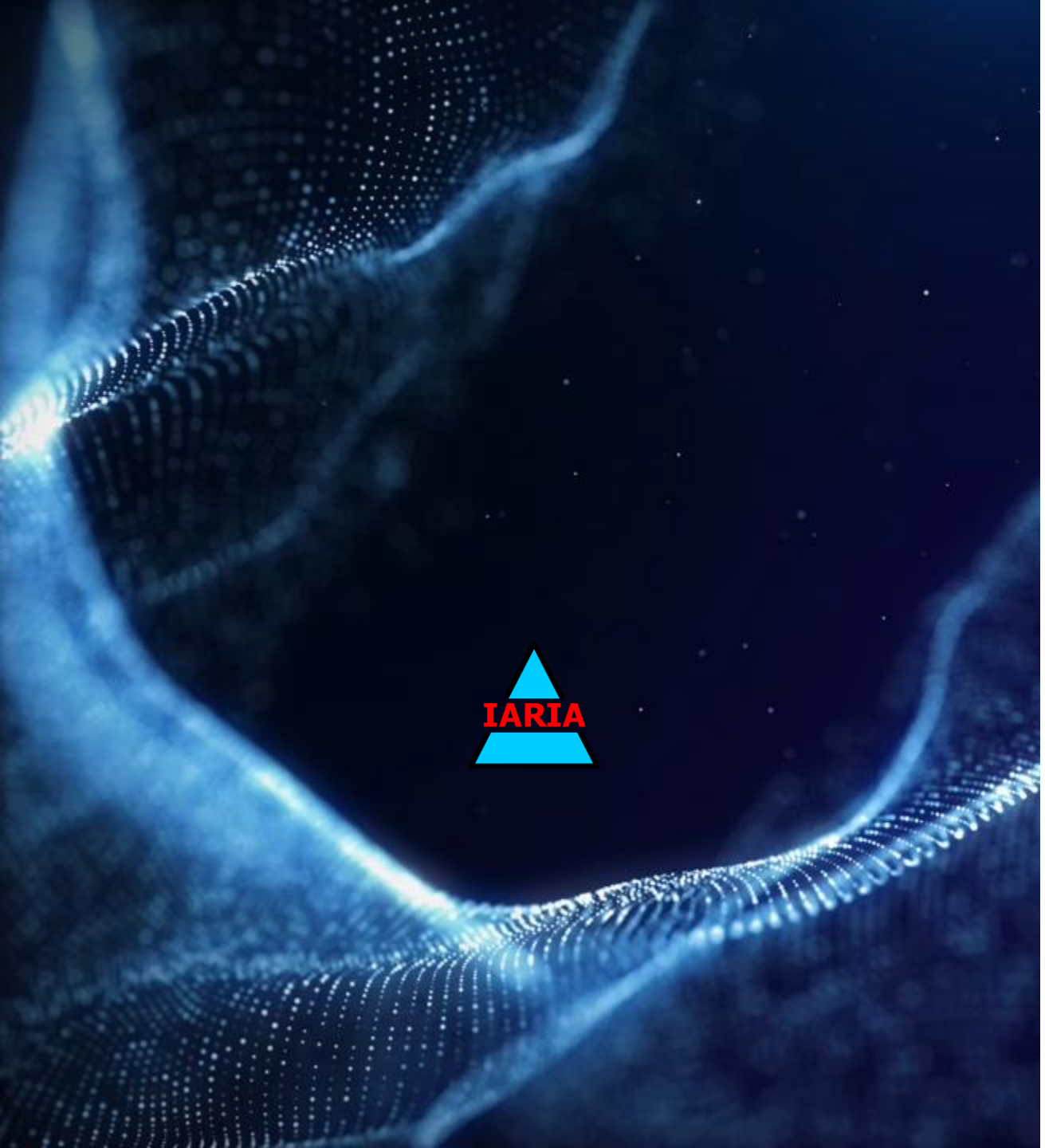


IARIA

COMPUTATIONWORLD 2023

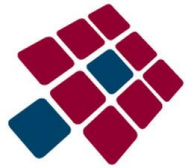
Combining DevOps and Normalized Systems Toward Software Rejuvenation Factories

HERWIG MANNAERT,

APRIL 16, 2024

Universiteit Antwerpen

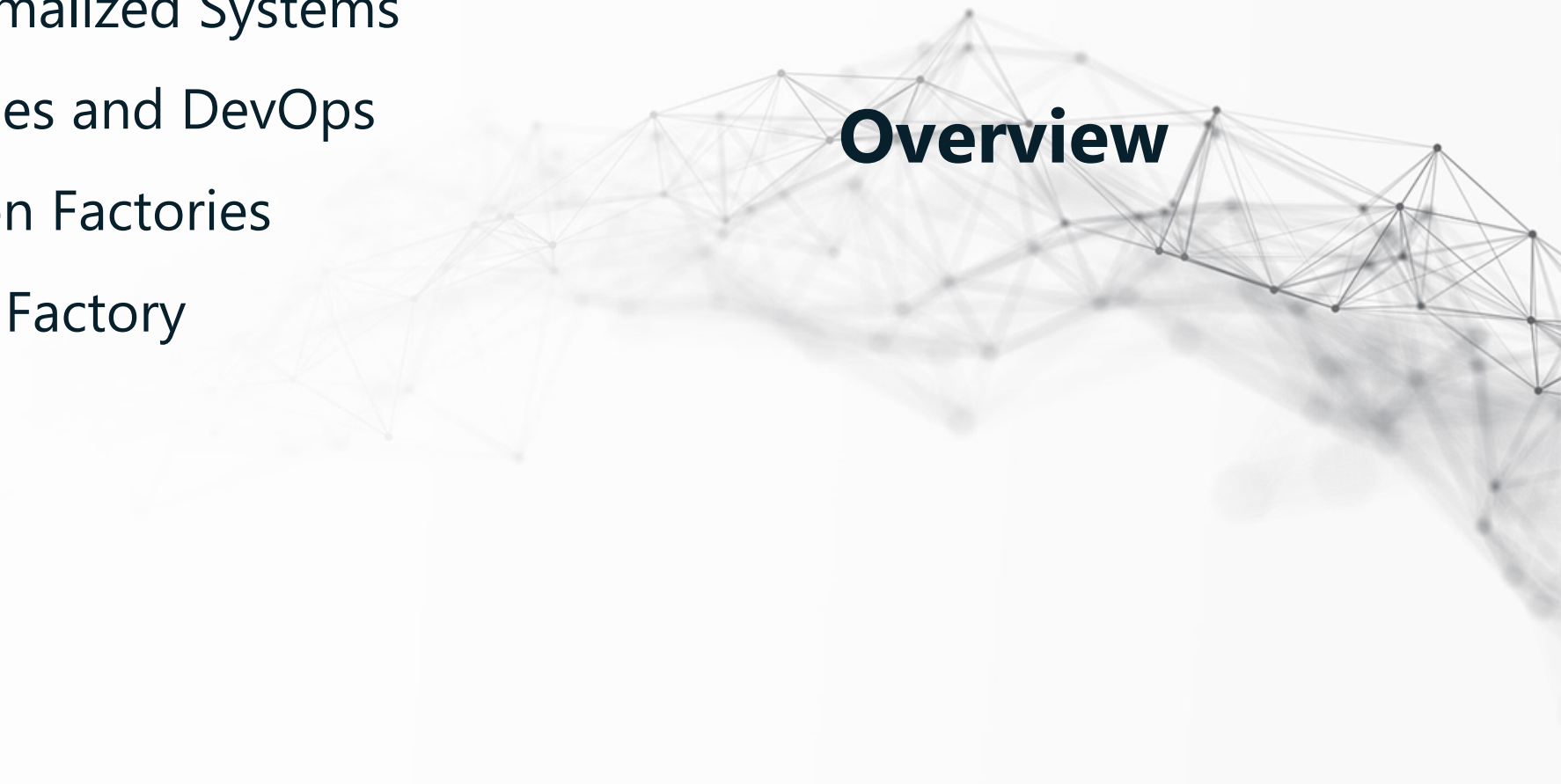
Intro on myself & my work



- 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 on University of Antwerp, not an esteemed researcher

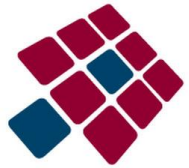
Overview

- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion



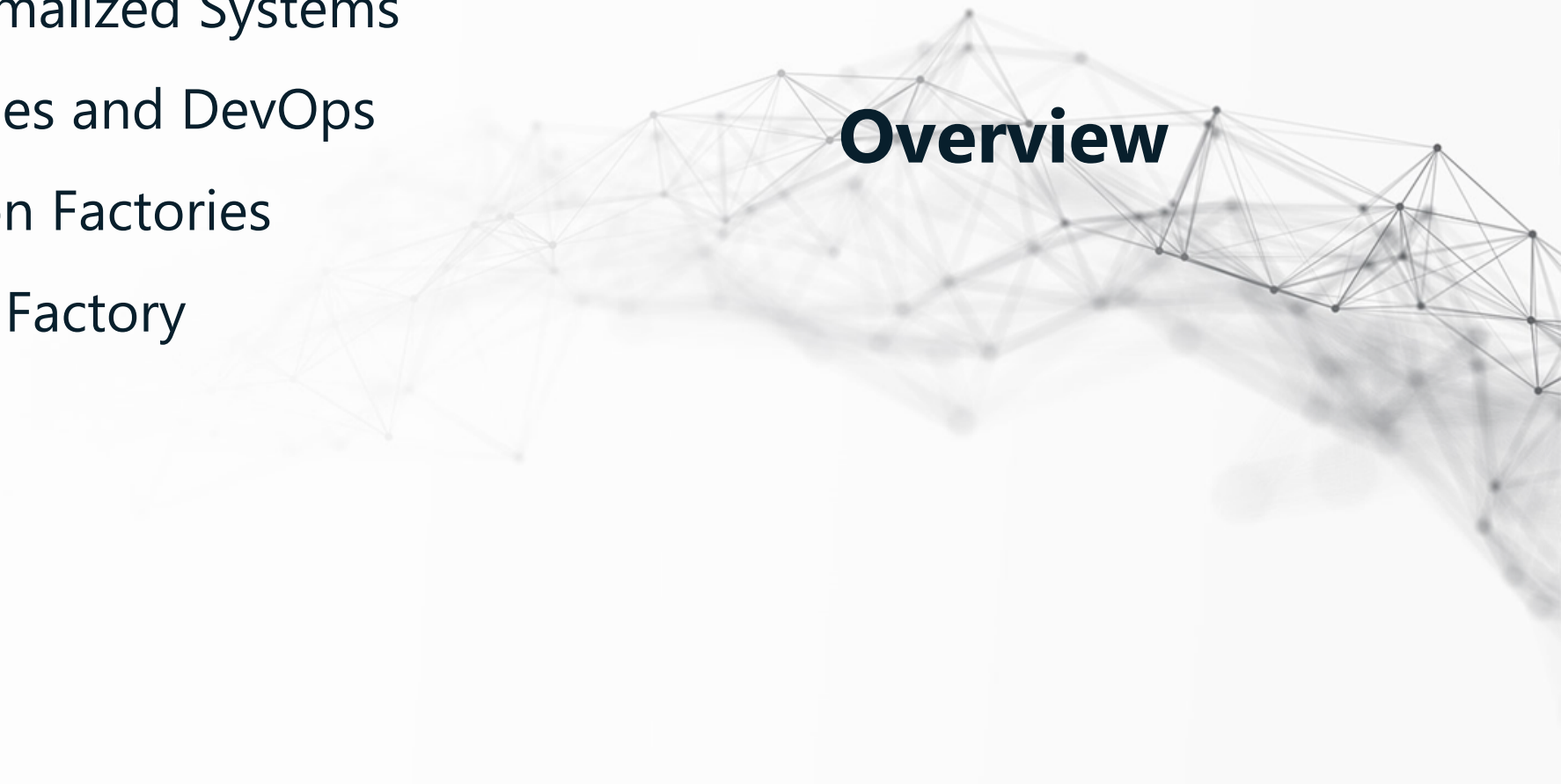
Overview

Introduction



- For decades, strong indications exist for *systemic issues in software evolution* and maintenance
- For decades, engineers have been *striving to produce software* in a more controlled and *industrial way*
- We have pursued the creation of more *evolvable software systems* through Normalized Systems Theory
- The current mainstream approach to organize the operations of so-called software factories is a methodology called DevOps
- We describe our approach *to combine NST and DevOps* to create evolvable systems at scale in so-called *rejuvenation factories*

Overview



The Law of Increasing Complexity.

