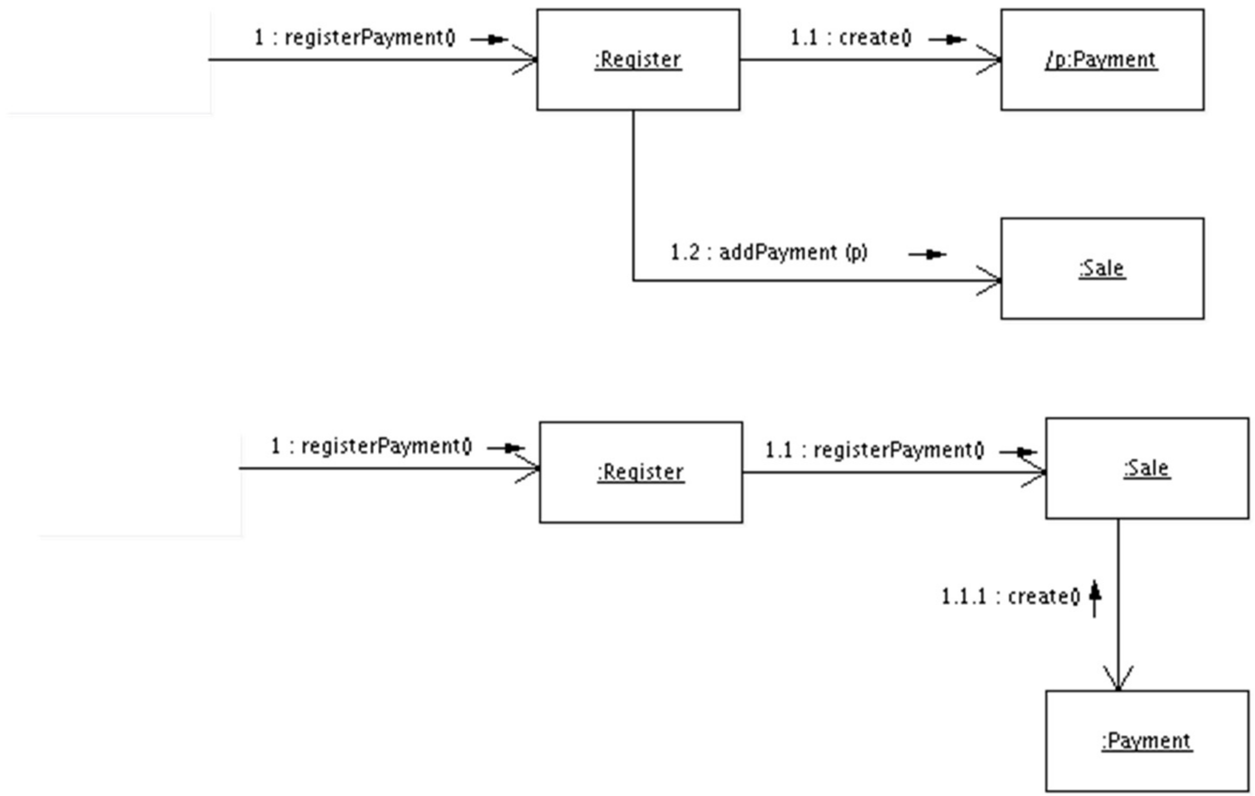

∞ issues
Innovation
(Complexity & Change)
Enterprise
Architectures
Normalized
Enterprise
Architectures
Business
EA
System
Development
Methodologies
BP
2009-
2009
Normalized
Systems
Software
IT
Combinatorial Effects
Static
Modularity
Evolvable
Modularity
Determinism

An Indication: IT Vagueness

- Different opinions about ‘good’ design
  - “Low coupling” is too vague !
  - “Information hiding” was formulated by Parnas in 1972, but still needs to be refined
  - Philippe Kruchten (2005): “We haven’t found the fundamental laws in software like in other engineering disciplines”

- *Low coupling and high cohesion. Everybody knows this. The question is how to do this.*

An Indication: IT Vagueness



An Indication: Some Thoughts

- Design, the mapping from functional requirements to constructive primitives, is a complex activity, e.g. designing a car based on use cases.

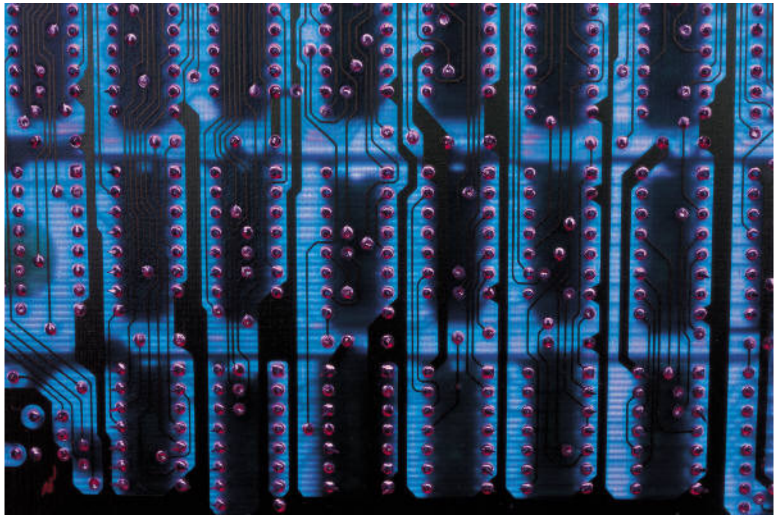
  It cannot be done on a 1-1 basis.

- Modularity in other disciplines, like hardware and aerospace, is static modularity. It does not accomodate continuous changes.

  We require evolvable modularity.

BROADENING THE SCOPE

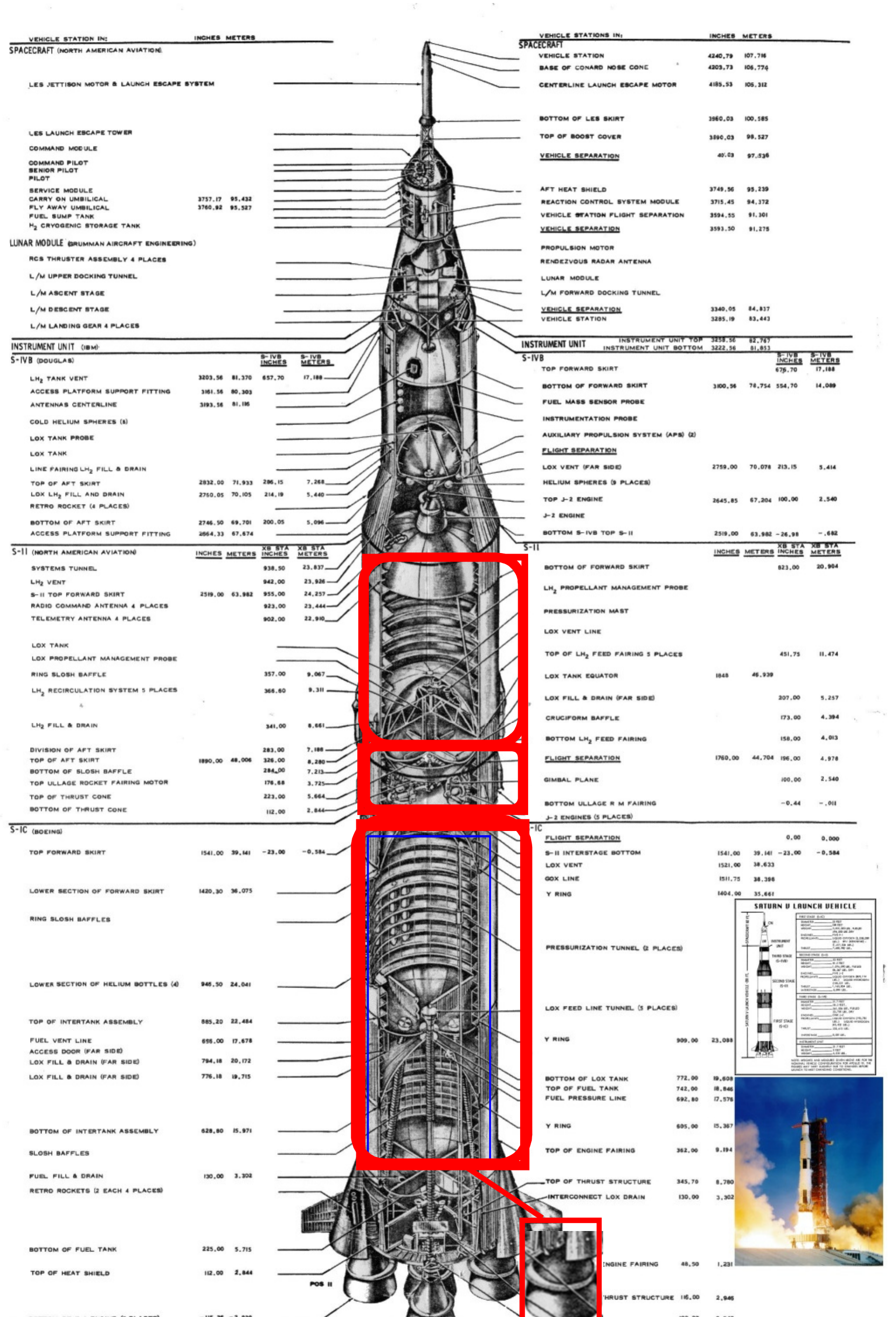
Controlling Complexity by modularity can be done …



Other disciplines have mastered the **structured assembly** of **large numbers** of **fine-grained static** modules… e.g. hardware !

