28th Annual INCOSE international symposium

Washington, DC, USA
July 7 - 12, 2018

Reverse Engineering Legacy Software in a Complex System: A Systems Engineering Approach

Maximiliano Moraga & Yang-Yang Zhao
University of Southeast Norway

www.incose.org/symp2018

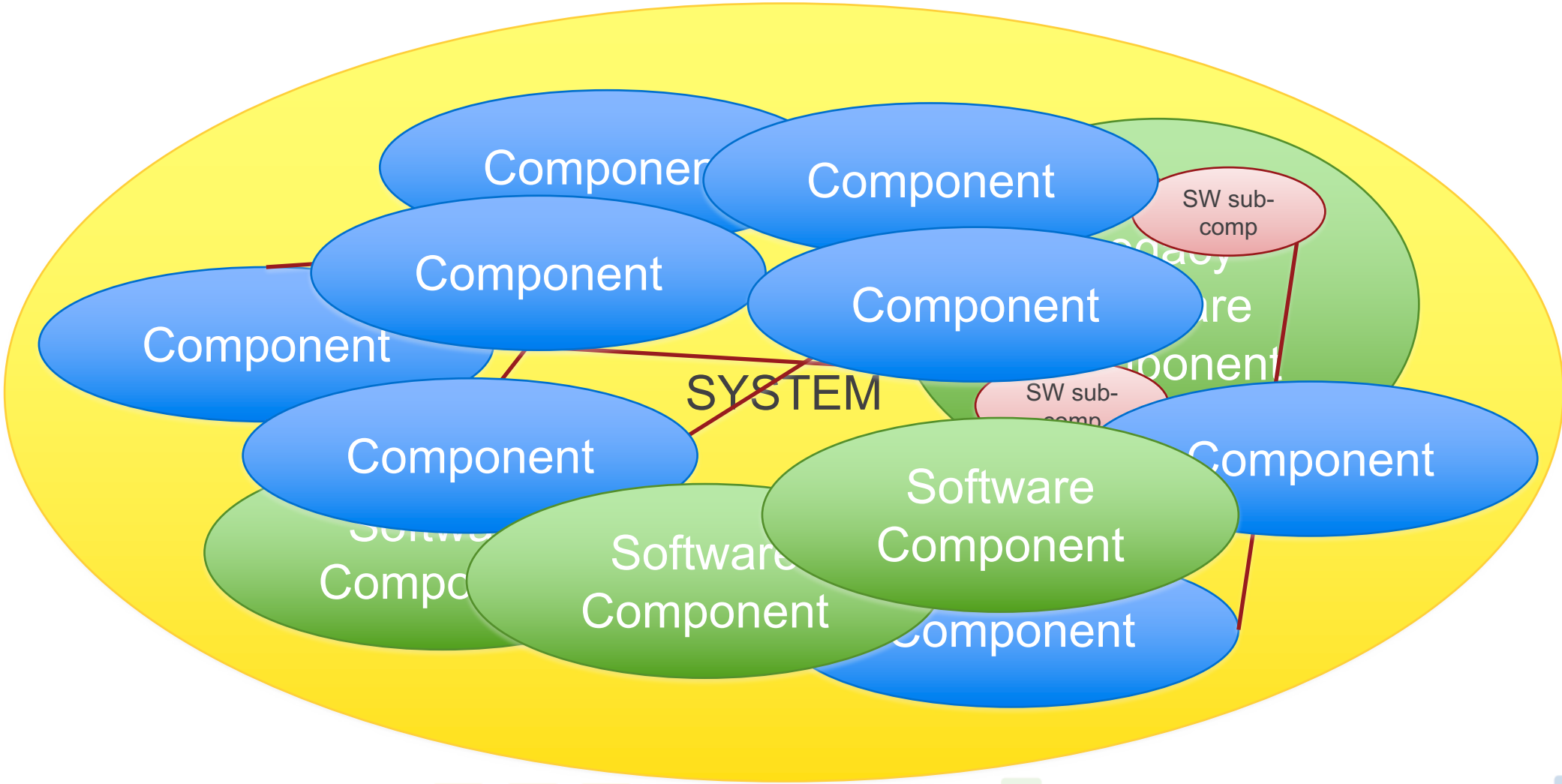
Legacy Software

Legacy Software

- **Poor quality!**
- **A legacy software** is a software where is not possible to understand all the fundamental concepts that shaped it as they could be neither available nor existent for understanding (Pressman, 2005).