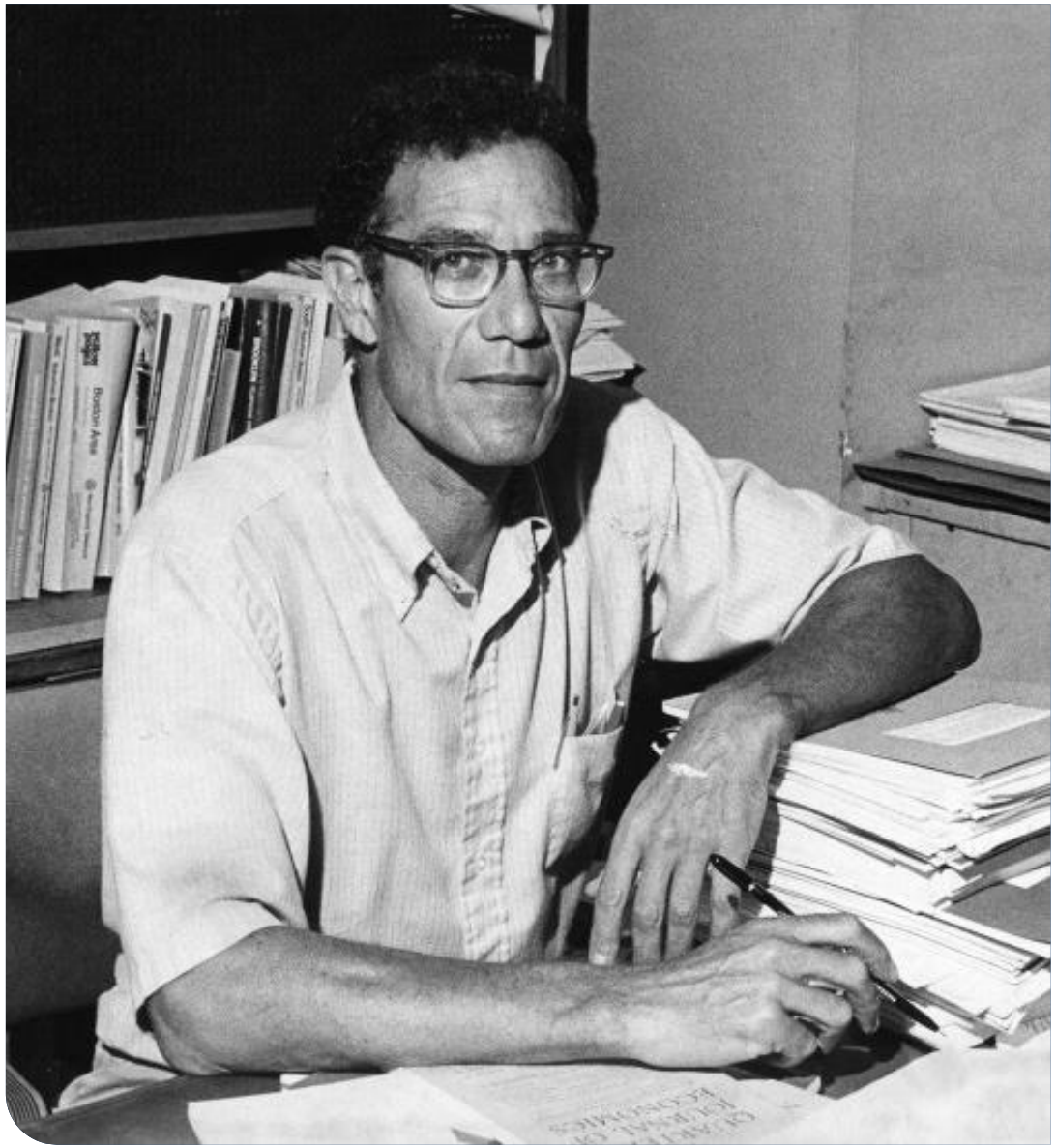
Manny Lehman

An Inconvenient Truth

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."

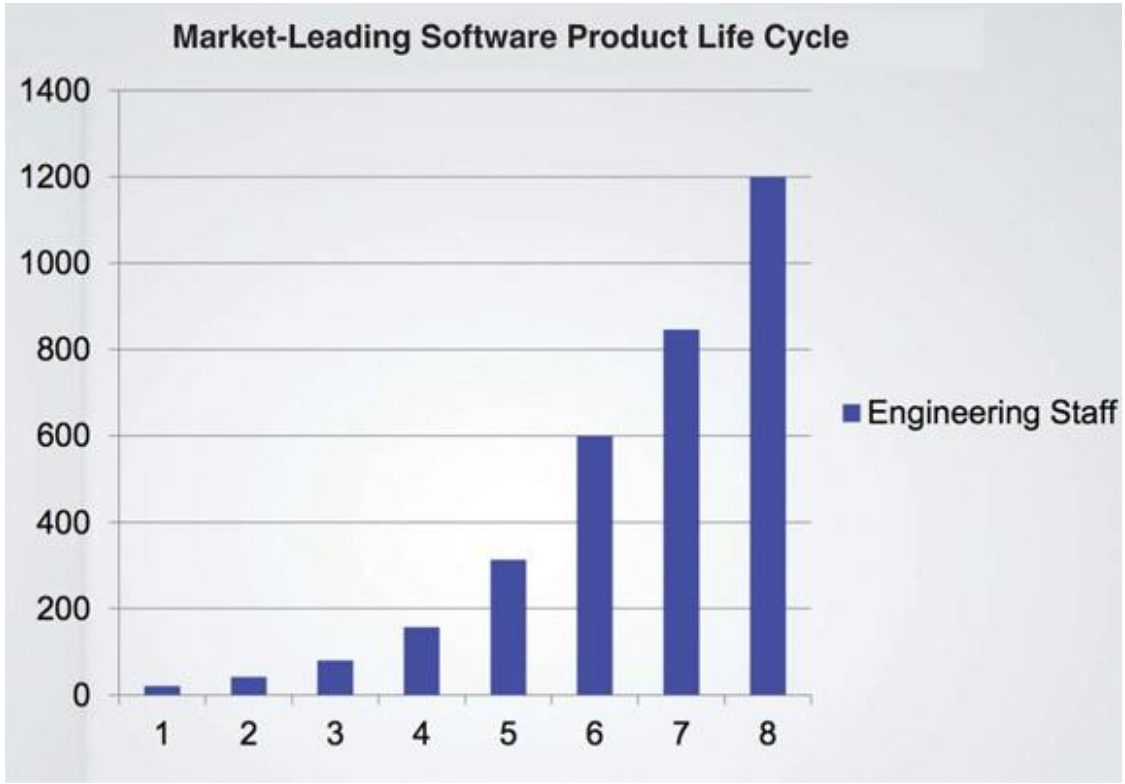
Proceedings of the IEEE, vol. 68, nr. 9, september 1980, pp. 1068.



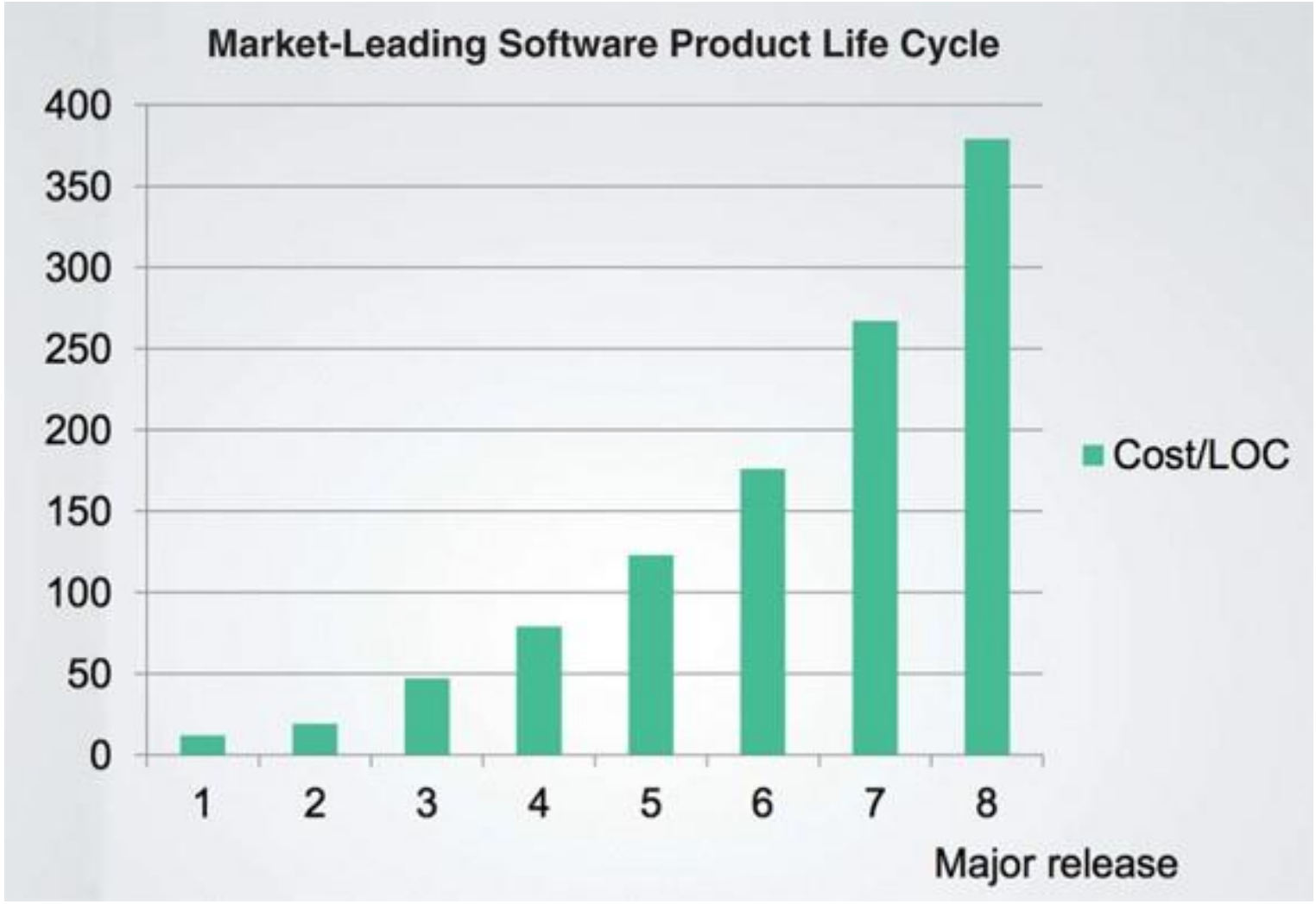
You can see the computer age everywhere but in the productivity statistics.

