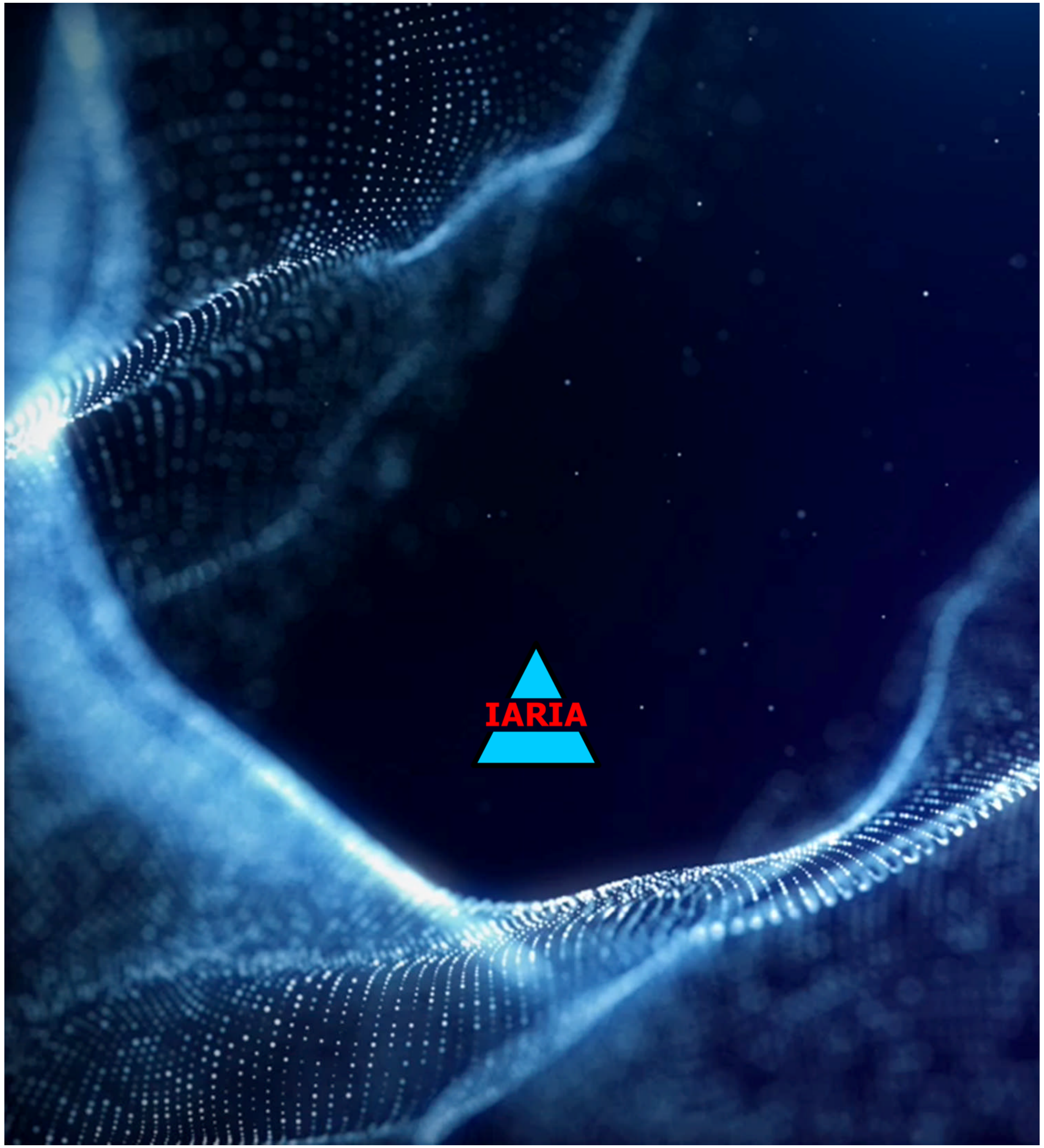


IARIA

COMPUTATIONWORLD 2024

Toward a Rejuvenation Factory For Software Landscapes

HERWIG MANNAERT,
TIM VAN WAES, FREDERIC HANNES

APRIL 15, 2024

Universiteit Antwerpen

Intro on myself & my work

- Electronics engineer, PhD in computer vision
- Co-created *Normalized Systems Theory* on engineering and architecture of evolvable software systems, i.e., enabling systems to cope with change
  - Books and papers (140 publications), and YouTube channel
  - Human adoption
    - Spin off company with 55 software engineers
    - > 65 software engineers at customers / partners
  - Software production
    - Suite of code generators and tools
    - Many software projects *AND* products, e.g.,
      - Energy monitoring and management suite
      - Command & Control Centre for medical drone transport
- Full professor at University of Antwerp, not an esteemed researcher