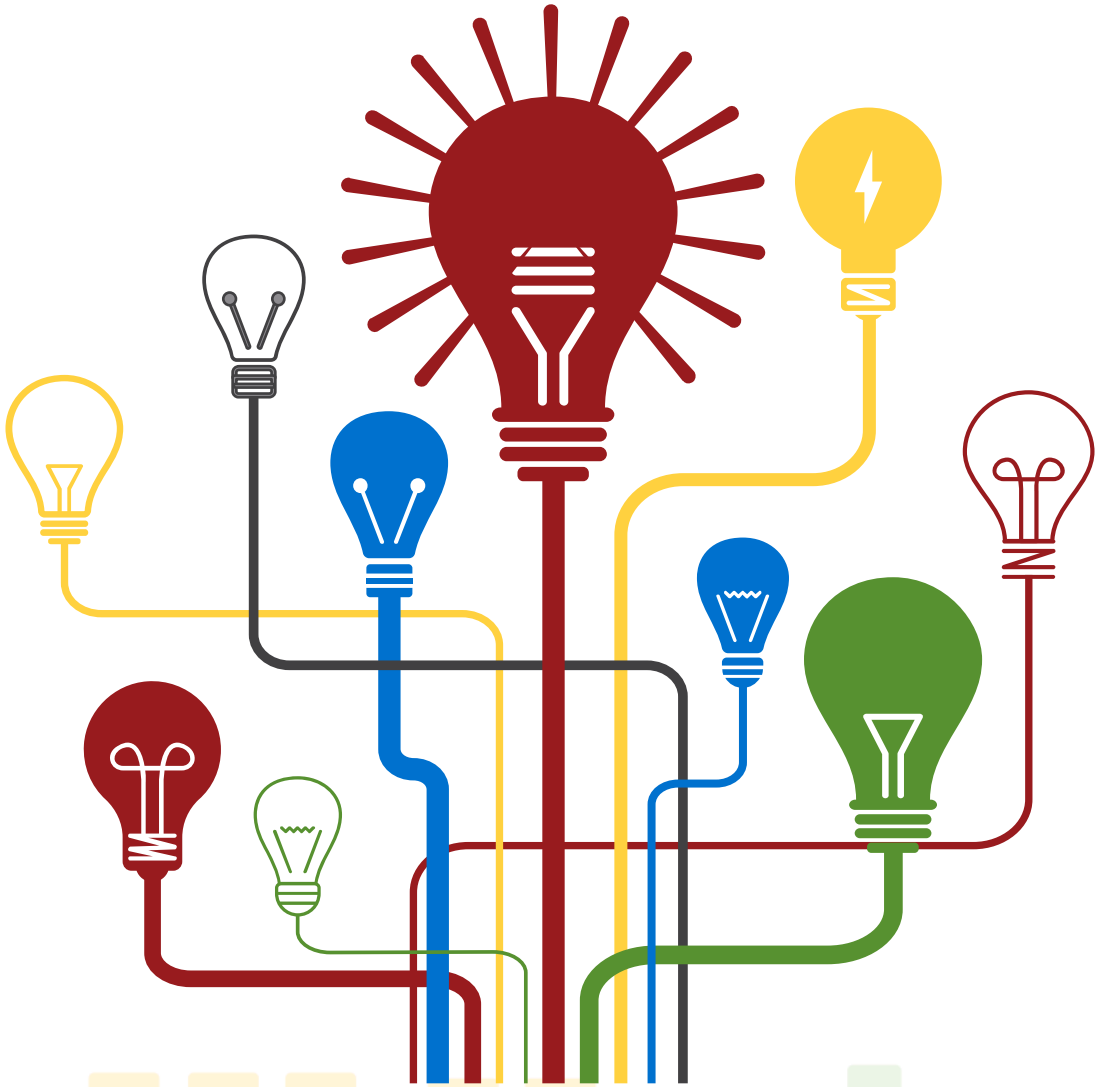
Background

Legacy Software as System component



Legacy Software as System component

What are the important parts of legacy software for the fit with the system context?



How to align legacy software maintenance and development for the fit with the entire system development?

Legacy Software as System component

Software as a component posits new challenges in software reverse engineering.

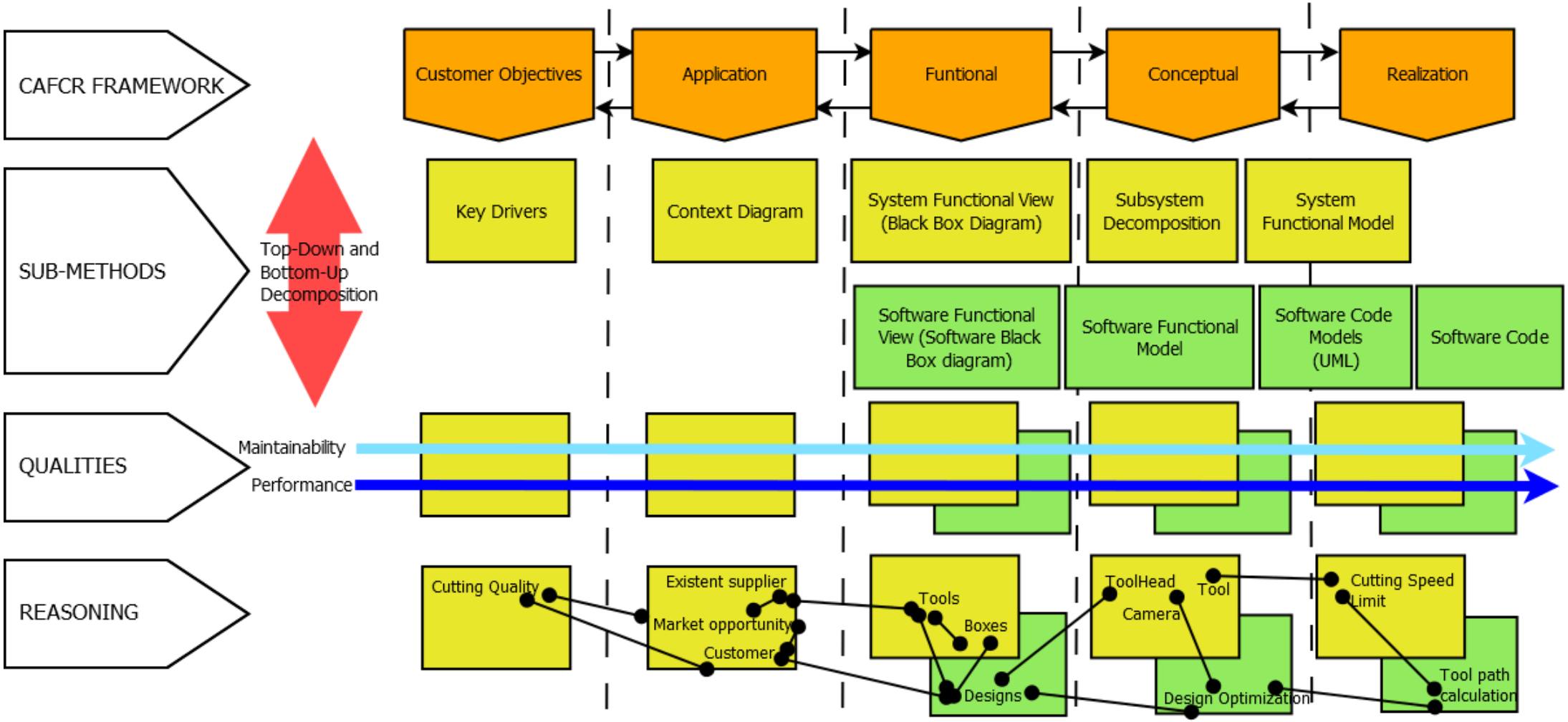
Existing techniques of software reverse engineering, are covering it mostly for the single discipline of software engineering