Robert Solow

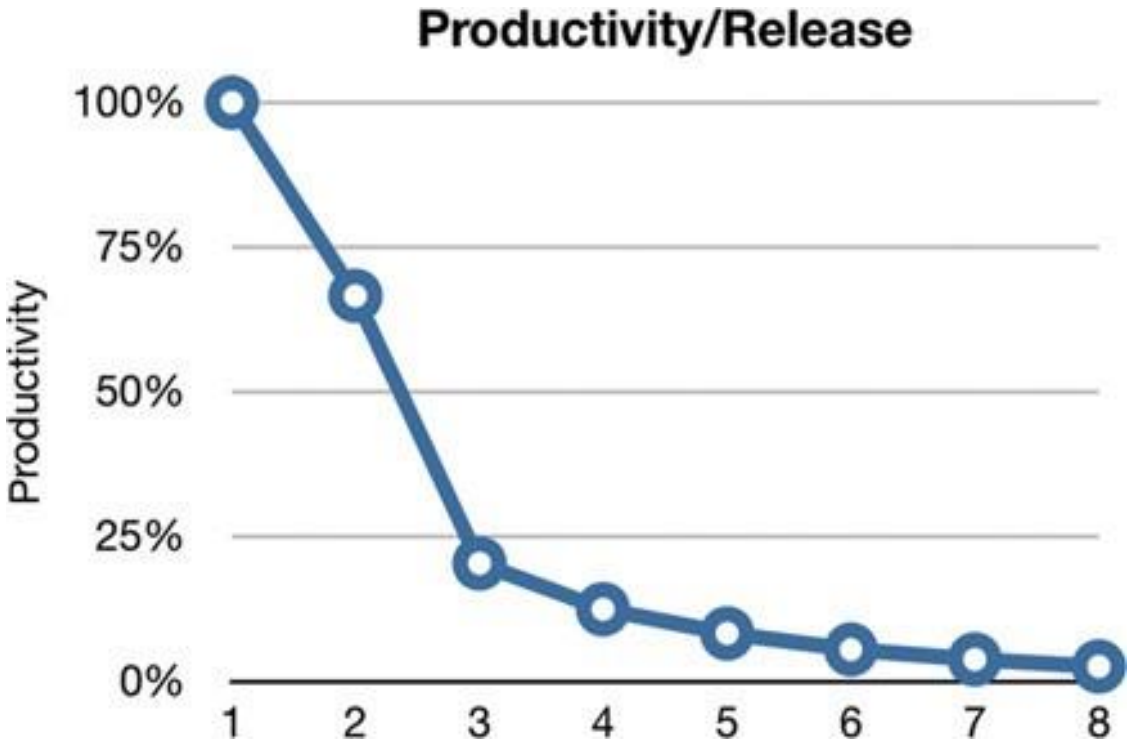
Clean Architecture – Robert C. Martin

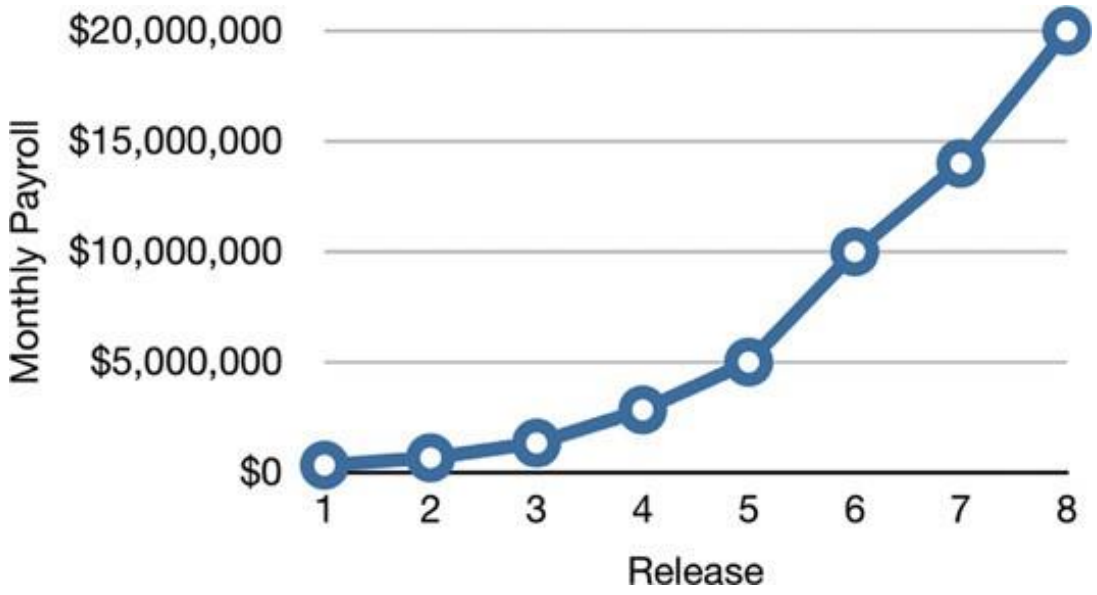


Clean Architecture – Robert C. Martin



Clean Architecture – Robert C. Martin





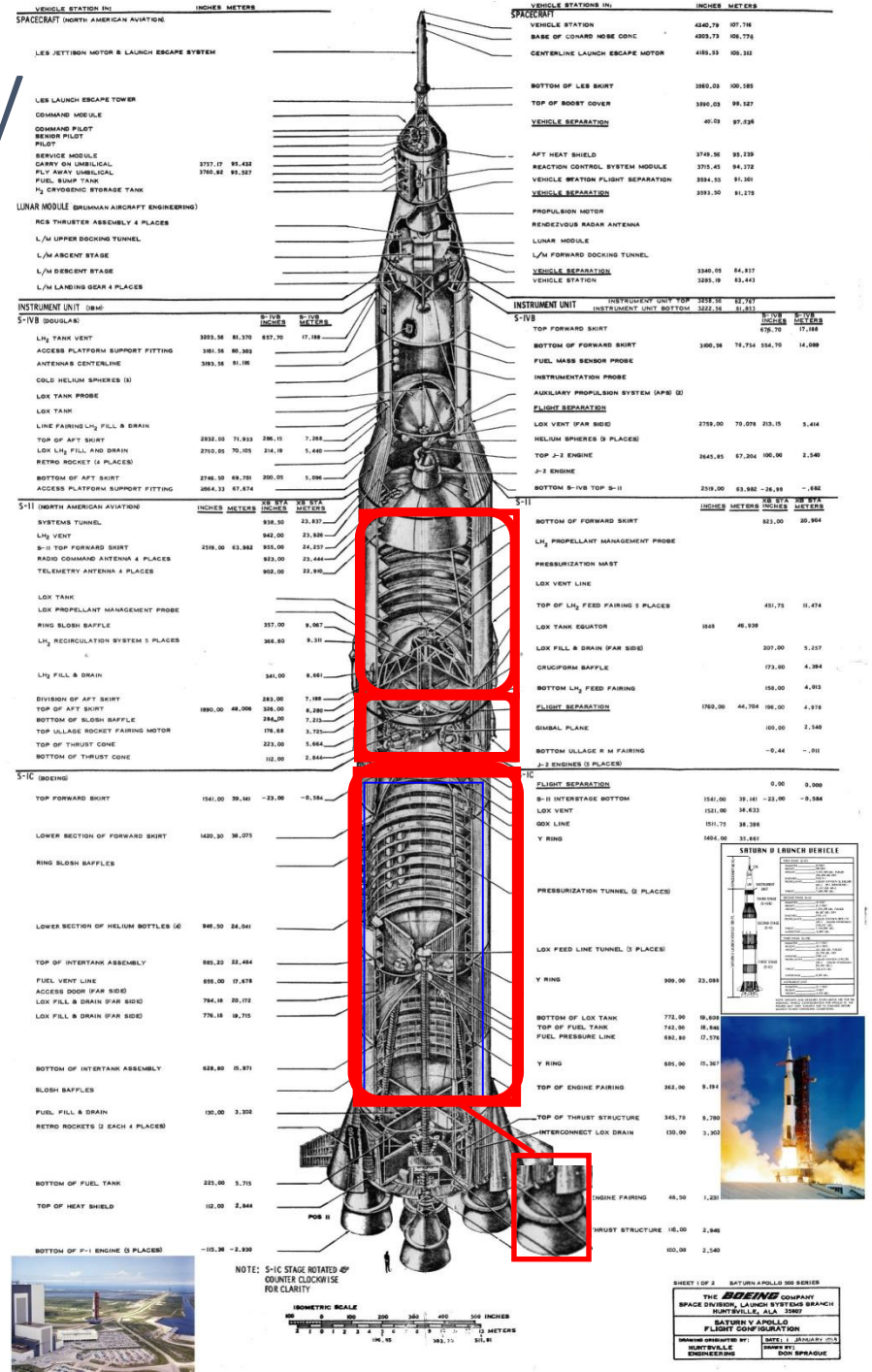
The Need for Software Evolution

- Laws of Manny Lehman
  - Increasing Complexity
  - Declining Quality
  - ….
- Solow's Productivity Paradox (2.0)
- Consequences of bad architecture by Robert C. Martin
- Dutch government *Elias Committee*
  - huge cost overruns, depreciation of IT systems after 7 years
- Concepts have been introduced like *technical debt*

But: Do not be ashamed !!