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

Introduction

- Agile paradigm has become default methodology in software
- There is a widespread belief in various benefits
  - E.g. timely delivery
- However, some disadvantages could be argued
  - E.g. increase of technical debt
- Normalized Systems Theory aims to improve evolvability through normative structure of software skeletons
- We investigate the balancing of evolvable architecture and agile design through the case study of an agile NST software factory

  → *DSR: Observational case study aiming to contribute to the rigor cycle*

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

Software Maintenance and Evolvability

- Software maintenance is intimately related to evolution as a large part is about non-corrective actions anf functional enhancements
- In depth studies of Manny Lehman lead to:
  - Insight that maintenance is evolutionary development
  - Formulation of *Lehman's Laws*, including ***Law of Increasing Complexity***
- Traditionally not much attention within IS community
- Recent more attention through the introduction of
  - *Technical debt*
  - *Maintenance debt*

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

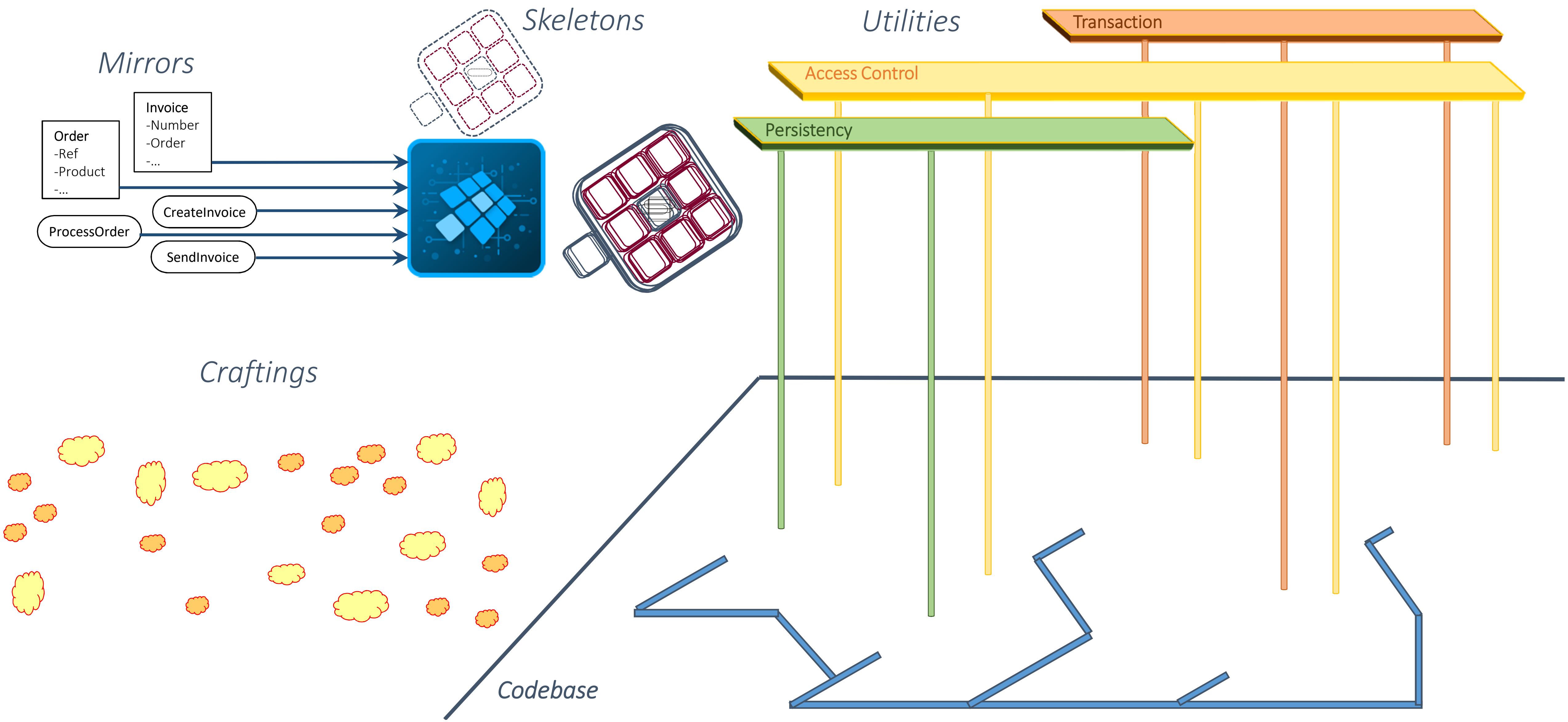
Design Theorems for Stable Software