However:

Modularity is static

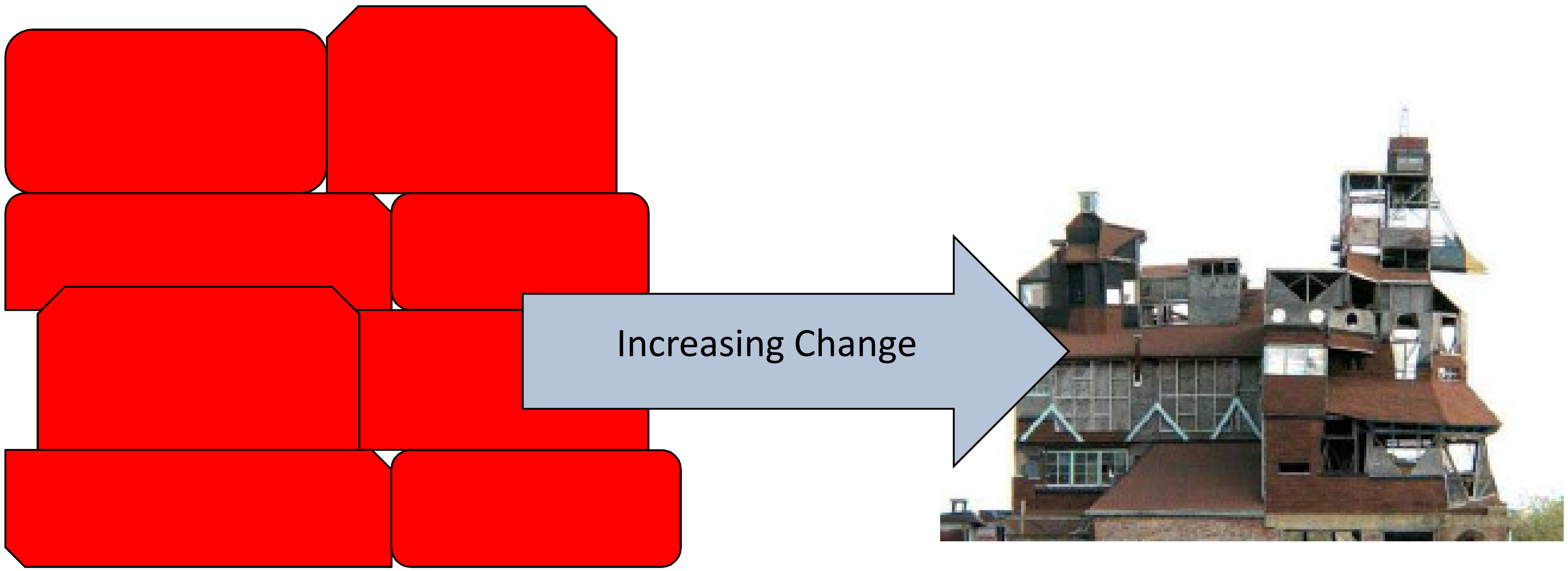


Some more examples

- Airbus 380 could not be designed by taking x2 for every measure of the Airbus 340 plan
- Instruction set of a microprocessor cannot be extended by adding another module
- Construction buildings cannot grow over time by simply adding additional units
- Car performance cannot be upgraded by adding additional parts to the engine
- …

Evolvability: The Main Issue

Static Modularity | Complexity



Lehman, No McIlroy

Systems Theory → Evolvability

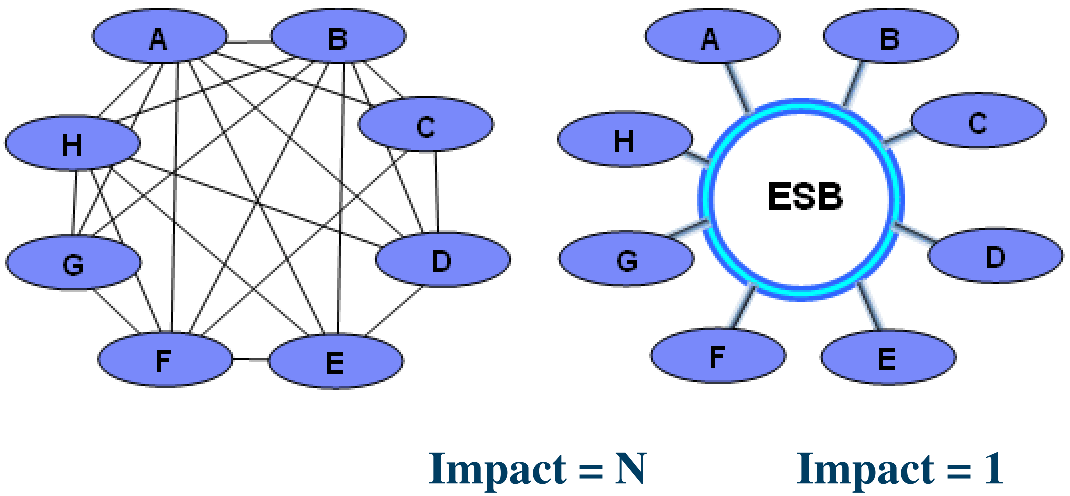
- Stability in System Dynamics:
  - In systems theory, the dynamic evolution of a system is studied based on a differential or difference equation
  - A system is stable if and only if:
    - a bounded input results in a bounded output
    - it has poles in the left plane or inside the unit circle:
  - For a first order model, **stability ←→ a<0**:
    - dy(t)/dt = x(t) + ay(t) ←→ Y(s)/X(s) = 1/(s-a)
    - y[k+1]-y[k] = x[k] + ay[k] ←→ Y(z)/X(z) = 1/(z-(1+a))
  - This means that the increase cannot have a positive contribution from the size of the system

Systems Theory → Evolvability

- Stability in system dynamics:
  - Is used to study dynamics of system operations
    - Mechanical, e.g. constructions, vehicles, …
    - Electrical, e.g. amplifiers, generators, …
    - Hydraulical, e.g. pumps, engines, …
    - …
  - Is not used to study dynamics of system artefacts
    - Rockets and airplanes
    - Software and information systems
    - Organizations and human enterprises
    - …

IT: Enterprise Service Bus

- The effort to include an additional component may or may not vary with the system size
  or: airline spoke and hub