To resolve the fit between a legacy software component and a complex system is to reverse engineer the reasoning in how the software component was developed according to the overall system needs for satisfying the stakeholder's requirements

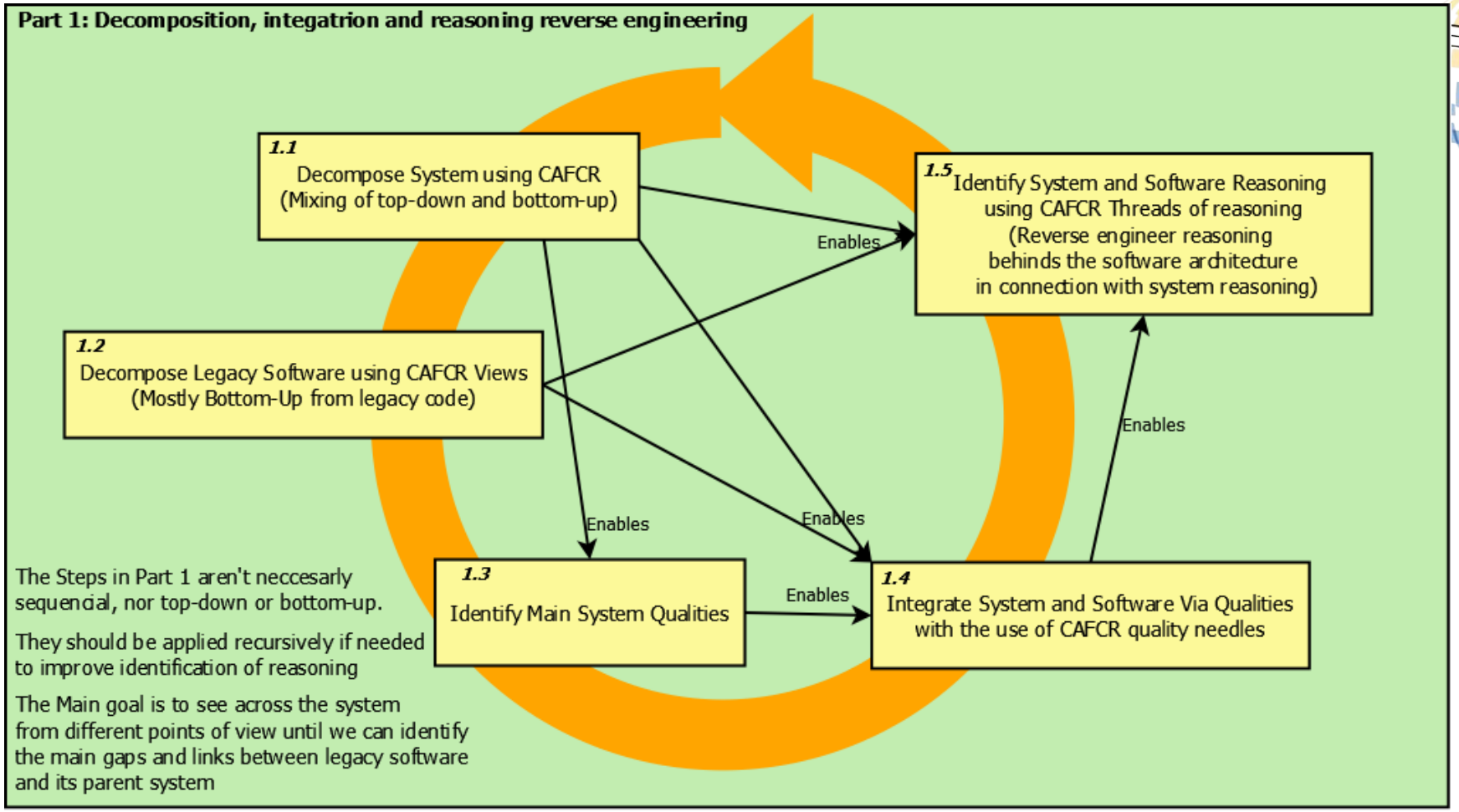
Systems engineering and its associated systems thinking are known to have the working methods for resolving complex systems' issues

For modeling and decomposing system and software, ISO/IEC/IEEE 42010 (ISO/IEC/IEEE, 2011) is an international standard for the description of systems and software engineering architecture methods.

CAFCR and ISO/IEC/IEEE 42010



Solution

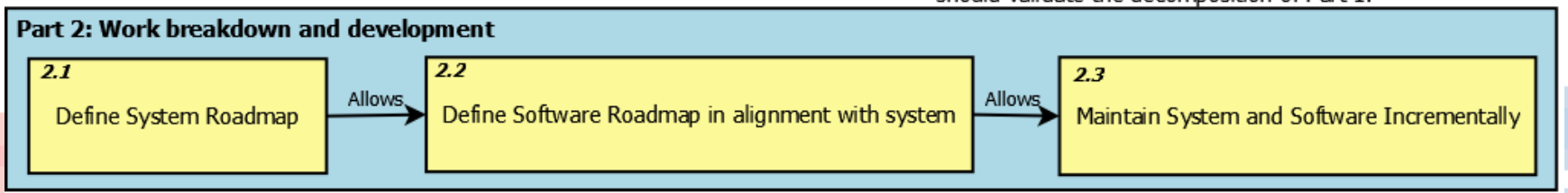


Part 1 will allow the generation of a work breakdown in Part 2.
Part 2 aligns the maintenance of the legacy software with the system needs over time.