Change Ripples: The Saturn V

- Increase thrust power
  - Add additional F1
  - Additional fuel line
  - More powerful fuel pump
  - Larger fuel tank stage 1
  - Stronger fuel tank stage 1
  - Adapt shape fuel tank stage 1
  - Enlarge interstage & second stage
  - Adapt interstage & second stage
  - …
  - → Design new rocket



Change Ripples: A Racing Bike

- Gear handle worn out
  - Replace gear handle
  - Handle for 8 gears retired
  - New handles only 7 or 9 gears
  - Replace gear block in the rear
  - Replace gear cabling
  - Replace gear block in front
  - …
  - → Replace racing bike

Overview

- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion

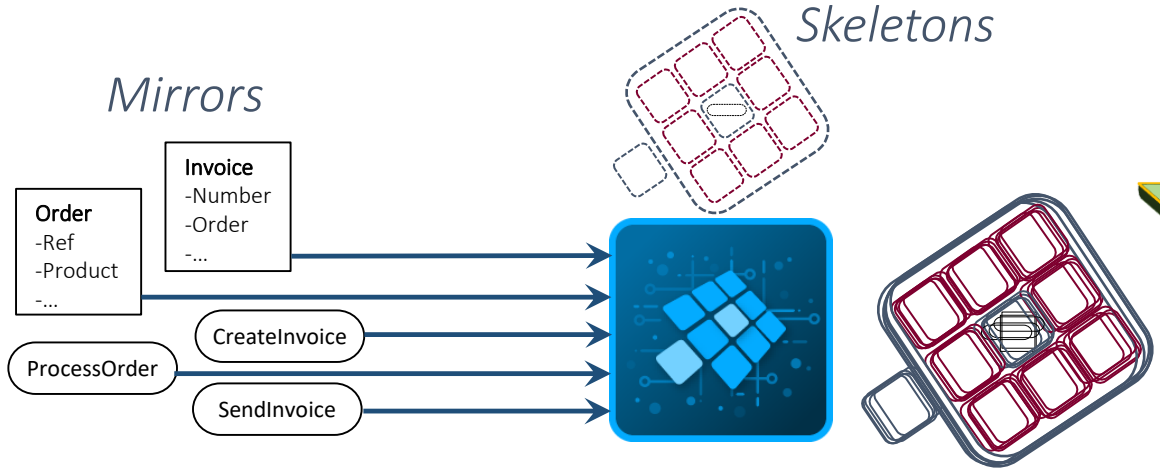
Design Theorems for Stable Software

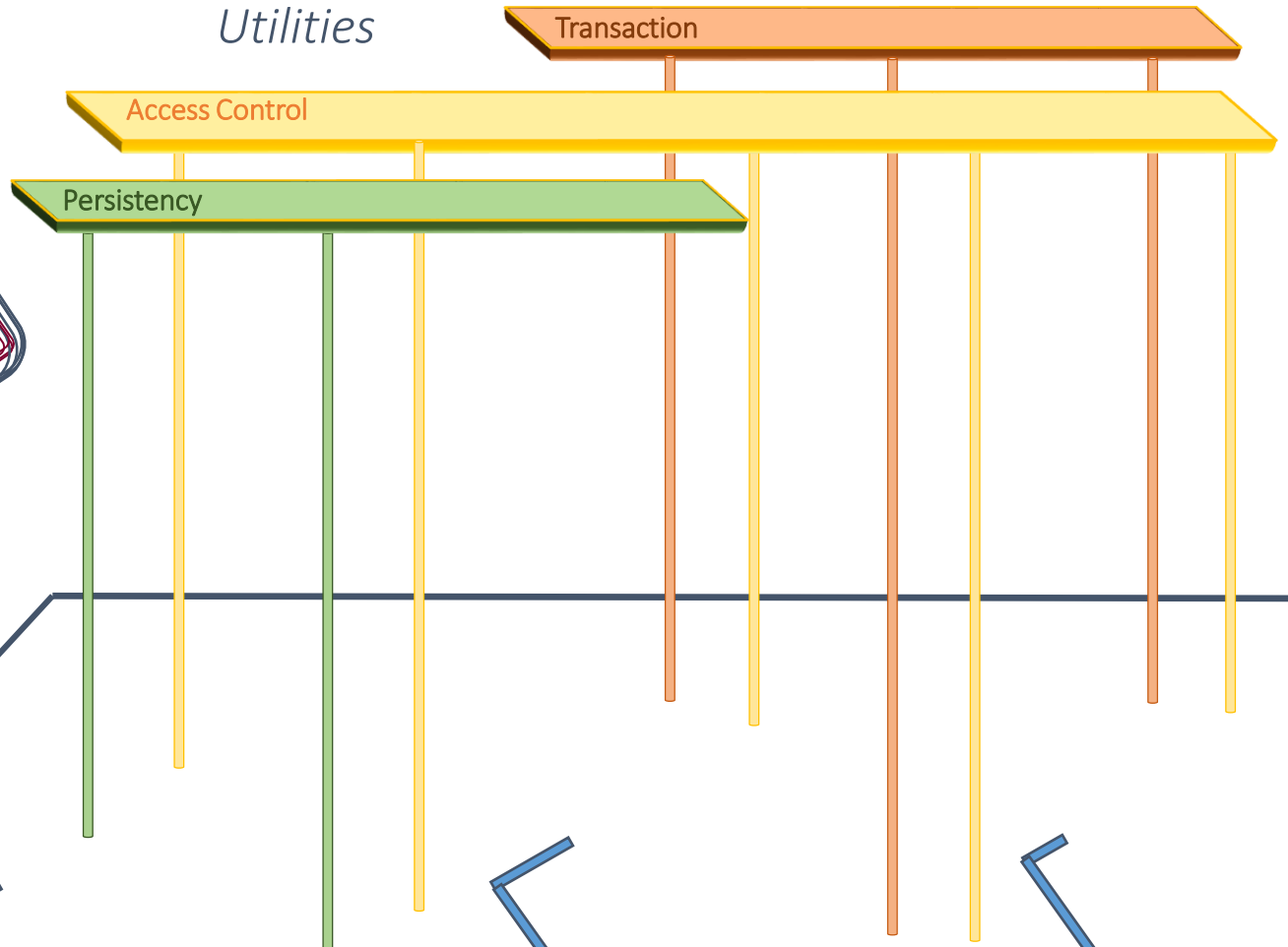
- In order to avoid dynamic instabilities in the software design cycle, the *rippling of changes needs to be depleted or damped: a = 0*
- As these ripples create *combinations of multiple changes* for every functional change, we call these instabilities ***combinatorial effects***
- Demanding systems 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*

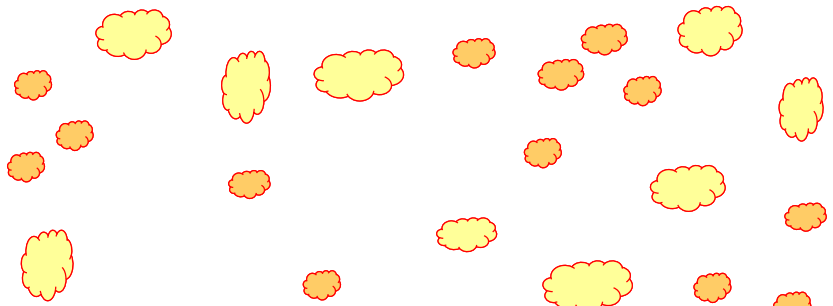
Software Elements for Stable Skeleton Structures

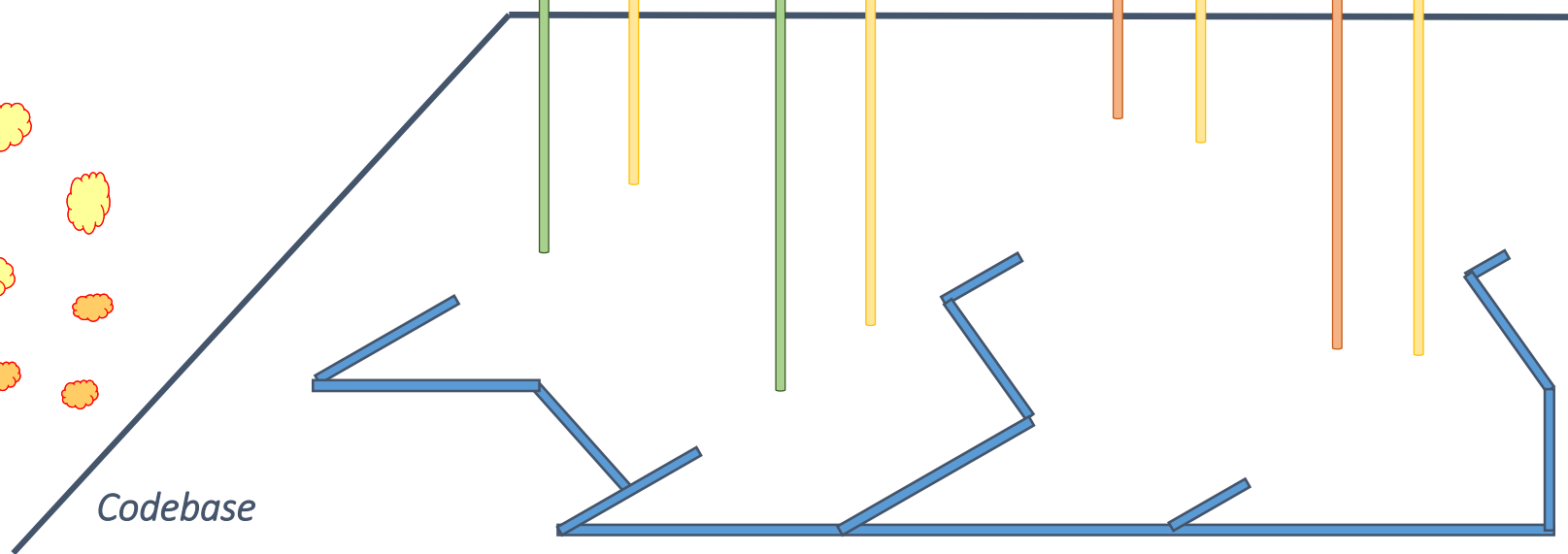
- Element structures are needed *to interconnect with CCC solutions*
- NS defines 5 types of elements, aligned with basic software concepts:
  - ***Data elements***, to represent data variables and structures
  - ***Task elements***, to represent instructions and/or functions
  - ***Flow elements***, 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