Source: http://nl.wikipedia.org/wiki/Enterprise_Service_Bus

EVOLVABILITY PRINCIPLES

The Transformation Model

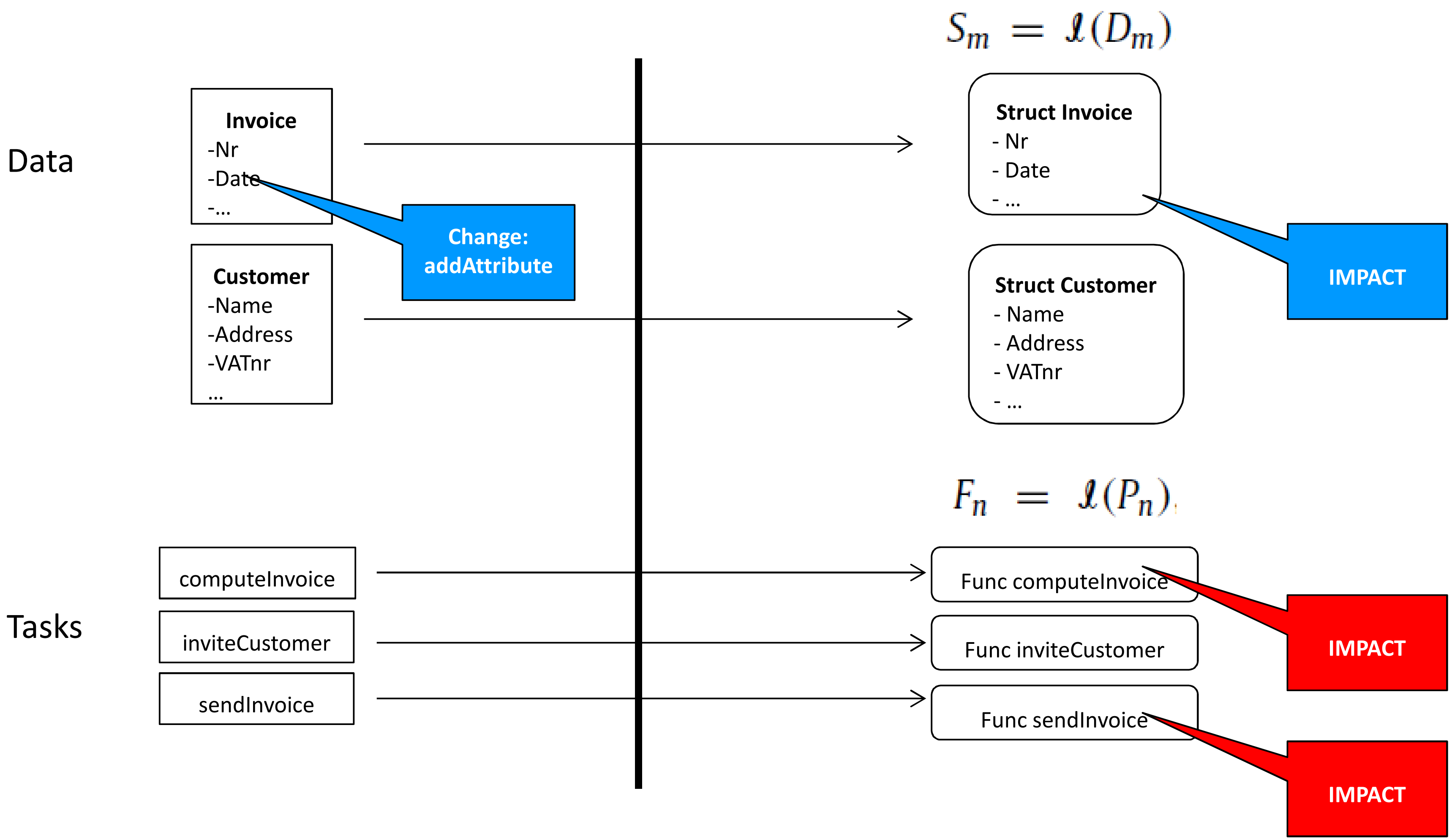
- Study the transformation of functional requirements into software primitives as a transformation:

$$\mathcal{S} = \mathcal{I}(\mathcal{R}).$$

- Consider the functional requirements at an extremely basic hierarchical level:
  - Data structures and processing tasks
  - → Software coding in its elementary form
  - → Implicit in every realistic software system

- Study the transformation of changes

A Simple Transformation



A Simple Transformation

- Demanding systems theoretic stability for this transformation, leads to the derivation of *principles* in line with existing heuristics:

$$\mathcal{S}_m = \mathcal{S}_t \setminus \mathcal{S} = \mathcal{I}(\mathcal{R}_m) \cup \boxed{\delta\mathcal{S}.}$$

To obtain a scalar equation, we use cardinalities of sets and a coefficient a:

$$|\mathcal{S}_m| = |\mathcal{S}_t| - |\mathcal{S}| = |\mathcal{I}(\mathcal{R}_m)| + \boxed{a|\mathcal{S}|}$$

or using the discrete variable k to represent ongoing development iterations:

$$|\Delta\mathcal{S}| = |\mathcal{S}[k+1]| - |\mathcal{S}[k]| = |\mathcal{I}(\mathcal{R}_m[k])| + \boxed{a[k]|\mathcal{S}[k]|.}$$

Action Version Transparency

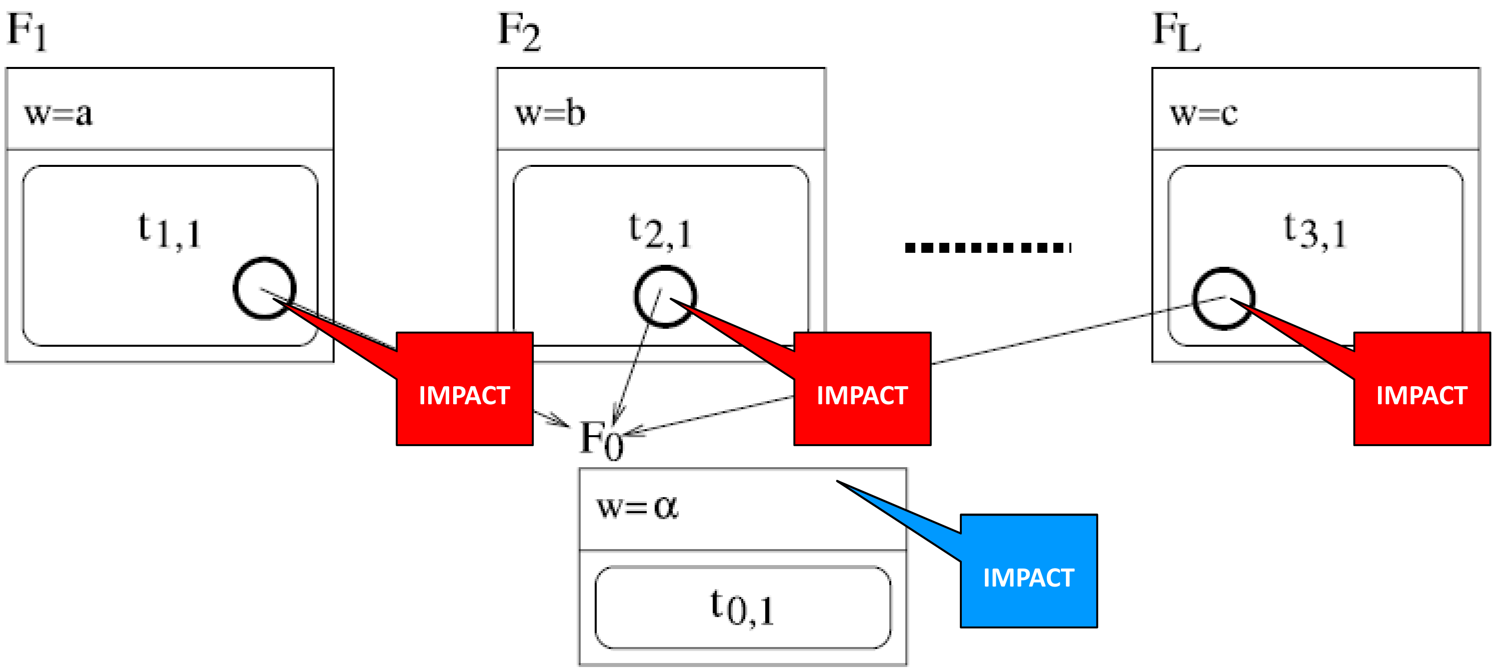


Fig. 4. Schematic representation of a function F_n with two tasks and multiple versions.

$$\mathcal{S}_m = \mathcal{I}(\mathcal{R}_m) \cup \left\{F_l(w, t_{l,2})\right\}_{l=1,\ldots,L}.$$