enables

validates

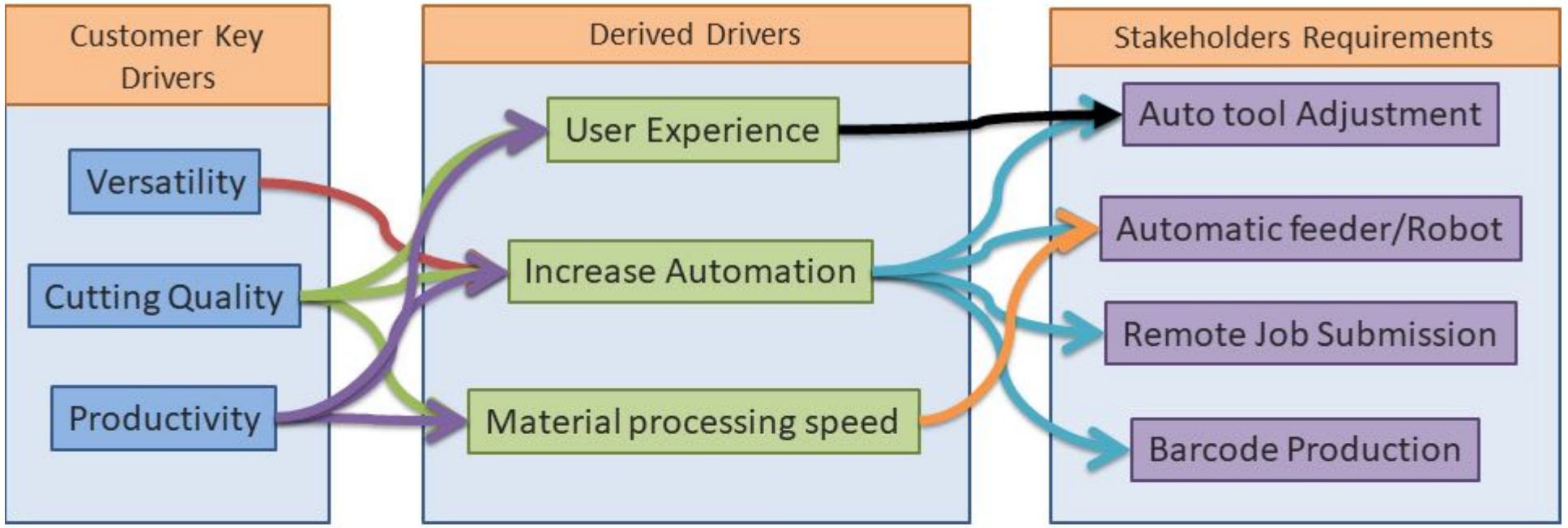
Part 1 represents snapshopts of the system life-cycle,
The needs of the system may change over time due to countless factors(stakeholder needs, market, regualtions, etc), therefore work breakdowns may have to be updated.
Part 1 will need to be applied several times recursively for keeping up with these changes.
The results from the system and software evolution in Part 2 should validate the decomposition of Part 1.



moraga.max@gmail.com
yangyang.zhao@usn.no

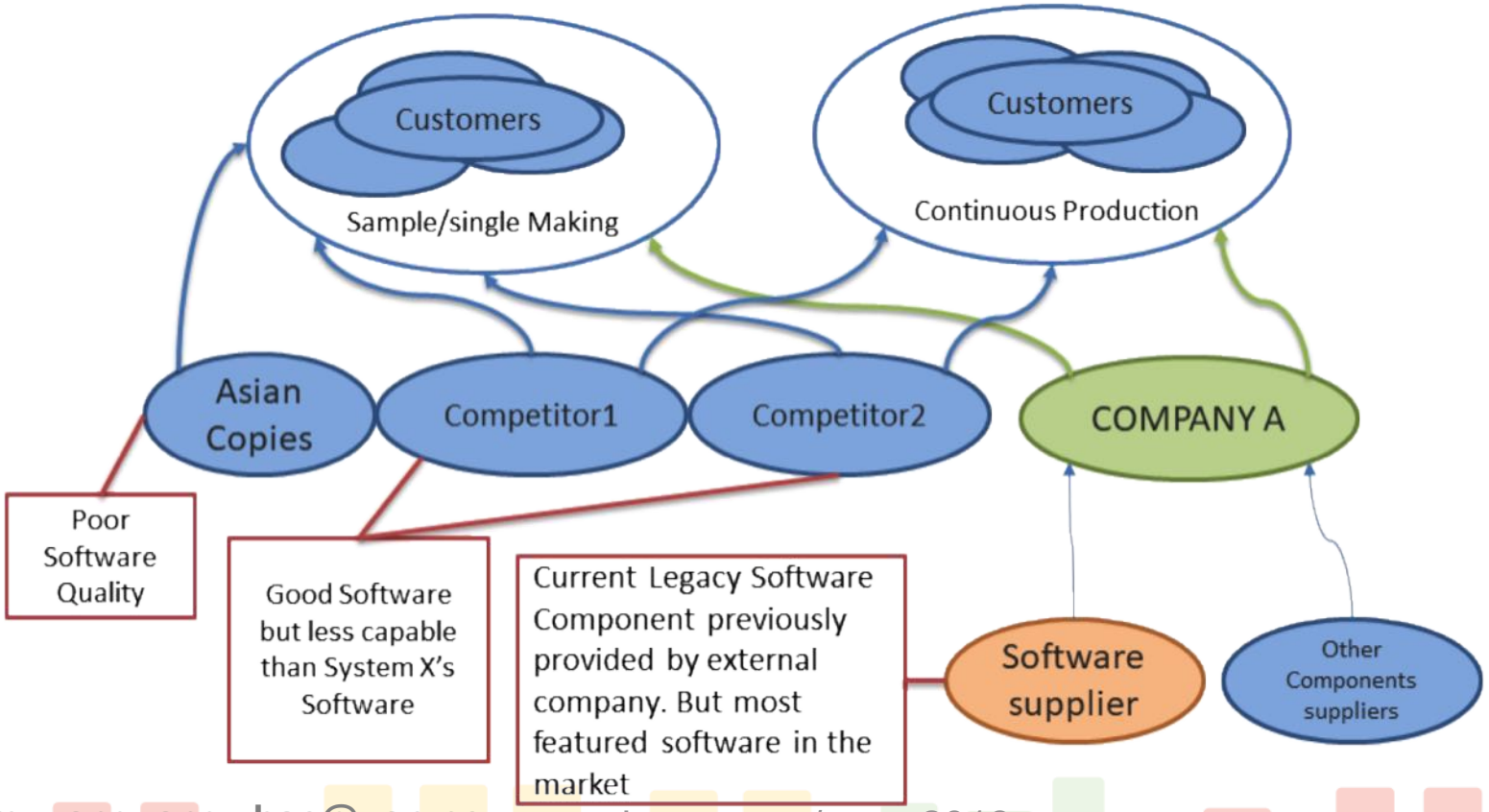
Case Analysis - Activity 1.1: System decomposition