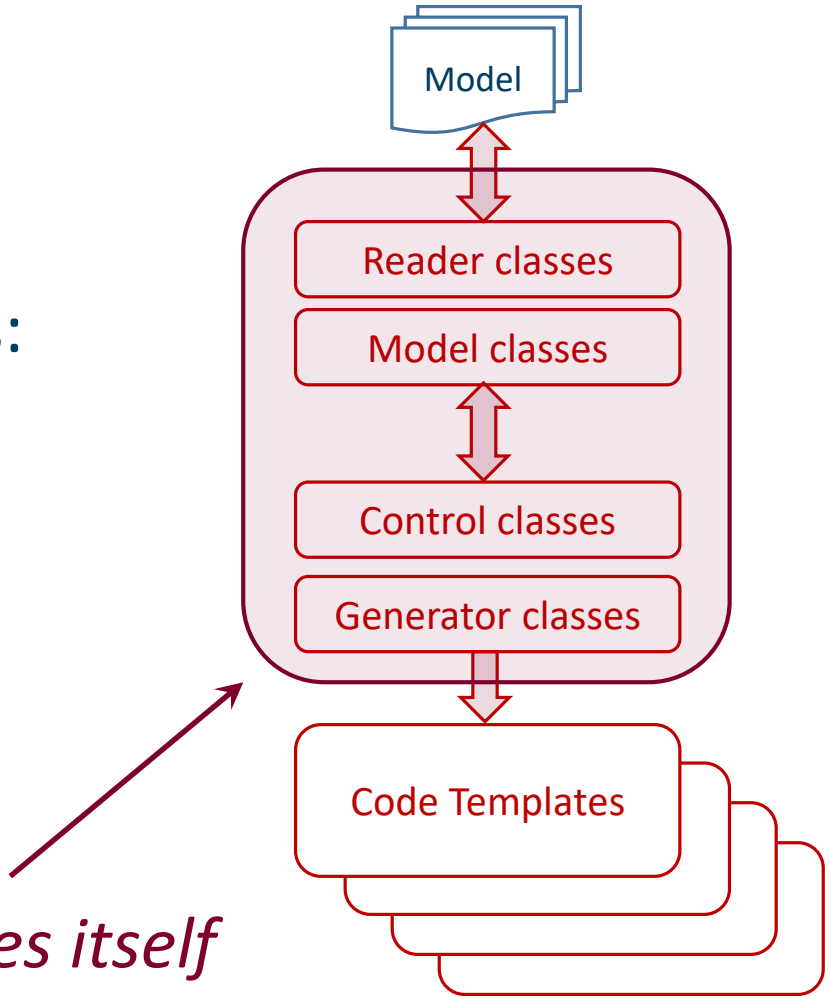
The Essence of Variability Dimensions

- 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 x #E x #I x #C
  - Where the same holds within any individual dimensions,

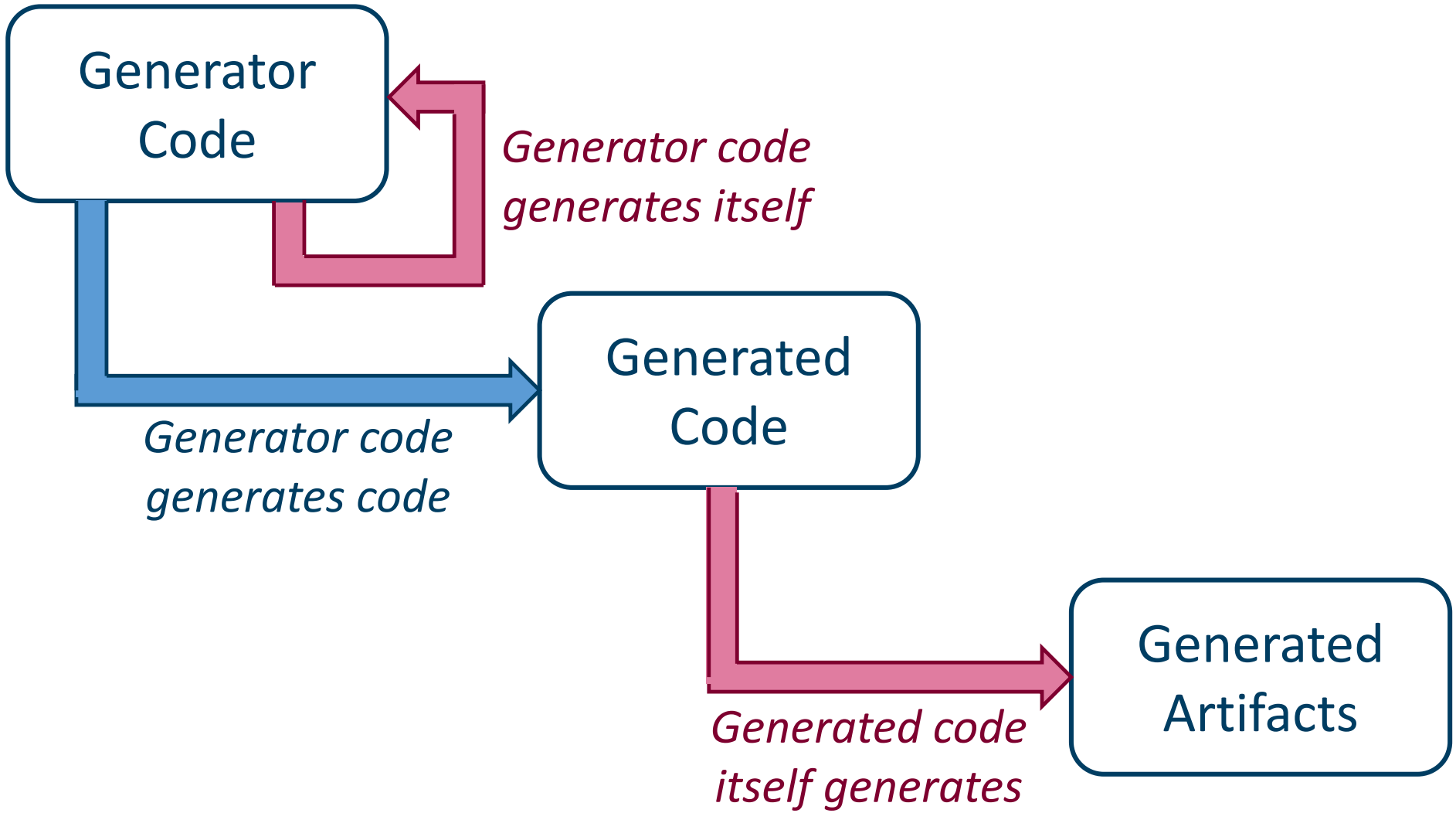
    e.g., infrastructure dimension: #I = #G x #P x #B x #T

But what about the generator code ?

- You also have to maintain the meta-code
  - Consists of several modules
  - Is in general not trivial to write
- Will face growing number of implementations:
  - Different versions
  - Multiple variants
  - Various technology stacks
- Will have to adapt itself to:
  - Evolutions of its underlying technology
    - Which even may become obsolete
- *Meta-Circularity: meta-code that (re)generates itself*

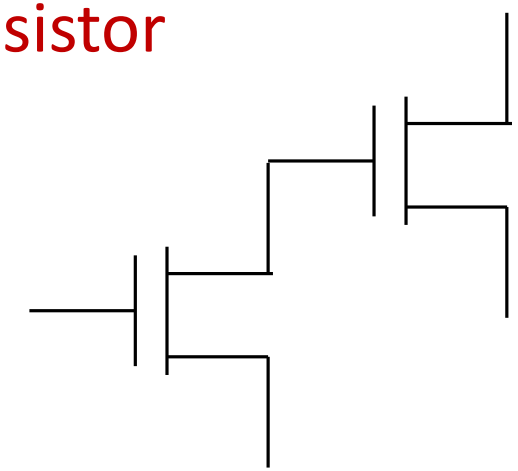


Creating Meta-Circular and Runtime Generation

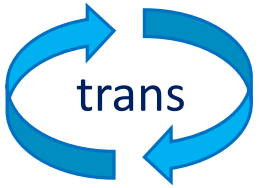


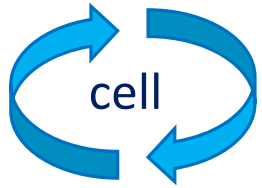
The Power of Circularity

- A transistor is switched by a transistor
- A cell is produced by a cell



- Enables rapid evolution
  - Single point of progress
    - Better transistor → better circuits
    - Improved cell → improved life forms
  - Collapses/shortcuts the design cycle
    - Even *positive feedback* or *resonance*





- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion

Overview

Mass Produced Software Components.

Doug McIlroy

On Software Factories and Reusability

- Produce and assemble software in more industrial way
  - Mass produced software components *(McIlroy)*
  - Software Product Lines *(SEI)*
    - Predicted versus opportunistic reuse
  - Software Factory *(Greenfield et al.)*
    - Techniques of traditional manufacturing
- Systematic reuse of software is not trivial
  - Methodological issues *(Saeed)*
  - Issues related to evolvability *(NST)*
    - Rippling of impacts due to new versions and variants

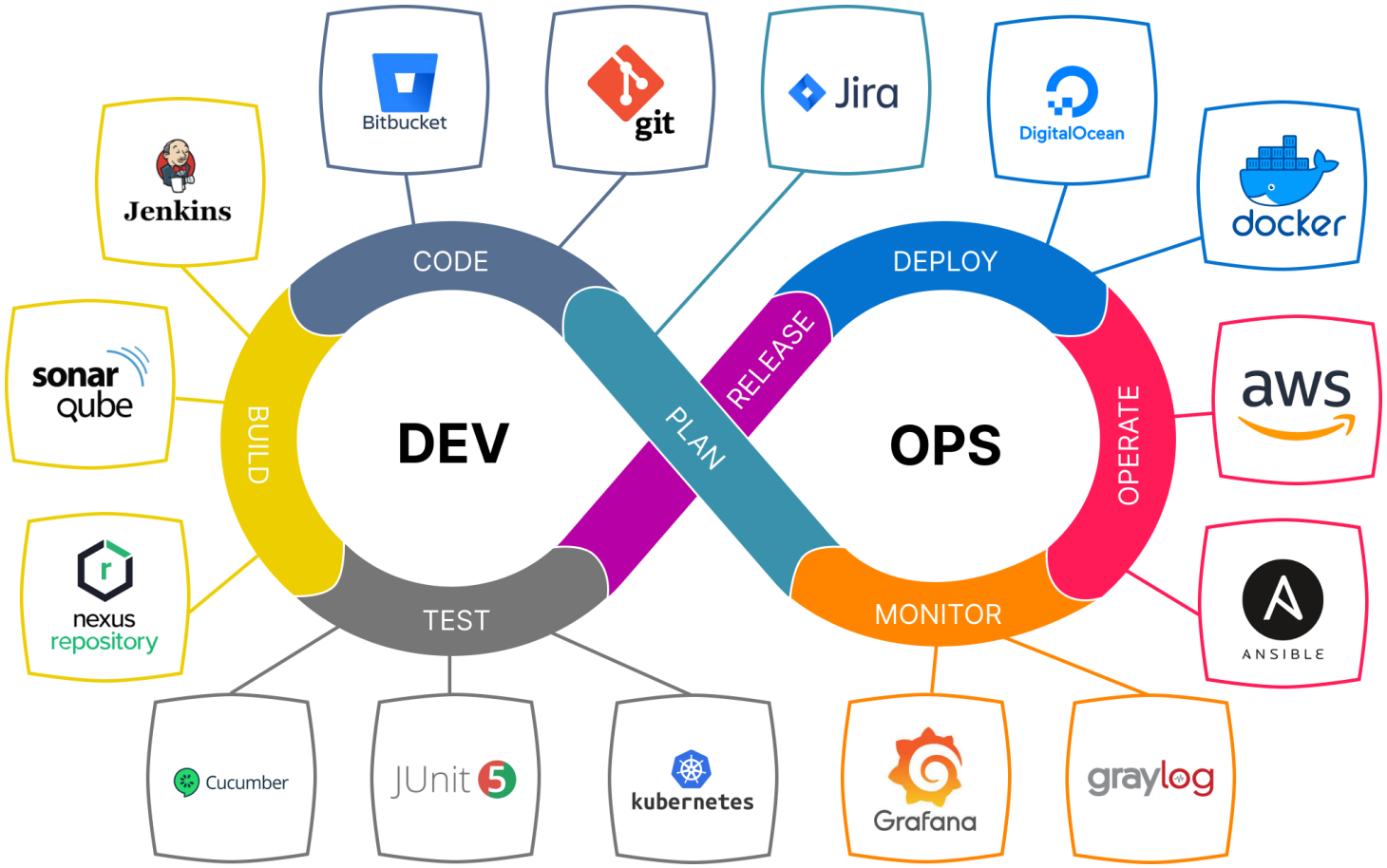
Building a factory is ‘100 times’ as hard as building a car.