Separations of Concerns

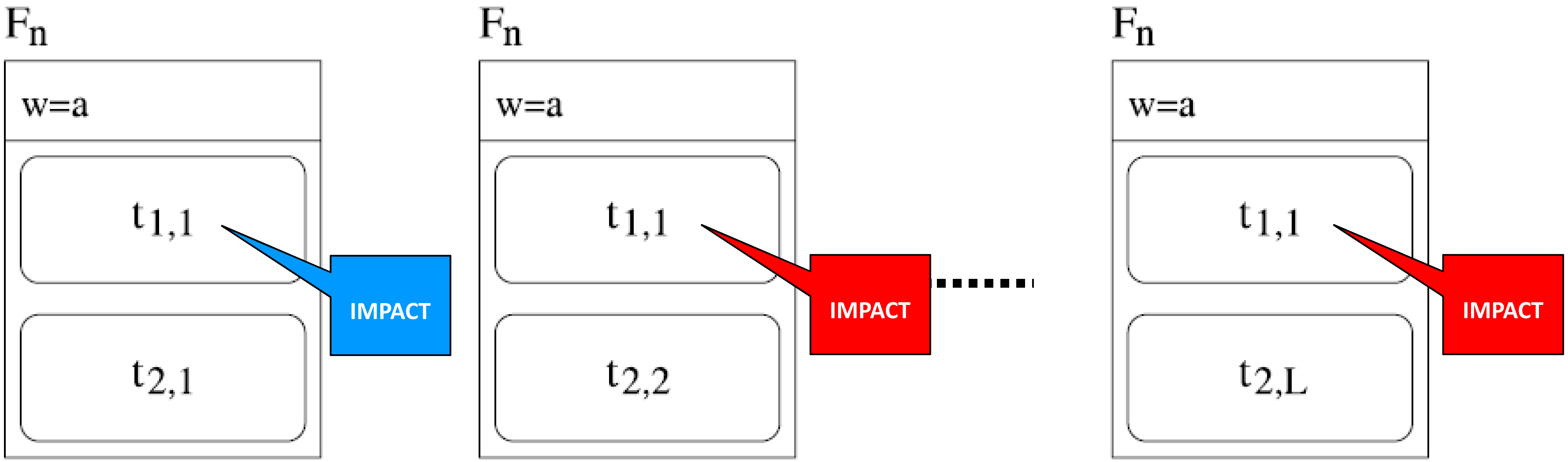


Fig. 1. Schematic representation of a function F_n with two tasks and multiple versions.

$$\mathcal{S}_m = \mathcal{I}(\mathcal{R}_m) \cup \left\{F_n(w = a, t_{1,2}, t_{2,l})\right\}_{l=1,\dots,L}$$

Separate an Unidentified Task

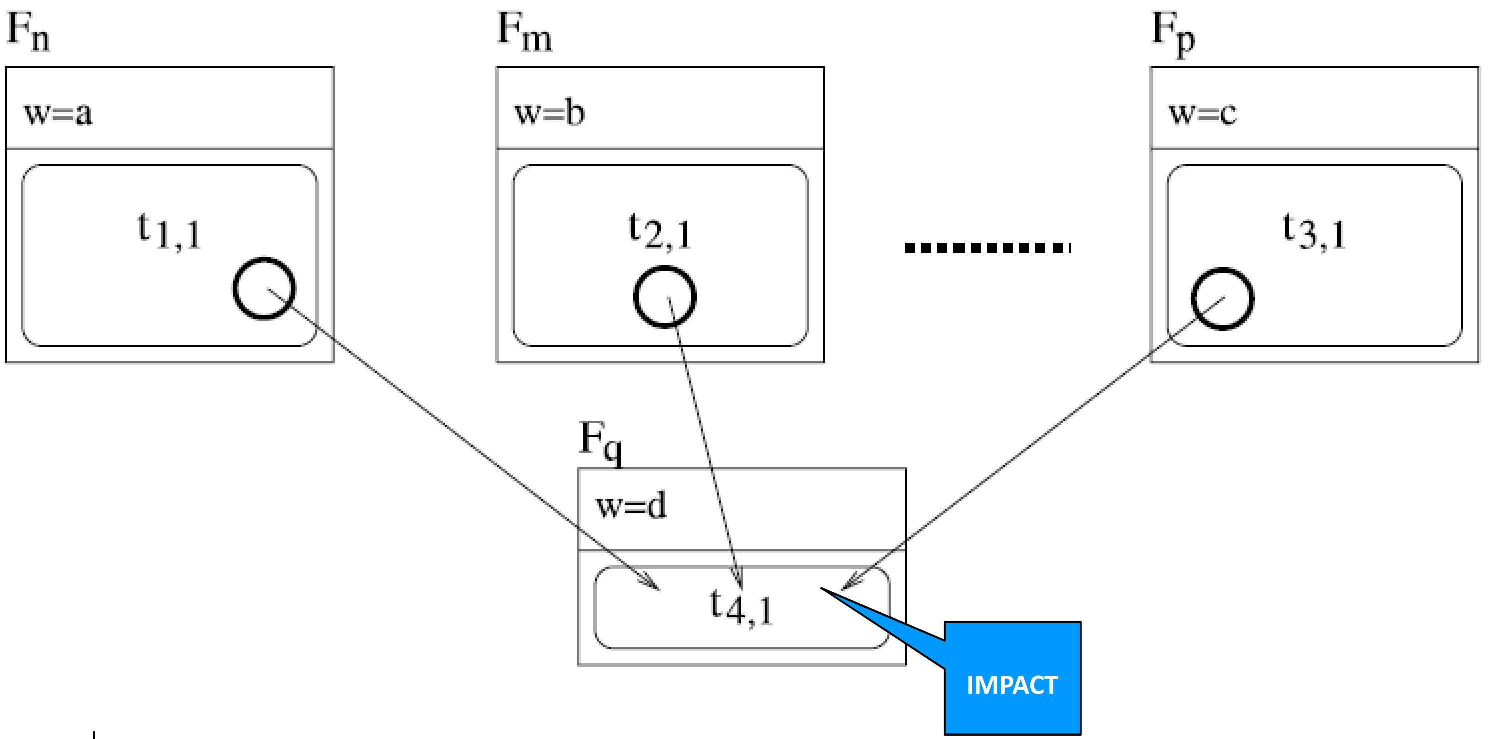


Fig. 2. Various functions *F* with a single task as an unidentified change driver.

Normalized Systems Principles

- Modularity x Change ➔ **Combinatorial Effects (CE)** !
  - CE = (hidden) **coupling** or dependencies, **increasing with size of the system** !
  - **NS Principles** identify CE at seemingly orthogonal levels
    - SoC: Which tasks do you **combine** in a single module ?
      - "An action entity can only contain a single task."
    - DVT: How do you **combine** a data and action module ?
      - "Data entities that are received as input or produced as output by action entities, need to exhibit version transparency."
    - AVT: How do you **combine** 2 modules ?
      - "Action entities that are called by other action entities, need to exhibit version transparency."
    - SoS: How do you **combine** modules in a workflow ?
      - "The calling of an action entity by another action entity needs to exhibit state keeping."
  - ➔ CE are due to the way tasks, action entities and data entities are **combined or integrated** !

Combinatorial Effects

Current constructs allow CE

⇓

"Any developer that violates any principle at any time during development or maintenance"

⇓

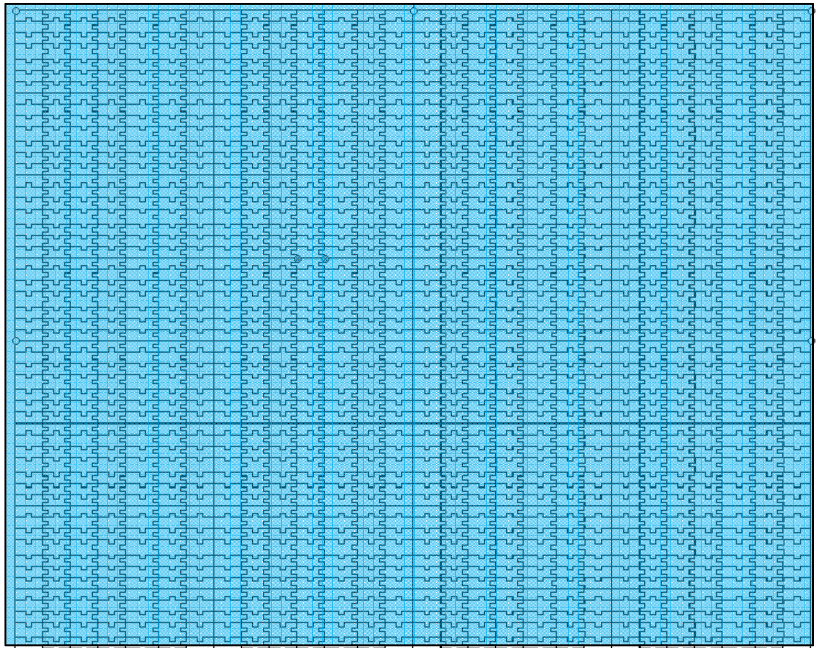
CE omnipresent,
during development and ever increasing during maintenance !