- Cafcr – Customer Objectives



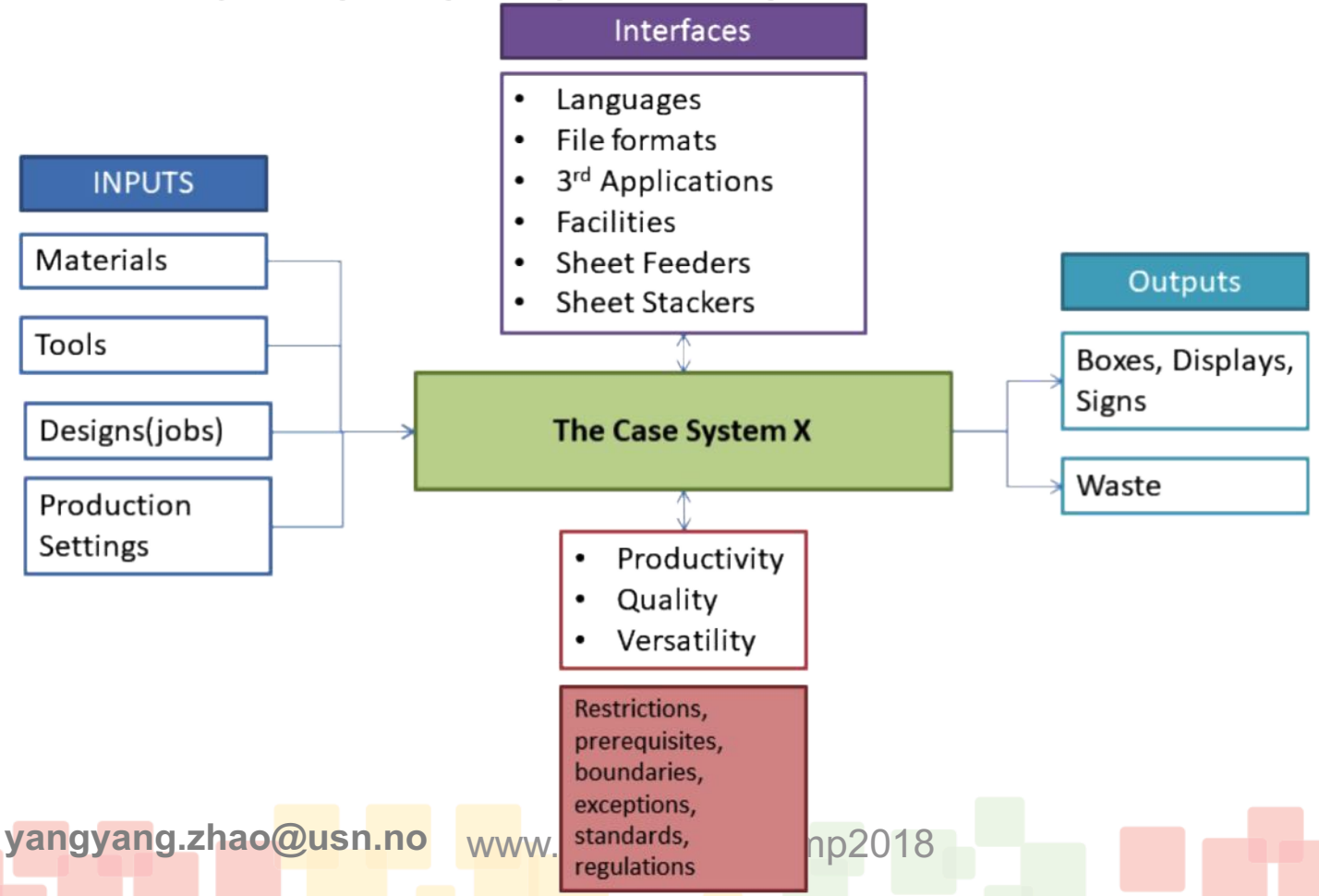
Case Analysis - Activity 1.1: System decomposition

- cAfcr – Application View



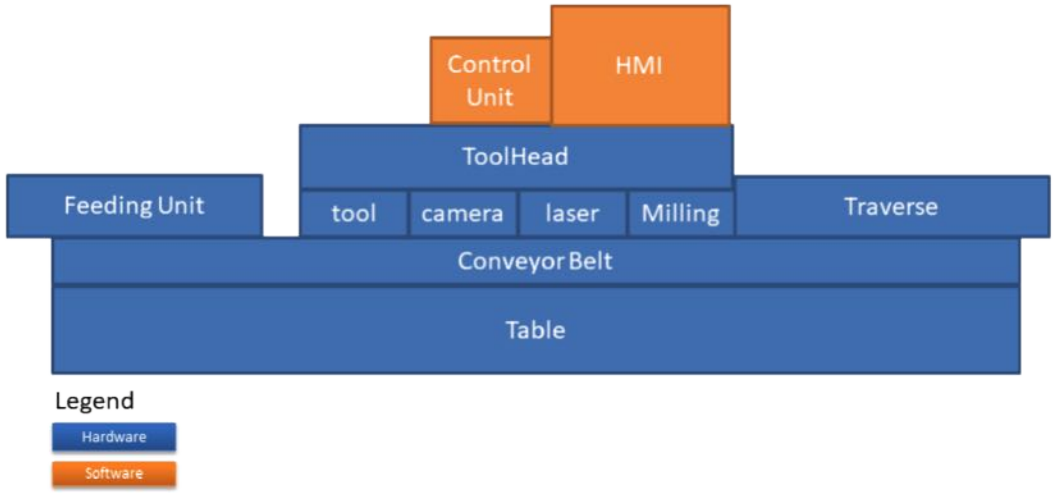
Case Analysis - Activity 1.1: System decomposition