Elon Musk

Software Factories and DevOps

- The current mainstream approach to organize and control the operations of so-called software factories is *a methodology called DevOps* to integrate and automate the work of software development (Dev) and IT operations (Ops).
- DevOps integrates and automates the work of software development (Dev) and IT operations (Ops) as a means for improving and shortening the systems development life cycle
  - an assumption that all functions can be carried out, controlled, and managed in a central place using a simple code
  - tools in Continuous Integration Continuous Deployment (CICD) infrastructure are in general numerous and versatile

Today : a Typical DevOps Environment



- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion

Overview

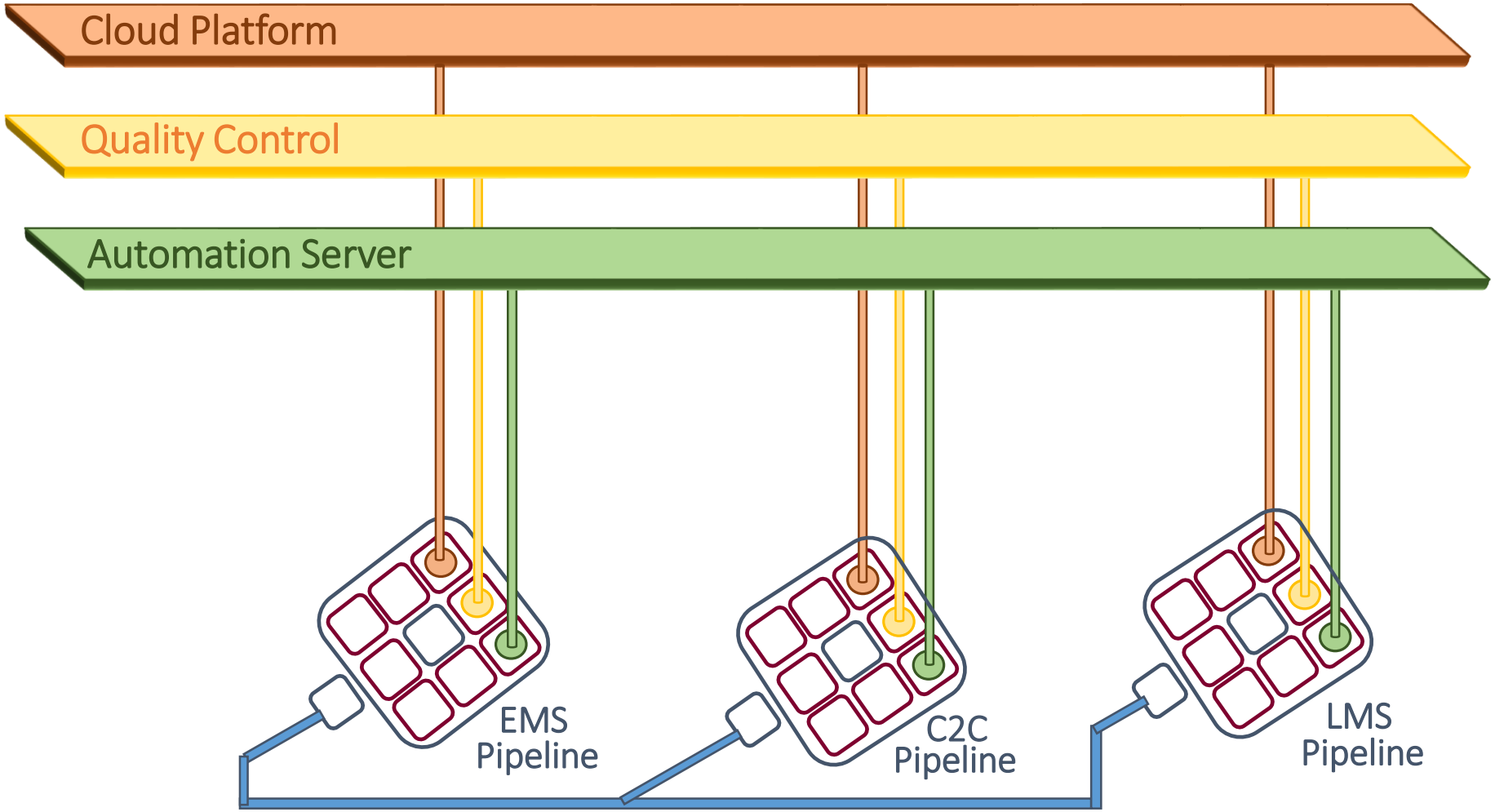
From DevOps to Evolvable Pipelines

- A clear need exists for the *structured separation and encapsulation* of the various functional tasks performed by numerous versatile *tools*

→ *similar to encapsulation of cross-cutting concerns in NS elements*

- Various tasks should be model-driven with tool implementation(s)
  - Automation server, e.g., Jenkins, Tekton
  - Build engine, e.g., Maven, Bamboo
  - Automated testing, e.g., Junit, Cucumber
  - Automated deployment, e.g., Docker, Ansible
  - Quality Control, e.g., SonarQube
  - …

From DevOps to Evolvable Pipelines



From DevOps to Integrated Control Systems

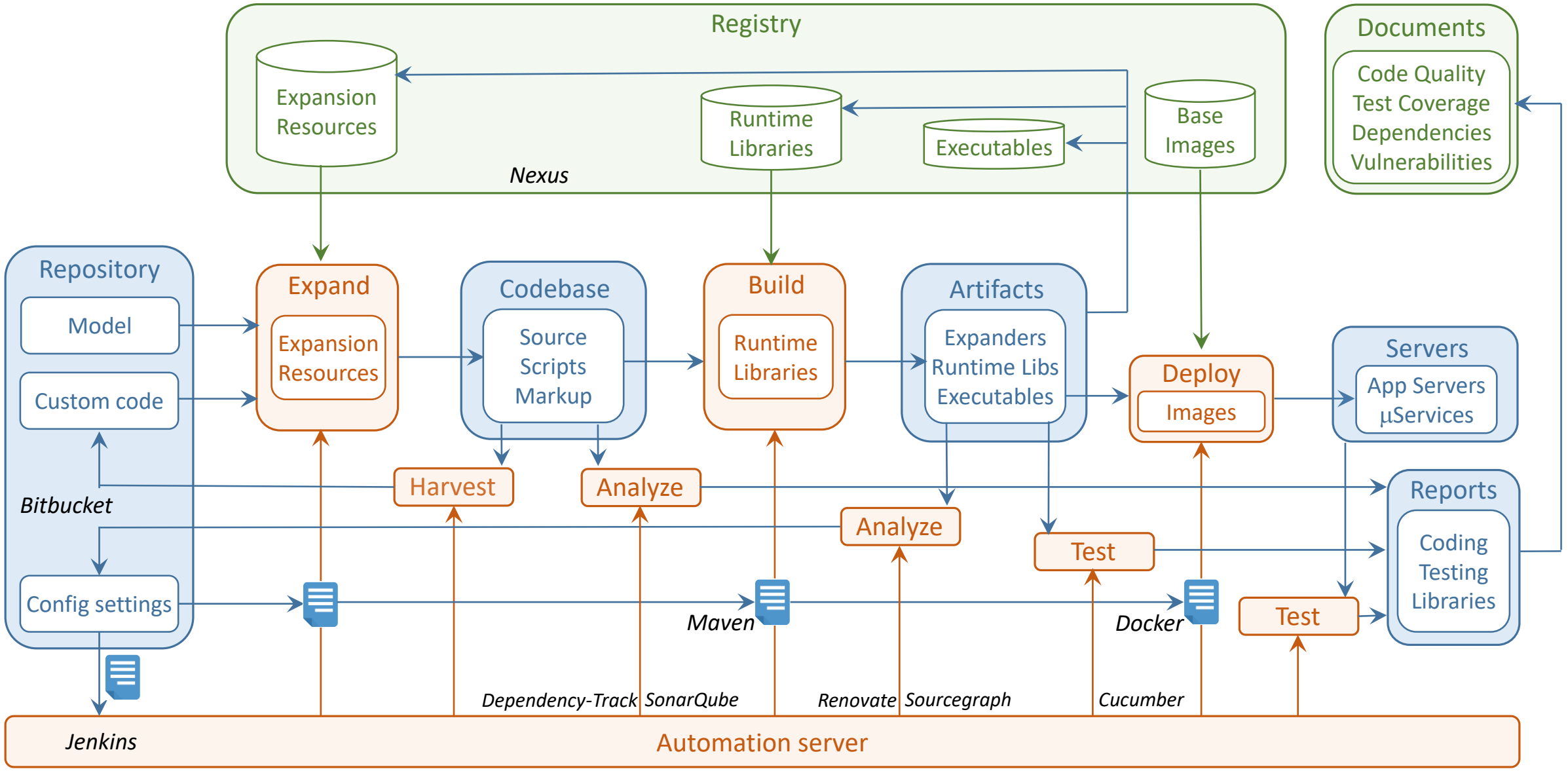
- A clear need exists for *integrated control systems, to manage and control end-to-end the building and assembly of software systems*
- Such a system needs to encompass the various processes and tools, and therefore allow
  - to breakdown DevOps to increase security and reduce technical debt
  - to offer a *SBOM (Software Bill Of Materials)* for quality assurance
- This is similar to
  - *MES (Manufacturing Execution Systems)* systems, that track and document the transformation of raw materials to finished goods
  - *SCADA (Supervisory Control and Data Acquisition)* systems that control production processes in real-time, in manufacturing

From DevOps CI/CD toward CI/CD/CR

- An NST software factory needs to encompass assembly lines for
  - NST code generators, e.g., expansion resources, runtime libraries
    - Runtime libraries and expansion resources
  - NS software applications, e.g., web information systems, tools and plugins
- An NST software factory needs to integrate build processing steps
  - *Expanding* and building
  - Unit testing and reporting
  - Deploying and integration testing
- An NST software factory needs to support
  - *Harvesting* and *Re-injection* of custom code