Normalized Systems Principles

- Are not new:
  - They are consistent with heuristic design knowledge
  - However, the way in which they are derived from a single postulate is new
- Presented principles solve the vagueness in identifying combinatorial effects:
  - Until now, no clear principles
    - ➔ subjectivity, ad hoc
  - McIlroy: “to be constructed on *rational principles”*
- Conclusion
  - Omnipresent CE ➔ No *evolvable* modularity !

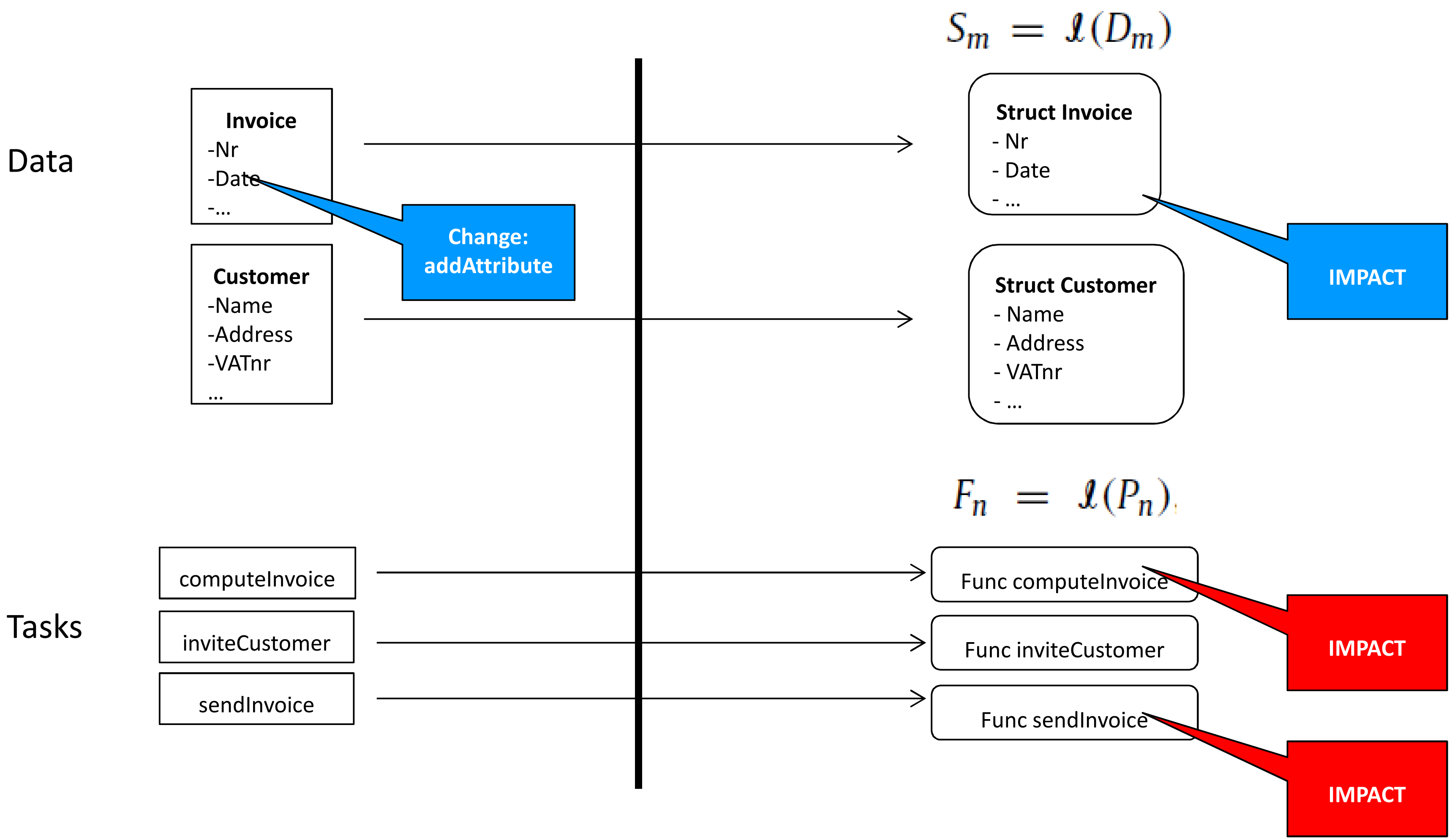
TOWARDS EVOLVABLE ELEMENTS

A necessary condition: Fine-grained Modular Structure

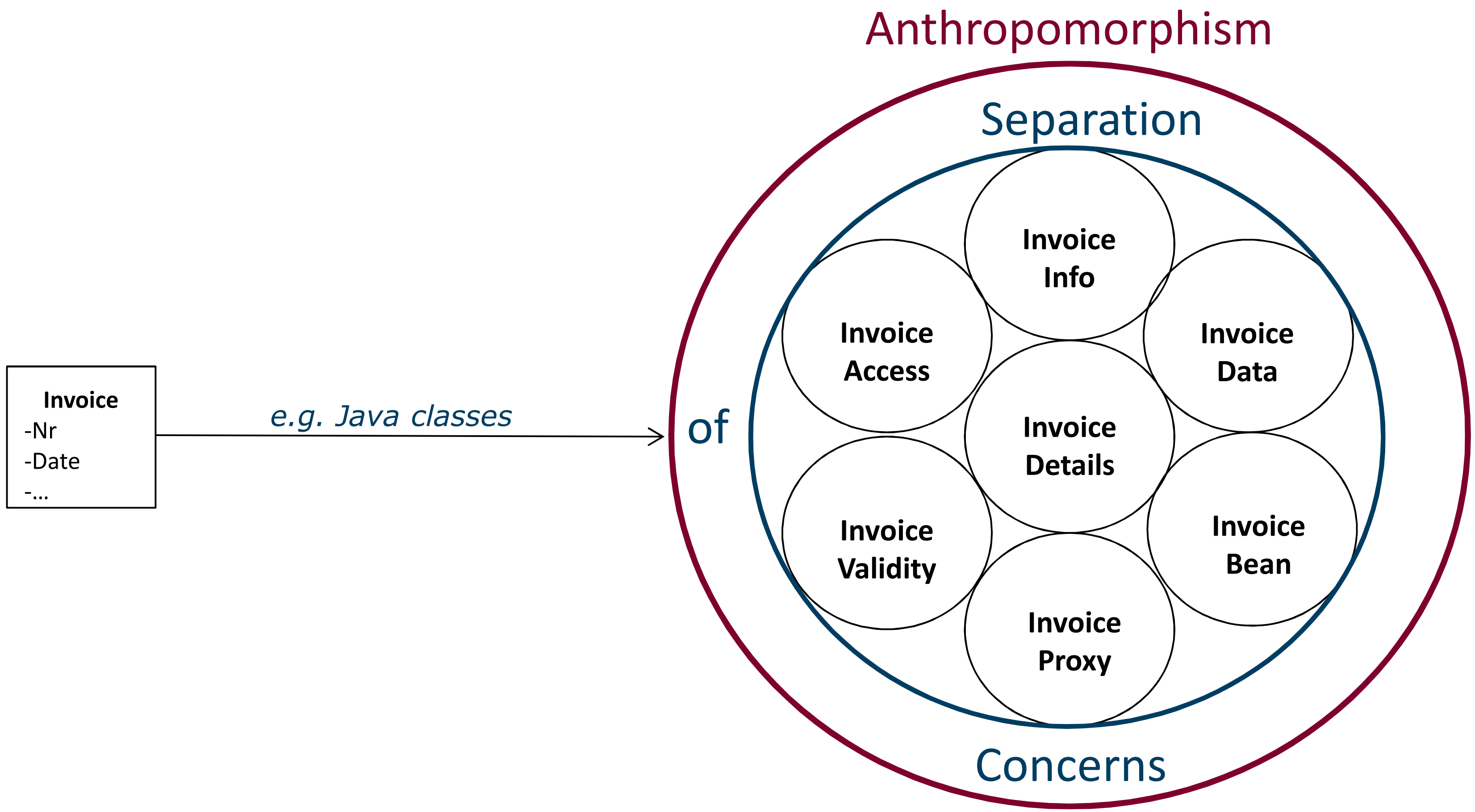


E.g. SoC: a module can know only 1 technology
➔ for every technology, a different module is required !