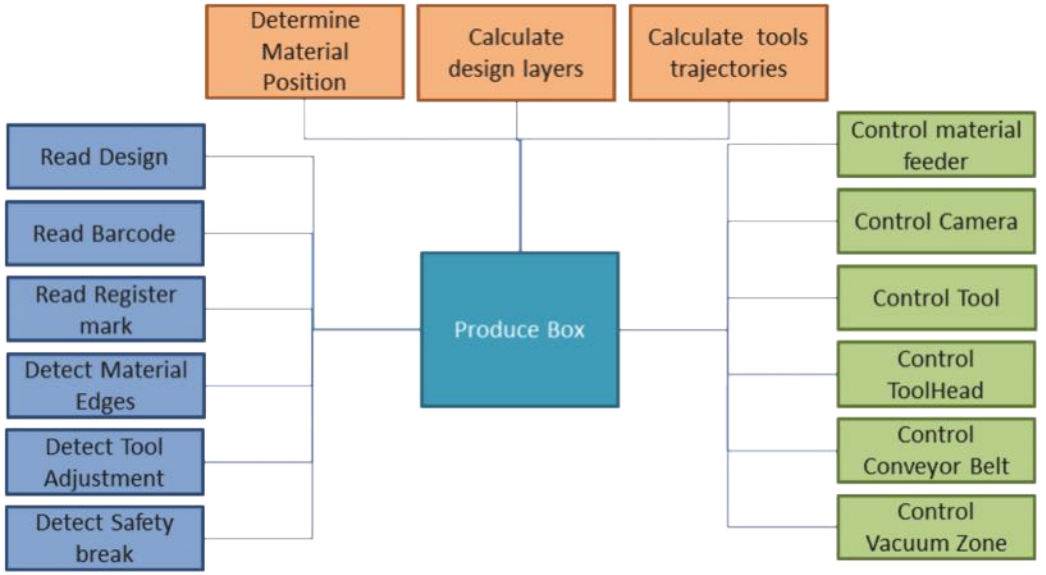
- caFcr – Functional View



Case Analysis - Activity 1.1: System decomposition

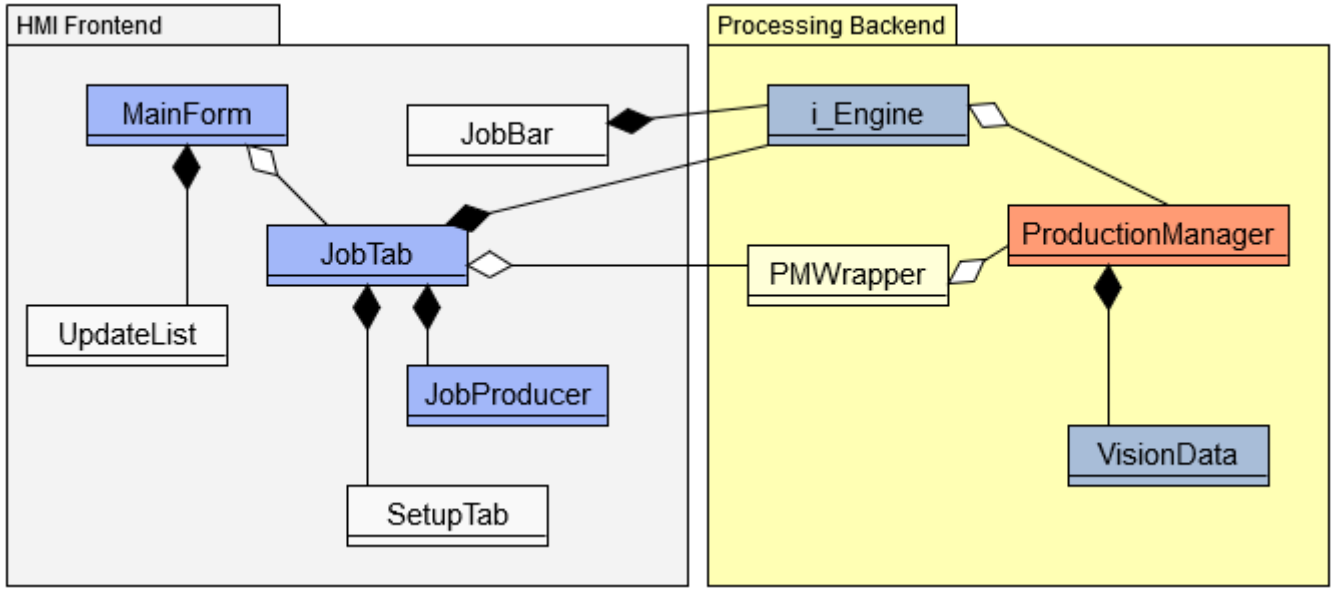