  → ***Systematic rejuvenation***

From DevOps to Software Assembly Units



Overview

- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion

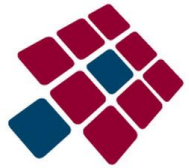
The State of our Rejuvenation Factory

- 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 State of our Rejuvenation Factory

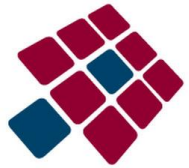


- Continuous development
  - Applications in full or extended development
  - Several applications have dedicated expanders
  - Daily build and test, bi-weekly deployments
- Updating dependencies
  - Similar to traditional CI/CD, cadence as above
- Rejuvenating skeletons
  - Expanders follows same cadence
  - 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 State of our 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*

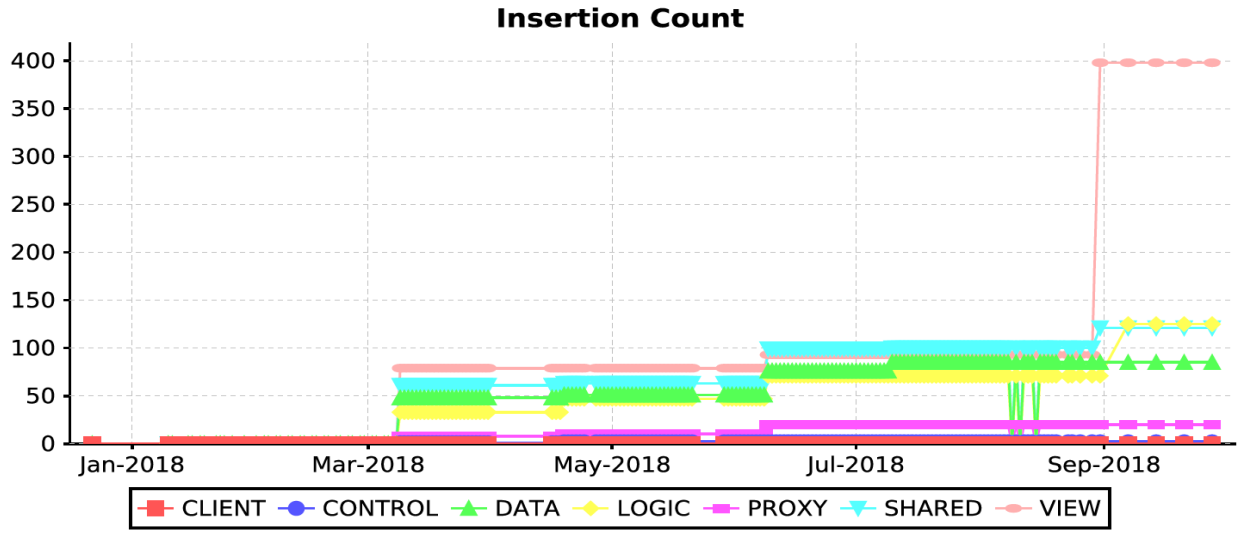
The State of our Rejuvenation Factory

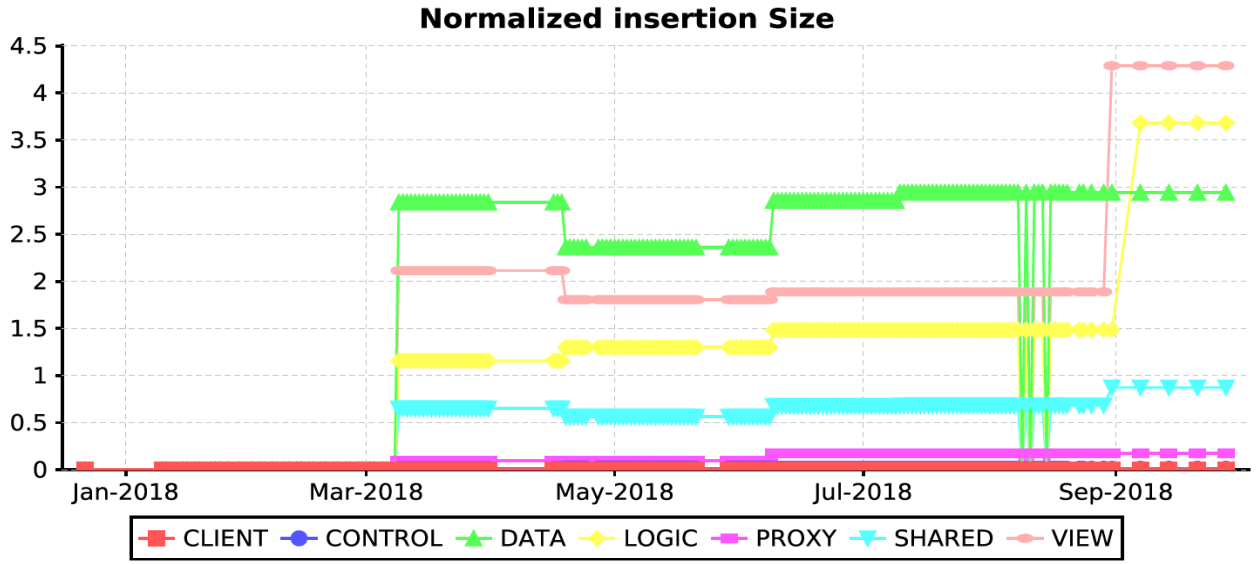


- A *Software Manufacturing Control System* is being developed
  - *Aggregated views* are provided across
    - *Time*
    - *Assembly unit hierarchy*
      - components, libraries
      - Expander bundles
  - *Aspect views* are provided based on
    - Custom code
    - Quality metrics
    - Test coverage
    - Model size
    - Dependencies
    - …