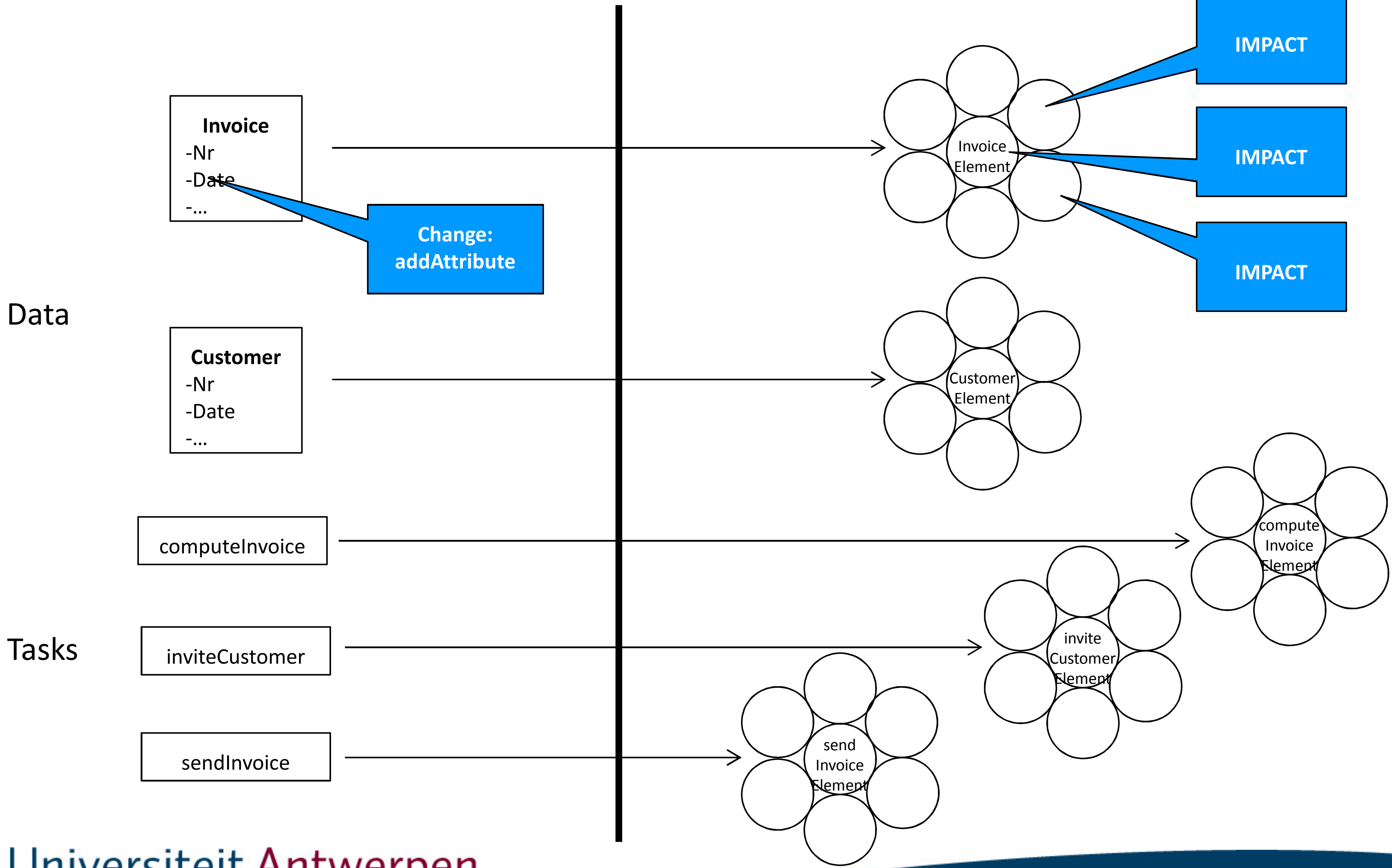
A Simple Transformation



A More Complex Transformation



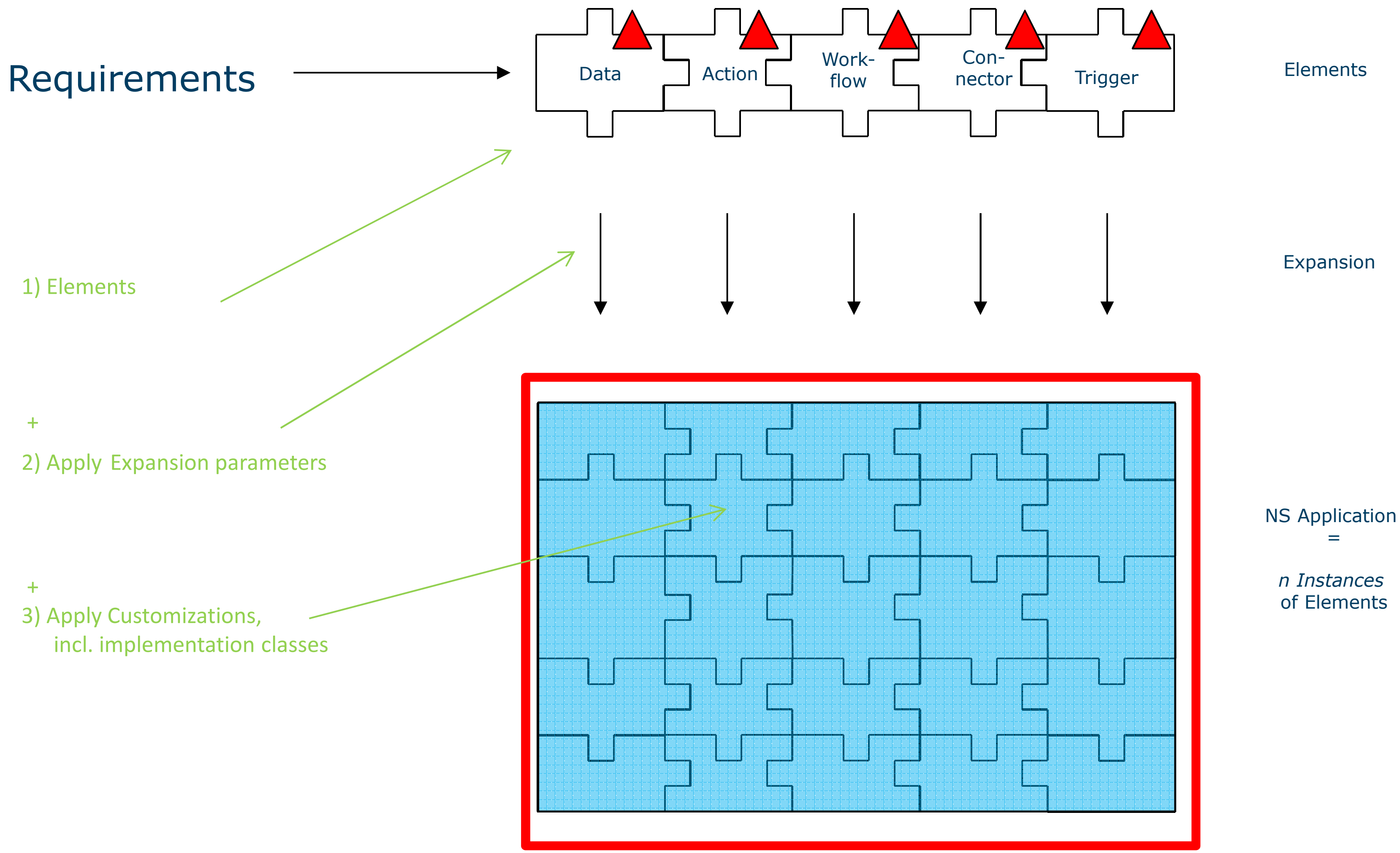
A More Complex Transformation



Normalized Systems Elements

- The proposed solution =
  - Structure through Encapsulations, called Elements
    - A Java class is encapsulated in 8-10 other classes, dealing with cross-cutting concerns, in order to deal with the anticipated changes *without CE*, and fully separating the element from all other elements.
    - Every element is described by a "detailed design pattern". Every element builds on other elements.
    - Every design pattern is executable, and can be expanded automatically.
  - Realizing the core functionality of Information Systems
- Application = *n* instantiations of Elements