- cafCr – Conceptual View





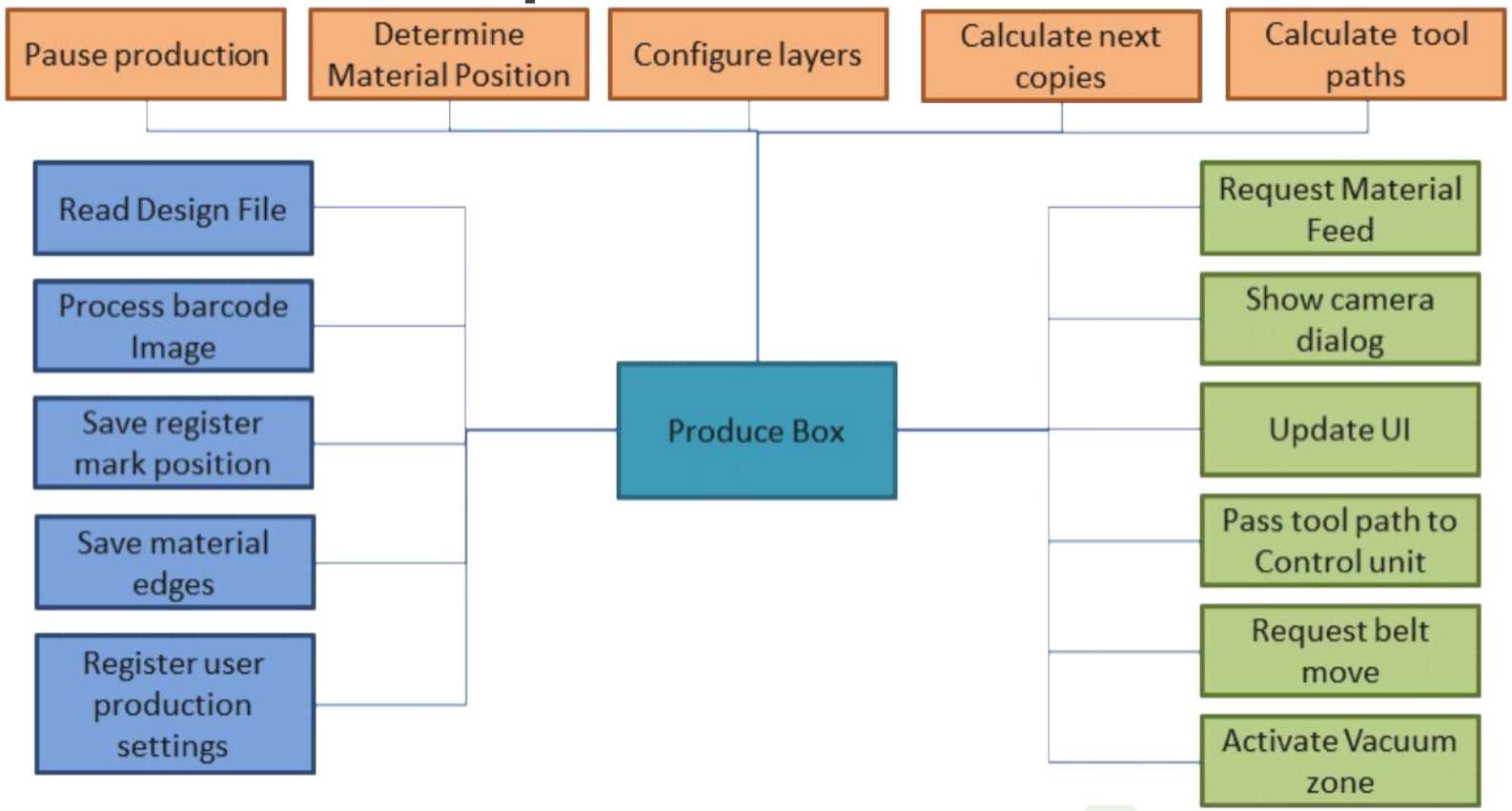
Case Analysis - Activity 1.2: Software Decomposition