- In order to avoid dynamic instabilities in the software design cycle, the *rippling of changes needs to be depleted or damped: a = 0*
- As these ripples create *combinations of multiple changes* for every functional change, we call these instabilities ***combinatorial effects***
- Demanding systems theoretic stability for the software transformation, leads to the derivation of ***principles*** in line with existing heuristics
- Adhering to these principles avoids dynamic instabilities, meaning that these *principles are necessary, not sufficient for systems stability*

Software Elements for Stable Skeleton Structures

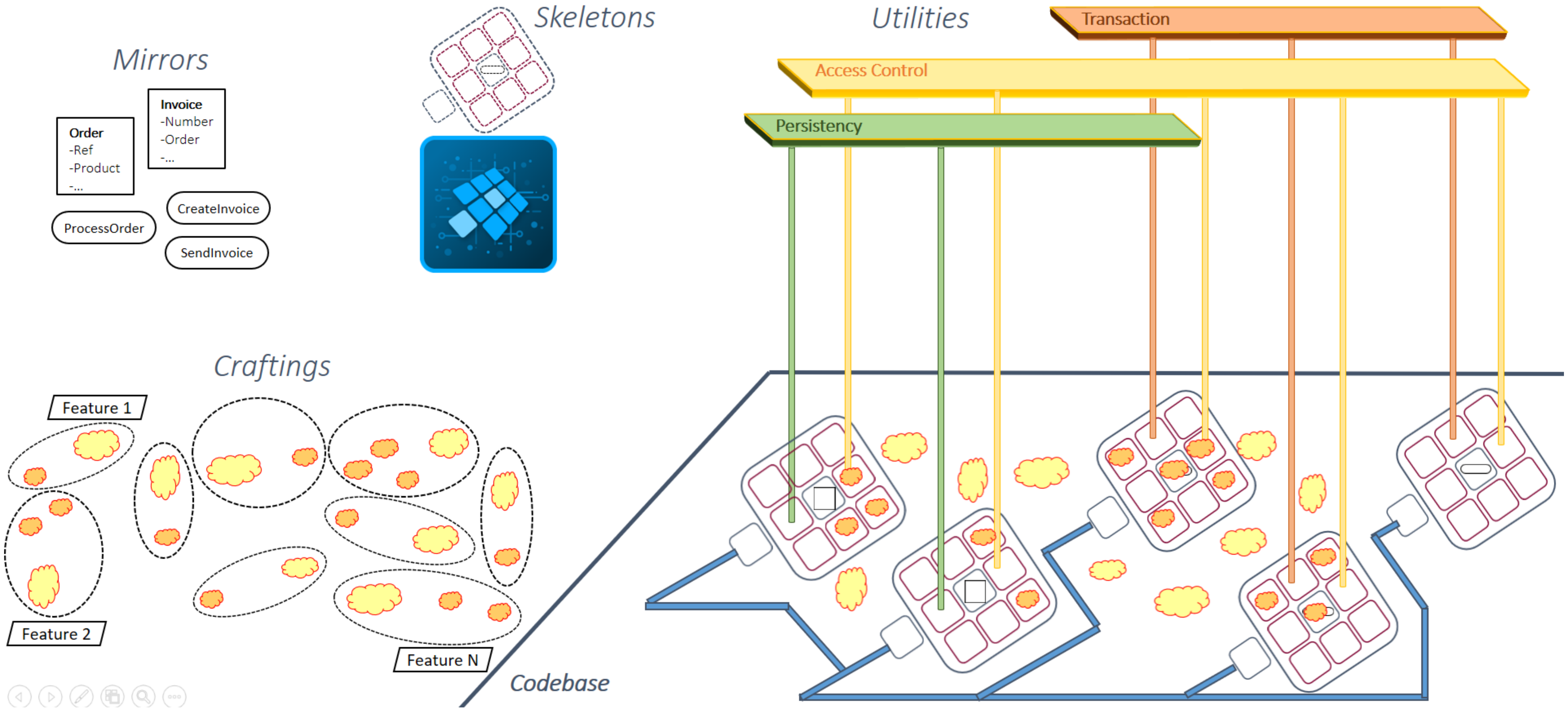
- Element structures are needed *to interconnect with CCC solutions*
- NS defines 5 types of elements, aligned with basic software concepts:
  - ***Data elements***, to represent data variables and structures
  - ***Task elements***, to represent instructions and/or functions
  - ***Flow elements***, to handle control flow and orchestrations
  - ***Connector elements***, to allow for input/output commands
  - ***Trigger elements***, to offer periodic clock-like control
- It seems obvious to use code generation techniques to create instances of these recurrent element structures
- Due to its simple and deterministic nature, we refer to this process as *expansion*, and to the generators as *expanders*