The State of our Rejuvenation Factory

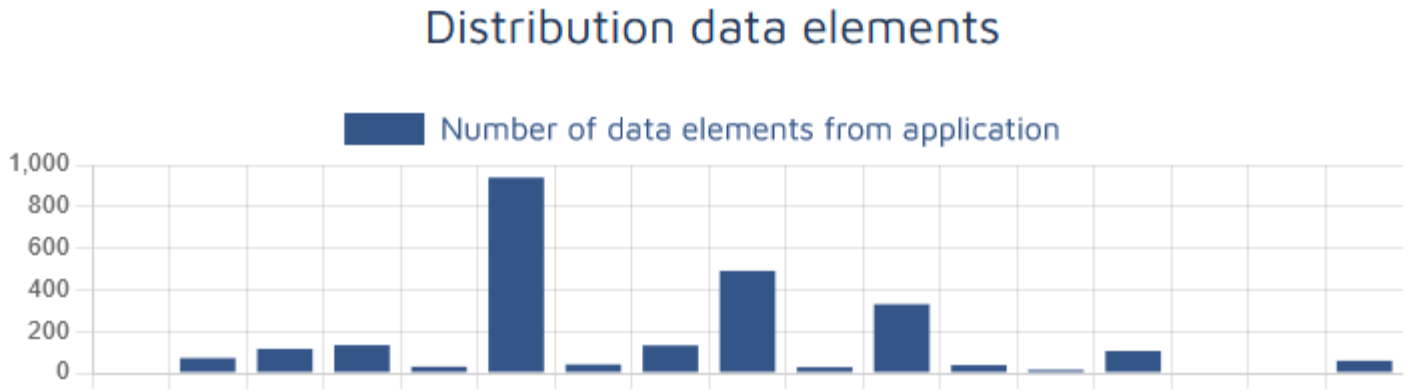
- Aggregated views

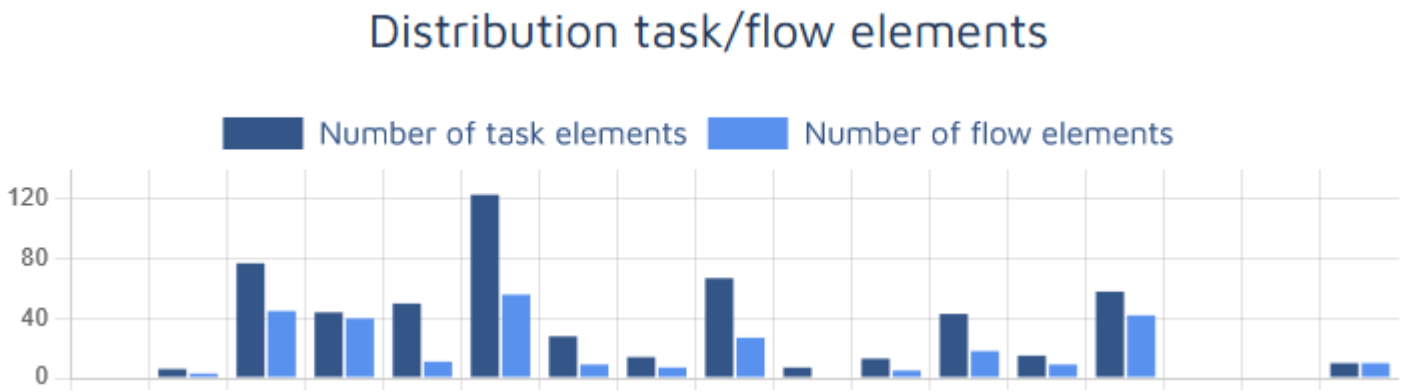




The State of our Rejuvenation Factory

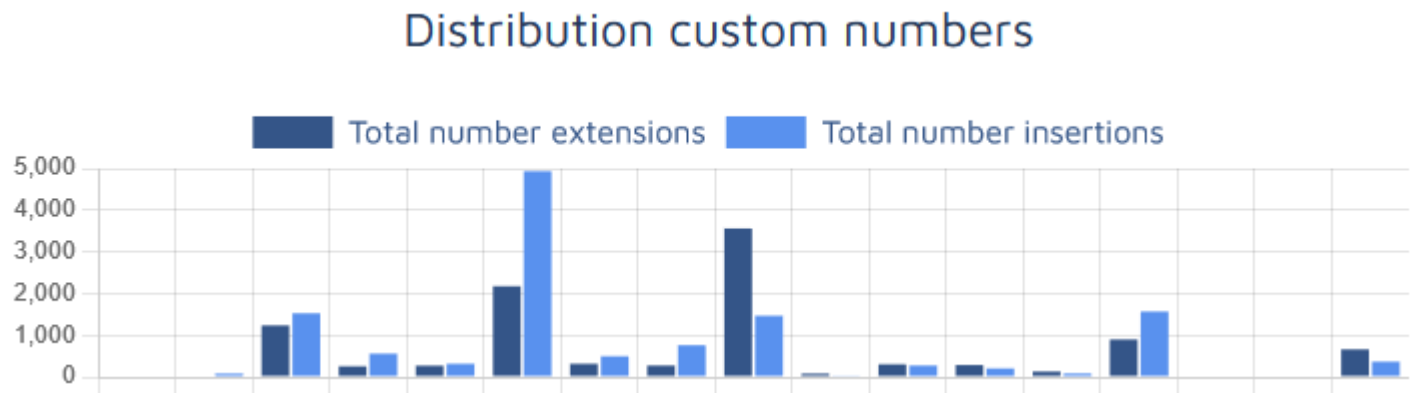
- Aggregated views

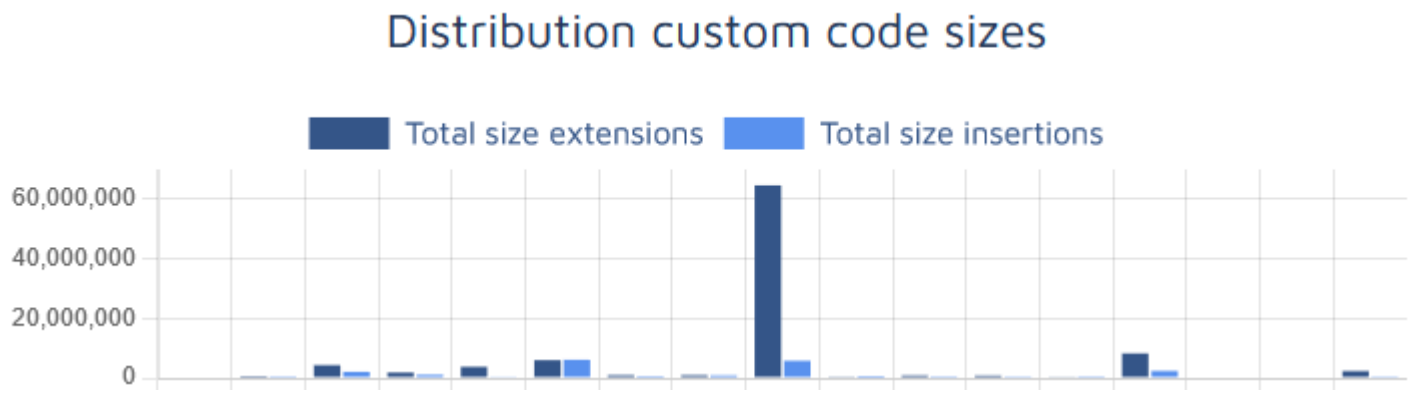




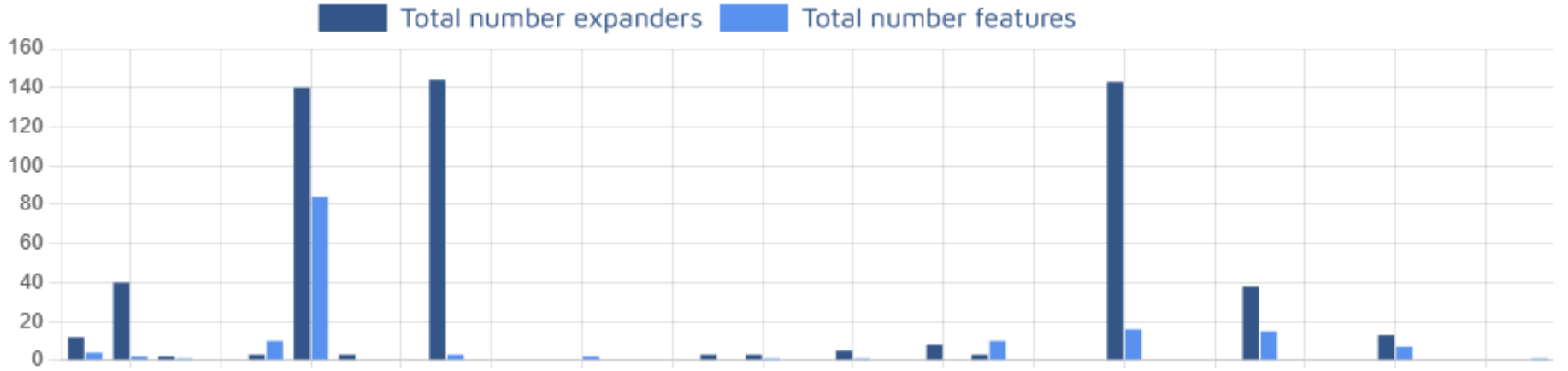
The State of our Rejuvenation Factory

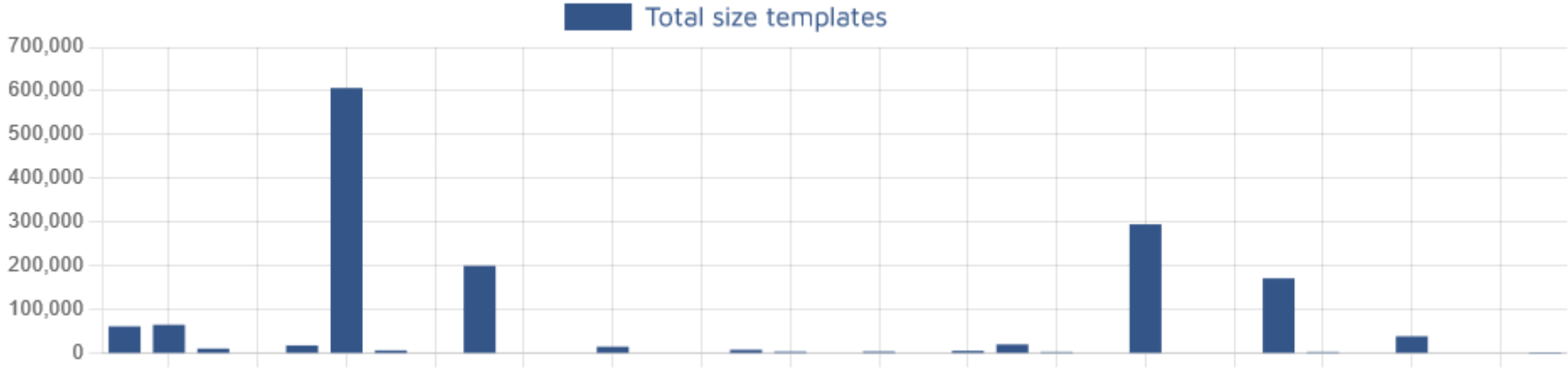
- Aggregated views





On the State of our Rejuvenation Factory



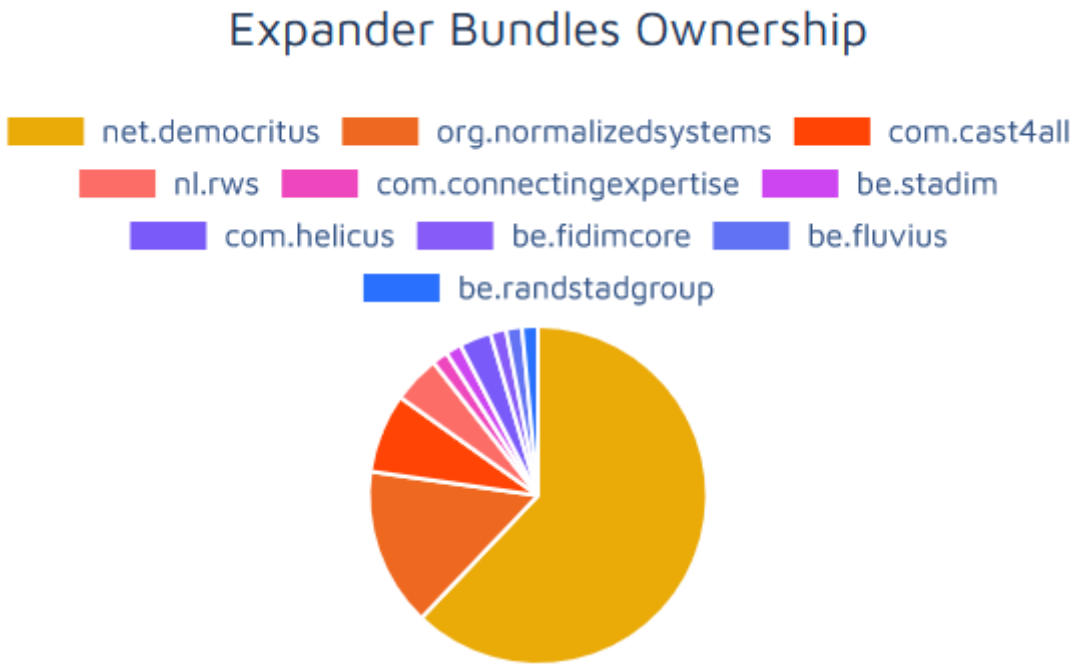


The State of our Rejuvenation Factory

- A control layer should allow to *optimize conditions and improve output* in the software factory, ideally on a continuous basis
  - Surfacing the various views on a continuous basis could facilitate
    - Instantaneous assessment of exposure to vulnerabilities
    - Continuous assessment of impact when retiring technologies
    - Continuous assessment of importance of in-house libraries

    and *guide the allocation of development resources*
  - Including the expanders in the factory control system could be pivotal for
    - *Trust* in the use of (additional) expander bundles
    - Adoption of *expander bundles from partners* into the factories

On the State of our Rejuvenation Factory



Overview

- Introduction
- The Need for Software Evolution
- The Premise of Normalized Systems
- On Software Factories and DevOps
- Toward Rejuvenation Factories
- On the State of our Factory
- Conclusion

Conclusion

- We have argued that strong indications exist for *systemic issues in software evolution* and maintenance
- Our work on NST aims to create more evolvable software systems
- Engineers have been *striving to produce software* in a more controlled and *industrial way*, currently using a methodology called DevOps
- We have described our approach *to combine NST and DevOps* to create *evolvable software systems at scale* in so-called *rejuvenation factories*
- An overview was given of the *current state of our rejuvenation factory*, including some key figures, control mechanisms, and potential issues

Some References

- Mannaert Herwig, McGroarty Chris, Gallant Scott, De Cock Koen, **Integrating Two Metaprogramming Environments : An Explorative Case Study :** ICSEA 2020 - ISSN 2308-4235 - IARIA, 2020, p. 166-172
- Mannaert Herwig, De Cock Koen, Uhnak Peter, **On the realization of meta-circular code generation : the case of the normalized systems expanders,** ICSEA 2019 - ISSN 2308-4235 - IARIA, 2019, p. 171-176
- De Bruyn Peter, Mannaert Herwig, Verelst Jan, Huysmans Philip, **Enabling normalized systems in practice : exploring a modeling approach**, Business & information systems engineering - ISSN 1867-0202 - 60:1(2018), p. 55-67.
- Mannaert Herwig, Verelst Jan, De Bruyn Peter, **Normalized systems theory : from foundations for evolvable software toward a general theory for evolvable design**, ISBN 978-90-77160-09-1 - Koppa, 2016, 507 p.
- 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
- 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
- Mannaert Herwig , De Bruyn Peter, Verelst Jan, **On the Interconnection of Cross-Cutting Concerns within Hierarchical Architectures**, IEEE Transactions on Engineering Management - ISSN 1558-0040 - 69:6(2022), p. 3276-3291.
- **Normalized Systems Foundation Lectures** : https://www.youtube.com/c/normalizedsystems
- **Normalized Systems Documentation and Tooling :** https://foundation.stars-end.net

QUESTIONS ?

herwig.mannaert@uantwerp.be