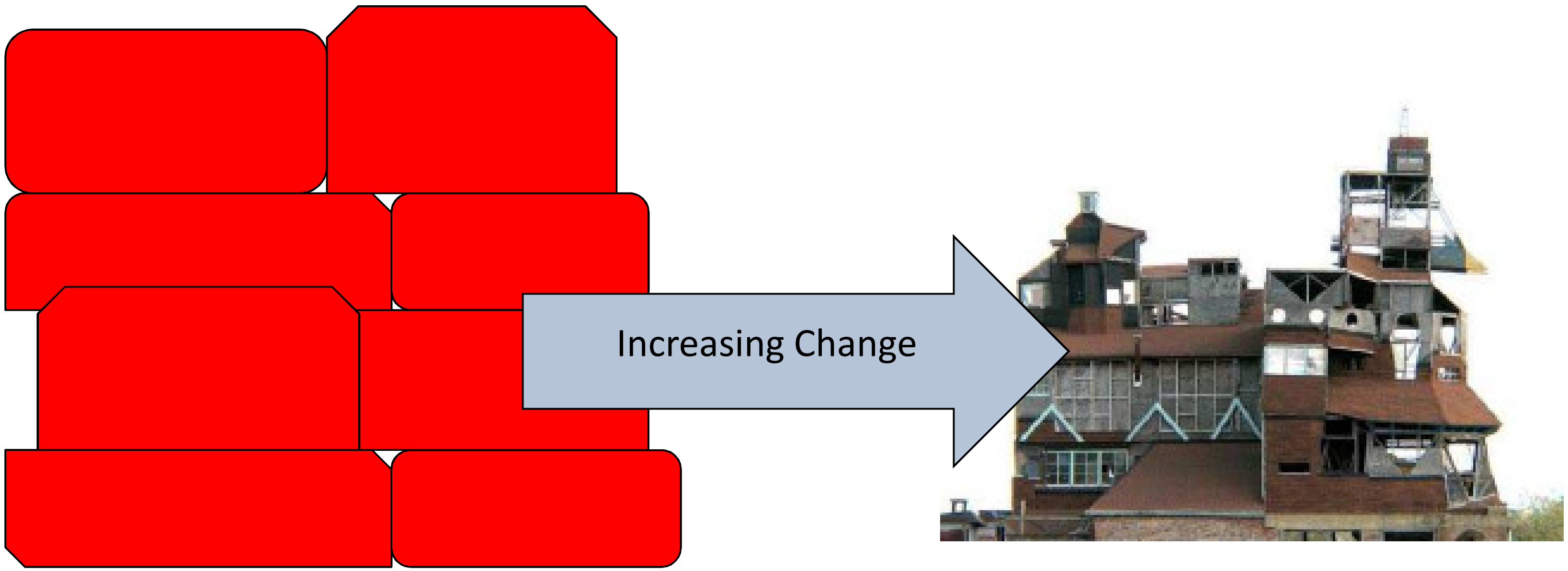
Instantiating Elements



Evolvability: The Main Issue

Static Modularity

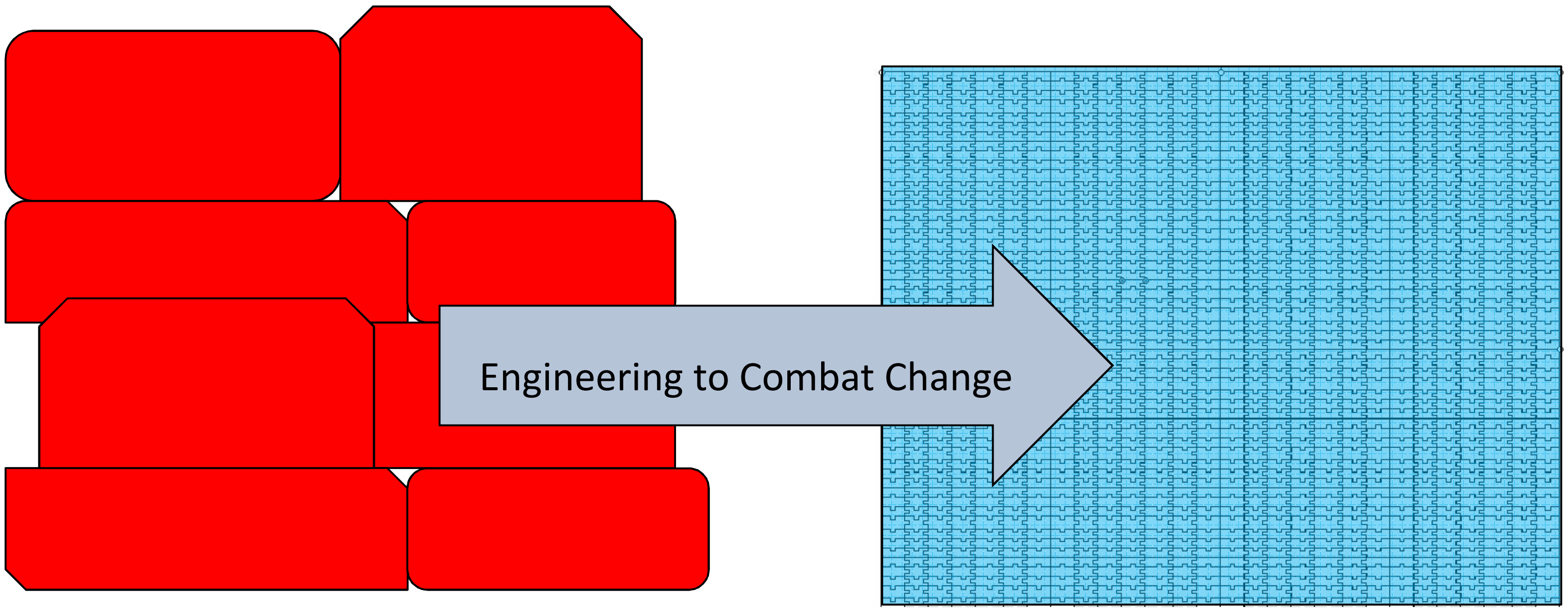
Complexity



Lehman, No McIlroy

Evolvability: The Main Issue

Static Modularity   Evolvable Modularity



Ever increasing complexity !

Not straightforward, but true Engineering, and Determinism !

The Final Goal: Determinism

- **Systematic elimination** of CE, using **fine-grained modular structures such as Elements**, while controlling their inherent complexity, leads to **determinism:**
  - All applications have similar fine-grained software architecture
    ➔ product line or product factory
  - Impact analysis
  - Correctness
  - Reliability and Performance
  - Traceable execution
  - …