Separating the Dimensions of Variability



The Premise of Normalized Systems Theory

- Variability Dimensions



The Essence of Variability Dimensions

- We identify four dimensions of variability:
  - **M**odels or *mirrors*, new data attributes/relations, new elements
  - **E**xpanders or *skeletons*, new or improved implementations of concerns
  - **I**nfrastructure or *utilities*, new frameworks to implement various concerns
  - **C**ustom code or *craftings*, new or improved implementations of tasks, screens
- *If separated and well encapsulated*
  - Number of versions to maintain is *additive*: $\#V = \#M + \#E + \#I + \#C$
  - Number of versions available is *multiplicative*: $\#V = \#M \text{ x } \#E \text{ x } \#I \text{ x } \#C$
  - Where the same holds within any individual dimensions,
    e.g., infrastructure dimension: $\#I = \#G \text{ x } \#P \text{ x } \#B \text{ x } \#T$

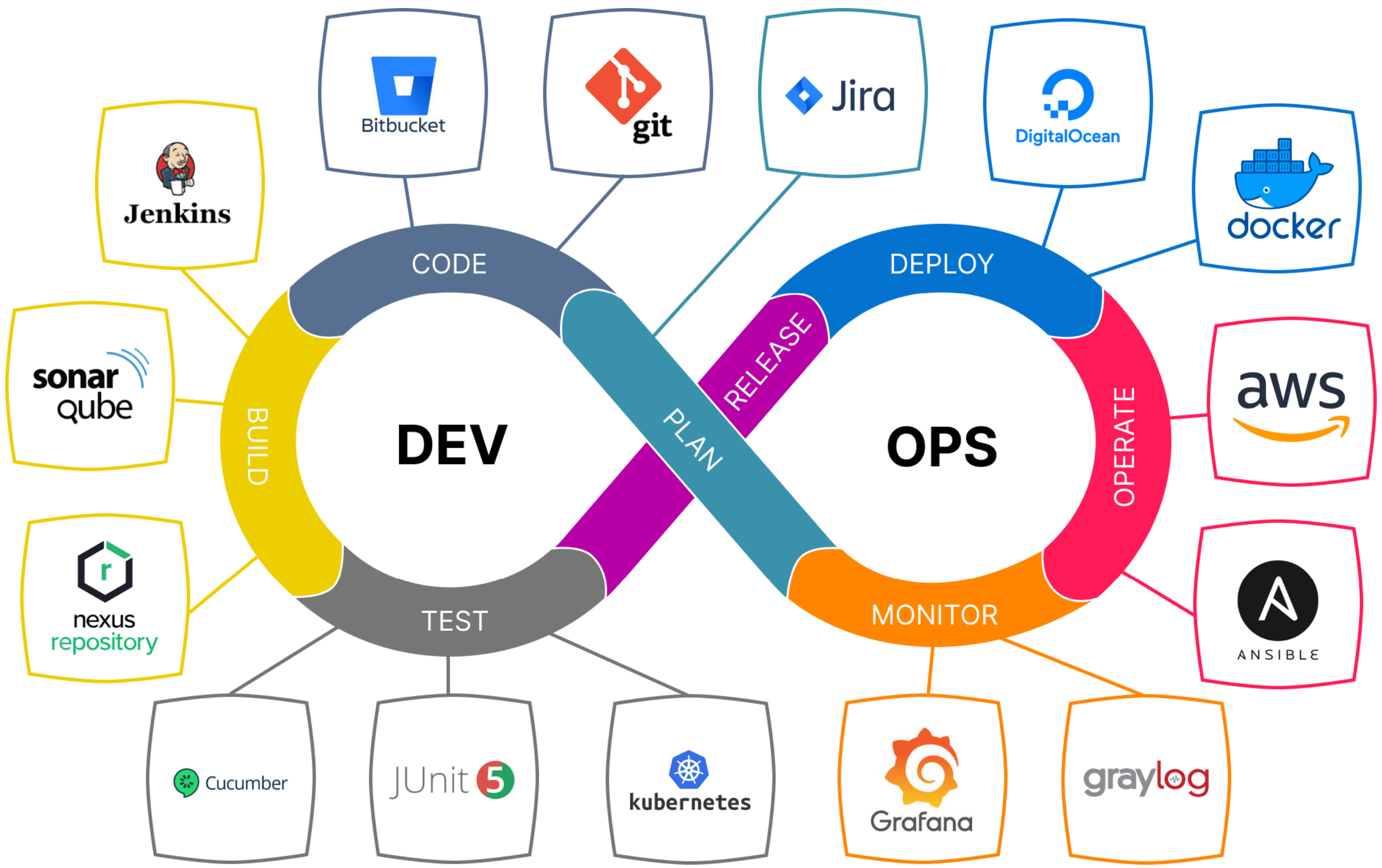
Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

Integrating Expansion in a Software Factory

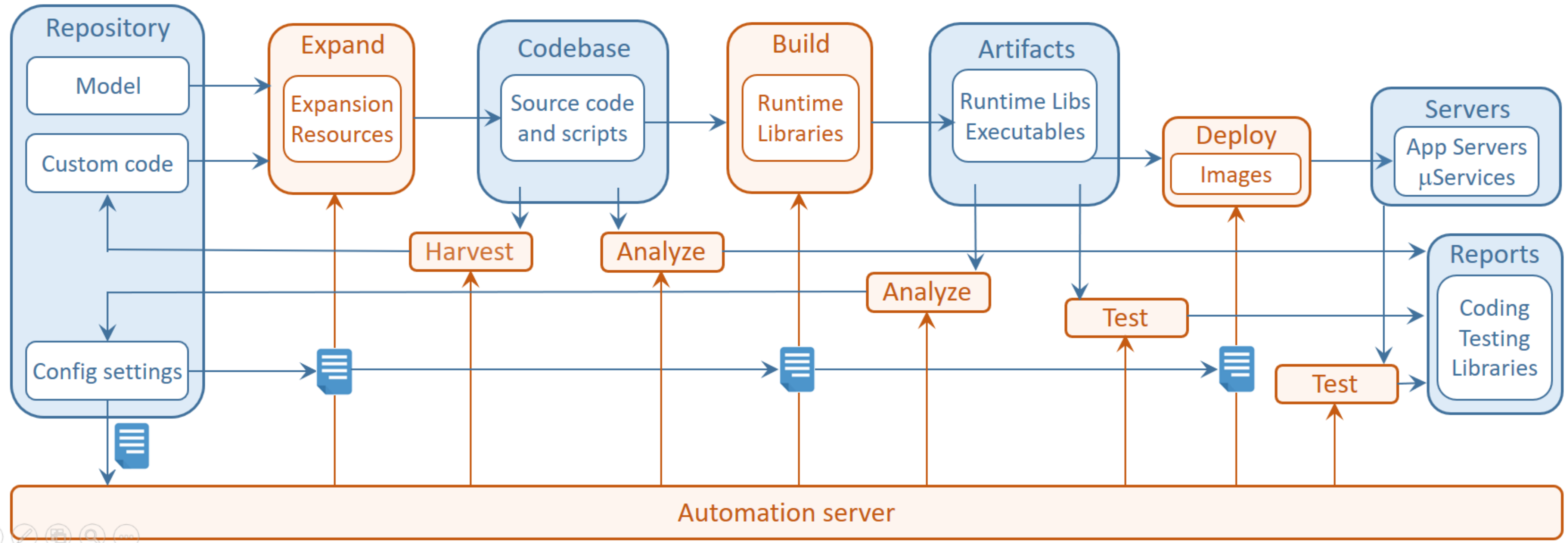
- More industrial assembly of software has been *pursued for decades*
  - Mass produced software components
  - Software product lines
  - Software factories
- Systematic reuse of software still faces *many issues*
- More challenging in a *code generation environment*
  - MDE, MDA
  - LCDP, NCDP
- NST software factory needs to support
  - *Harvesting* and *Re-injection*