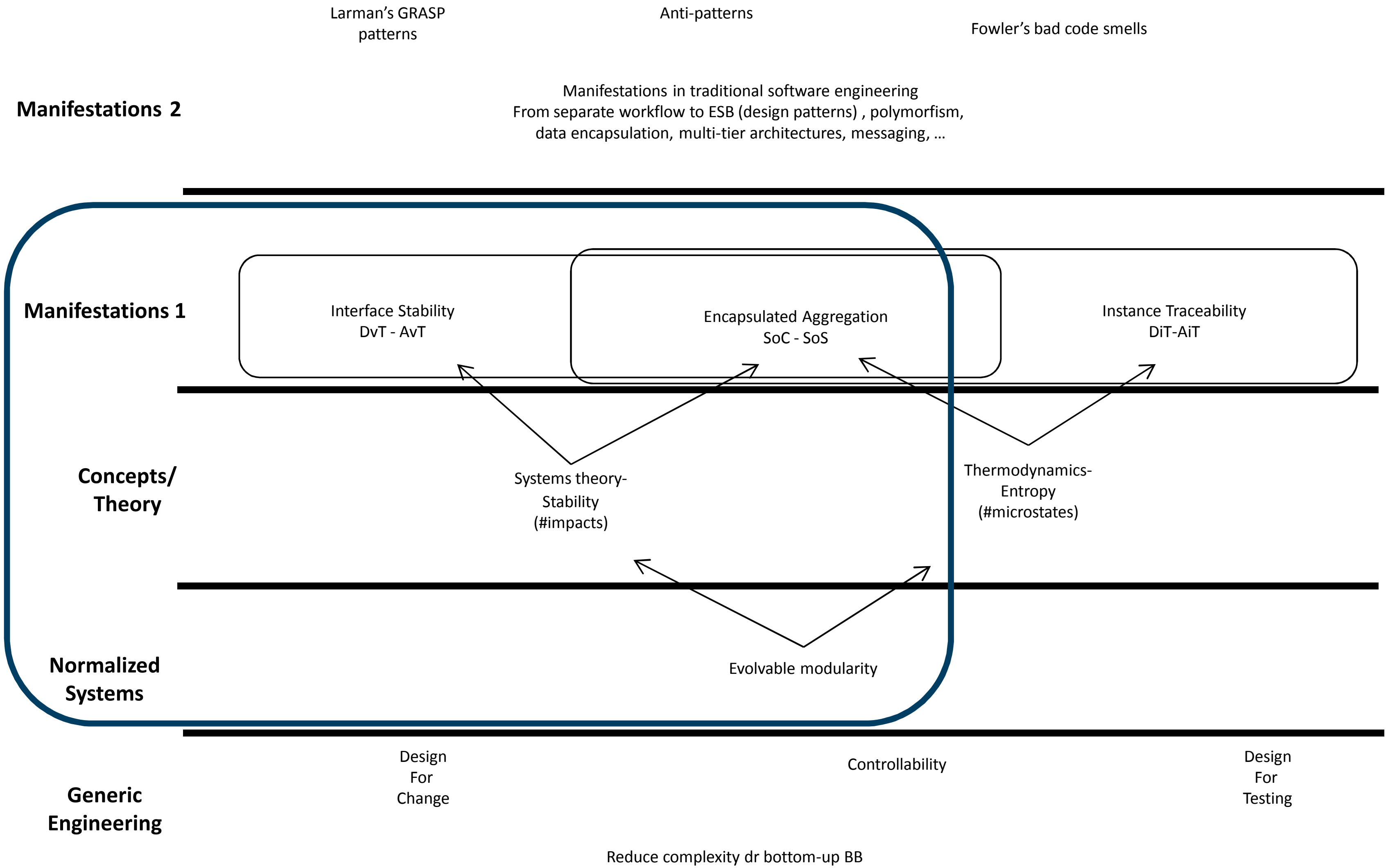
FACTS, THOUGHTS AND DREAMS

Knowledge: Contributions

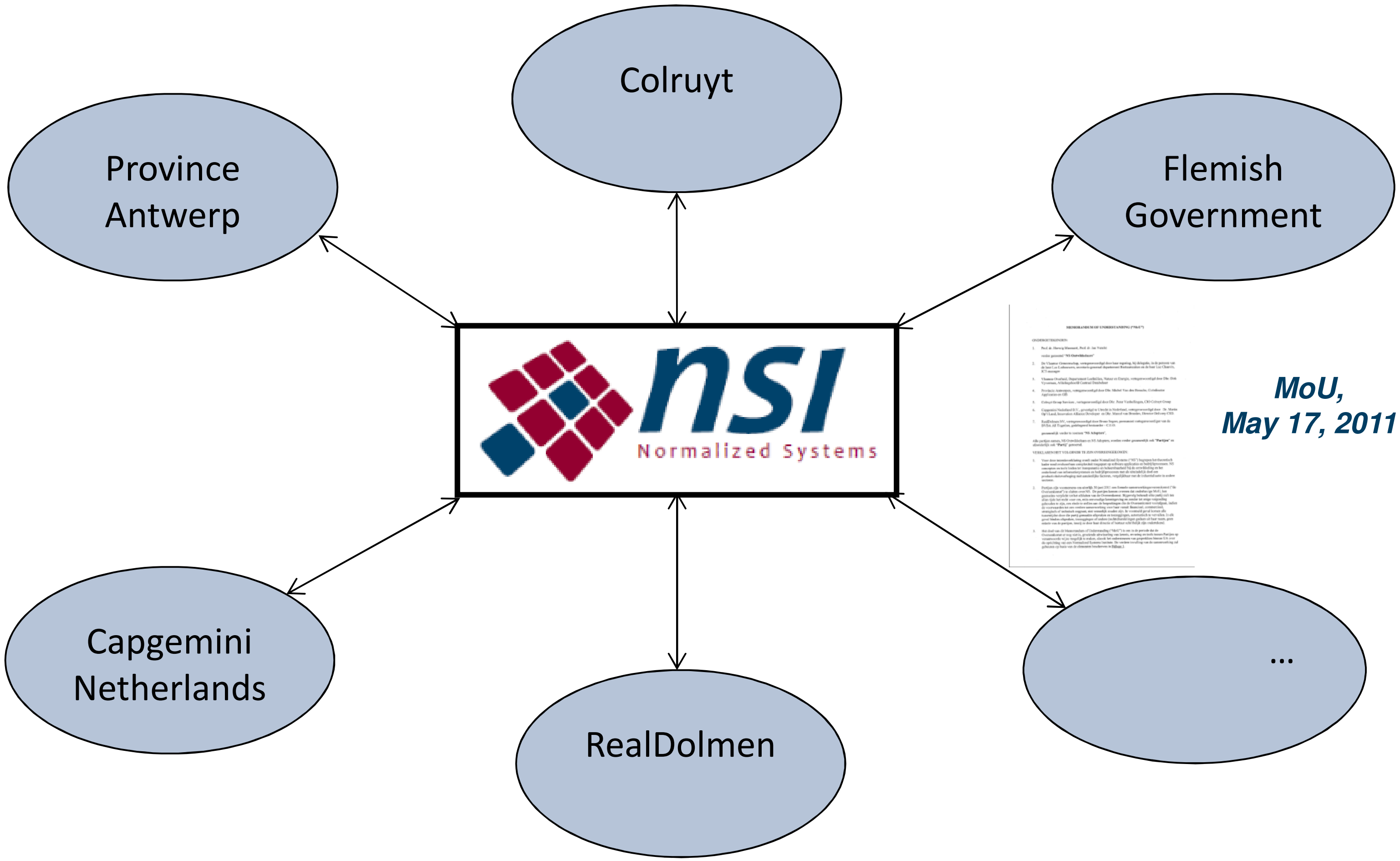
- Contributions to ***insight*** into current problems
  - Proposing a mechanism of Lehman's Law
  - Explaining why software reuse is so difficult
  - Linking evolvability issues to non-software

- Proposing the structure of a possible ***solution***
  - Software elements to guarantee evolvability
  - Applications as instantiations of elements

Knowledge: The Vision



Valorization: The Model



Valorization: Achievements

- Community codebase of element expanders:
  - Application stack in EJB2 and EJB3
  - Presentation stack in Cocoon and Struts2
  - Client interactivity stack in Knockout/Bootstrap
- Applications made by the partners:
  - >20 currently in production
  - >10 in acceptance testing
  - Specified in detail by elements and extensions

The Future: Dreams

- Pursue existing software efforts:
  - More partners and applications
  - Rejuvenation application portfolio
- New areas now being initiated:
  - Business processes
  - Industrial controllers
  - Smart energy grids
- Maybe one day followers:
  - Rockets and airplanes
  - Buildings, cars, …

Some References

- De Bruyn Peter, van Nuffel Dieter, Verelst Jan, Mannaert Herwig.- Towards applying normalized systems theory implications to enterprise process reference models. Lecture notes in business information processing - ISSN 1865-1348 - 110(2012), p. 31-45
  http://dx.doi.org/10.1007/978-3-642-29903-2_3
  [c:irua:98376]
- Mannaert Herwig, Verelst Jan, Ven Kris.- Towards evolvable software architectures based on systems theoretic stability. Software practice and experience - ISSN 0038-0644 - 42:1(2012), p. 89-116
  http://dx.doi.org/doi:10.1002/spe.1051
- Mannaert Herwig, Verelst Jan, Ven Kris.- The transformation of requirements into software primitives : studying evolvability based on systems theoretic stability. Science of computer programming - ISSN 0167-6423 - 76:12(2011), p. 1210-1222
  http://dx.doi.org/doi:10.1016/j.scico.2010.11.009
  [c:irua:91112]
- van Nuffel Dieter, Mannaert Herwig, de Backer Carlos, Verelst Jan.- Towards a deterministic business process modelling method based on normalized systems theory. International journal on advances in software - ISSN 1942-2628 - 3:1/2(2010), p. 54-69
  [c:irua:83738]

Some Questions

- herwig.mannaert@ua.ac.be