From CI/CD to Continuous Rejuvenation



From CI/CD to Continuous Rejuvenation

- Need for an *expansion cycle before the build phase*



Normalized Systems Rejuvenation Modes

- *Structural rejuvenation along dimensions of variability*

- Upgrading *external frameworks* to new versions
  - Standard practice
  - NST may facilitate evolution of interface code
- Upgrading *expander skeletons* to new versions
  - From bug fixes to code improvements
  - To adding features and functionality
- Upgrading *infrastructure* to new frameworks
  - For existing or new cross-cutting concerns
  - For entire application landscapes

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

The Case of an NST Rejuvenation Factory

- Structural rejuvenation
  - According to different modes
  - For single, significant observation
  - Under normal market conditions

Application Domain	**Age** (*yrs*)	**Data Model** (*Nr. elem.*)	**Custom Code** (*Size kBytes*)
Energy Monitoring	> 10	116	6,352
	$3 - 5$	38	1,010
Power Grid Management	$1 - 3$	106	10,642
Human Resource Services	$3 - 5$	940	12,103
	$3 - 5$	59	1,433
Real Estate Services	> 10	491	70,449
	$1 - 3$	331	1,412
Unmanned Aviation	$5 - 10$	30	4,230
Traffic Management	$1 - 3$	134	2,896
Learning Management	$1 - 3$	133	1,794

TABLE I. Domain, lifespan, model and custom size of various applications.

The Case of an NST Rejuvenation Factory

- Continuous development
  - Applications in full or extended development
  - Several applications have dedicated expanders
  - Daily build and test, bi-weekly deployments
- Updating dependencies
  - Similar to traditional CI/CD, cadence as above
- Rejuvenating skeletons
  - Expanders follows same cadence
  - Rejuvenated skeletons in production (bi-)monthly
  - Structural rejuvenation of skeletons across application landscape, the CI/CD/CR has only been realized the last 4 to 5 years

The Case of an NST Rejuvenation Factory

- Replacing technologies
  - *Throughout the years*, support has been introduced in logic/data layer for
    - Additional databases
    - Additional providers for transactions, persistency, access control
  - *In the early years*, systematic migrations have been done in view/control layer
    - *MVC* → *MVC: Cocoon* to *Struts2*
    - *MVC* → *MVC-MVVM: Struts2* to *Struts2/Knockout*
  - *In recent years*, technologies were introduced without systematic migration
    - *JAX-RS* in control layer
    - *Angular* in view layer
  - Systematic migration seems to be hampered by *discipline creep*

Overview

- Introduction
- Software Maintenance and Evolvability
- The Premise of Normalized Systems Theory
- A Normalized Systems Software Factory
- The Case of an NST Rejuvenation Factory
- Conclusion and Future Work

Conclusion and Future Work

- We have presented an observational case study to evaluate the realization of the envisioned evolvability characteristics in an agile state-of-the-art NST software factory
- Contributions:
  - Described application of NST at scale in agile software factory
  - Validated that some levels of evolvability can be operationalized
  - Identified a concern that may hamper evolvability in realistic environment
- Limitations:
  - Case study factory was set up in the NST spin-off company
  - Software factory has only been operating at scale for a few years
- We plan to continue this study
  - In an extended time period
  - Operating at an increasing scale


QUESTIONS ?
herwig.mannaert@uantwerp.be