- cafcR – Realization View

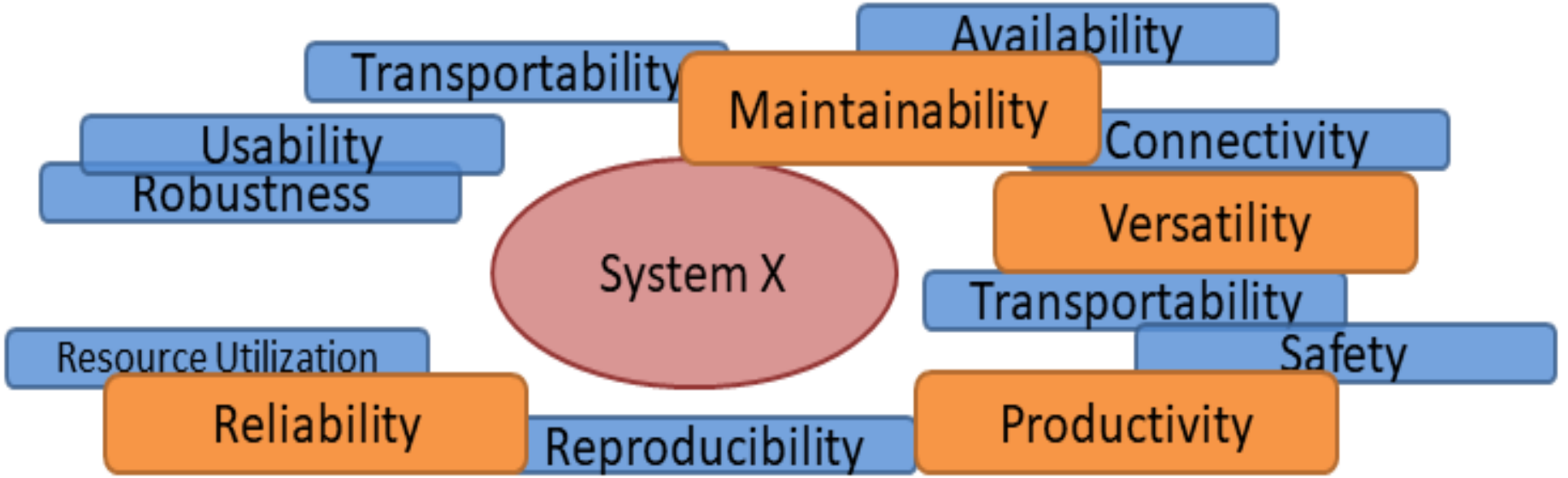


Case Analysis - Activity 1.2: Software Decomposition

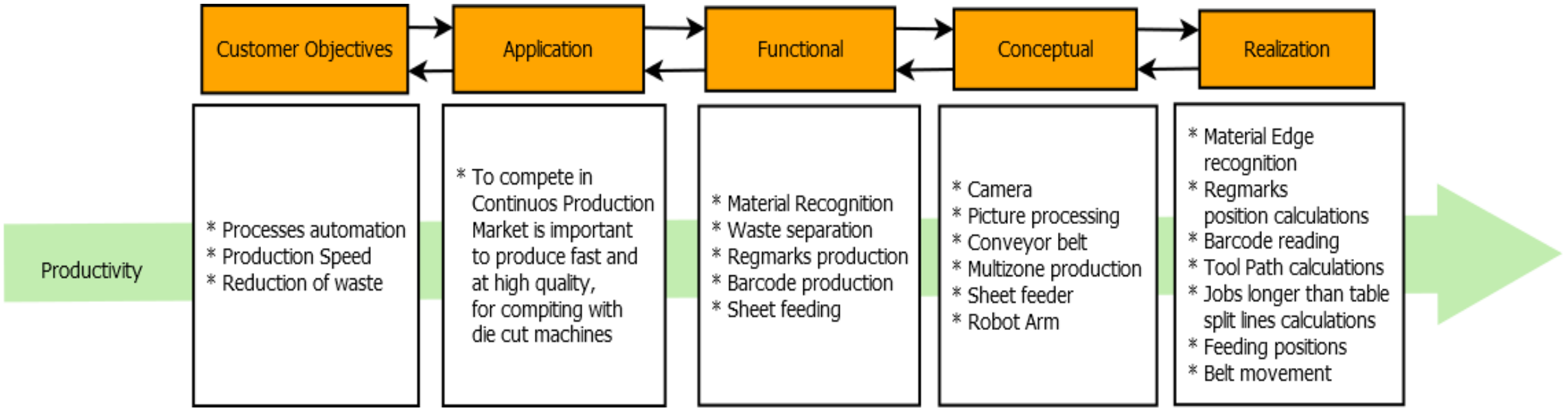
- cafCr – Conceptual View



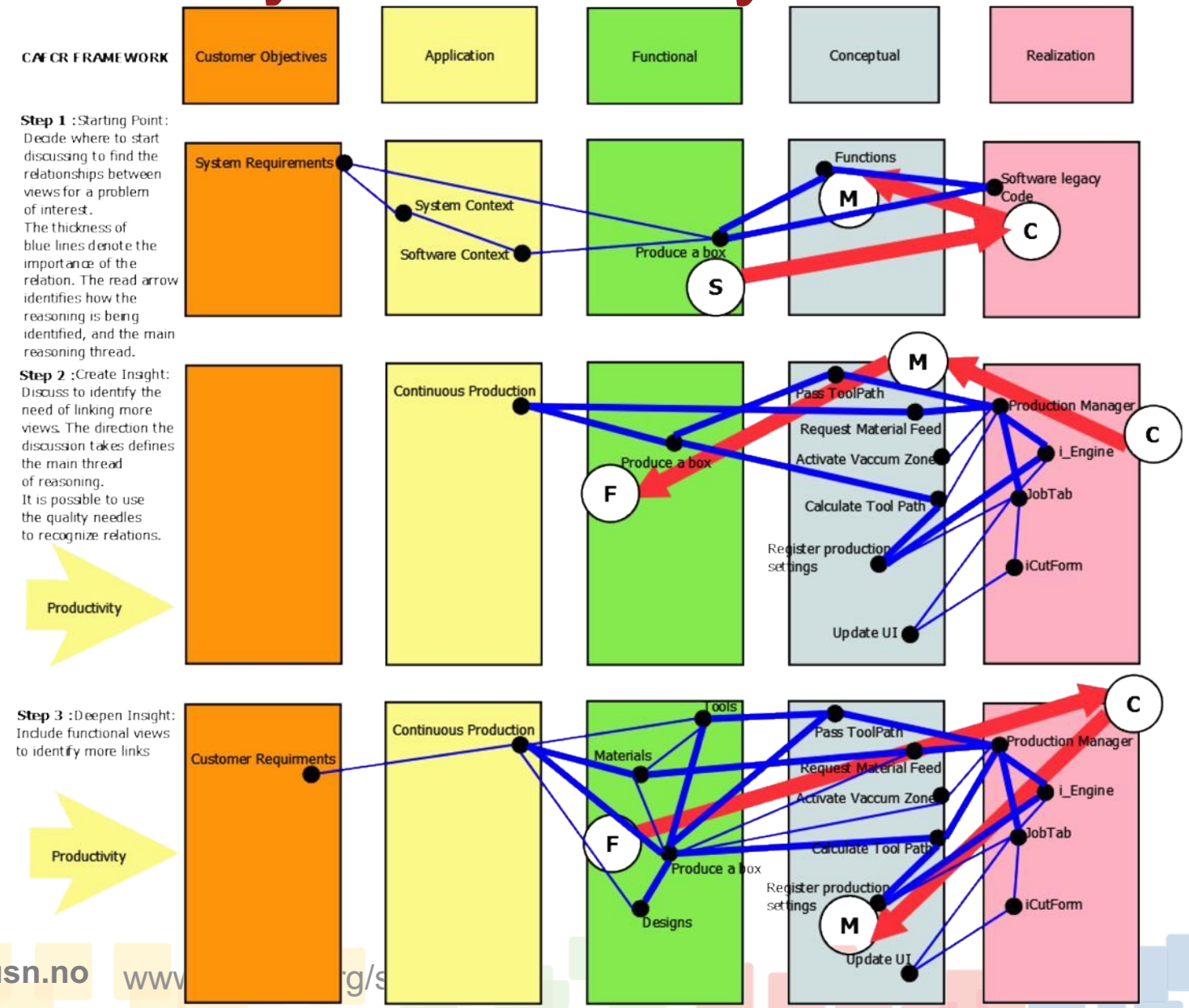
Case Analysis - Activity 1.3: Identify Main Qualities



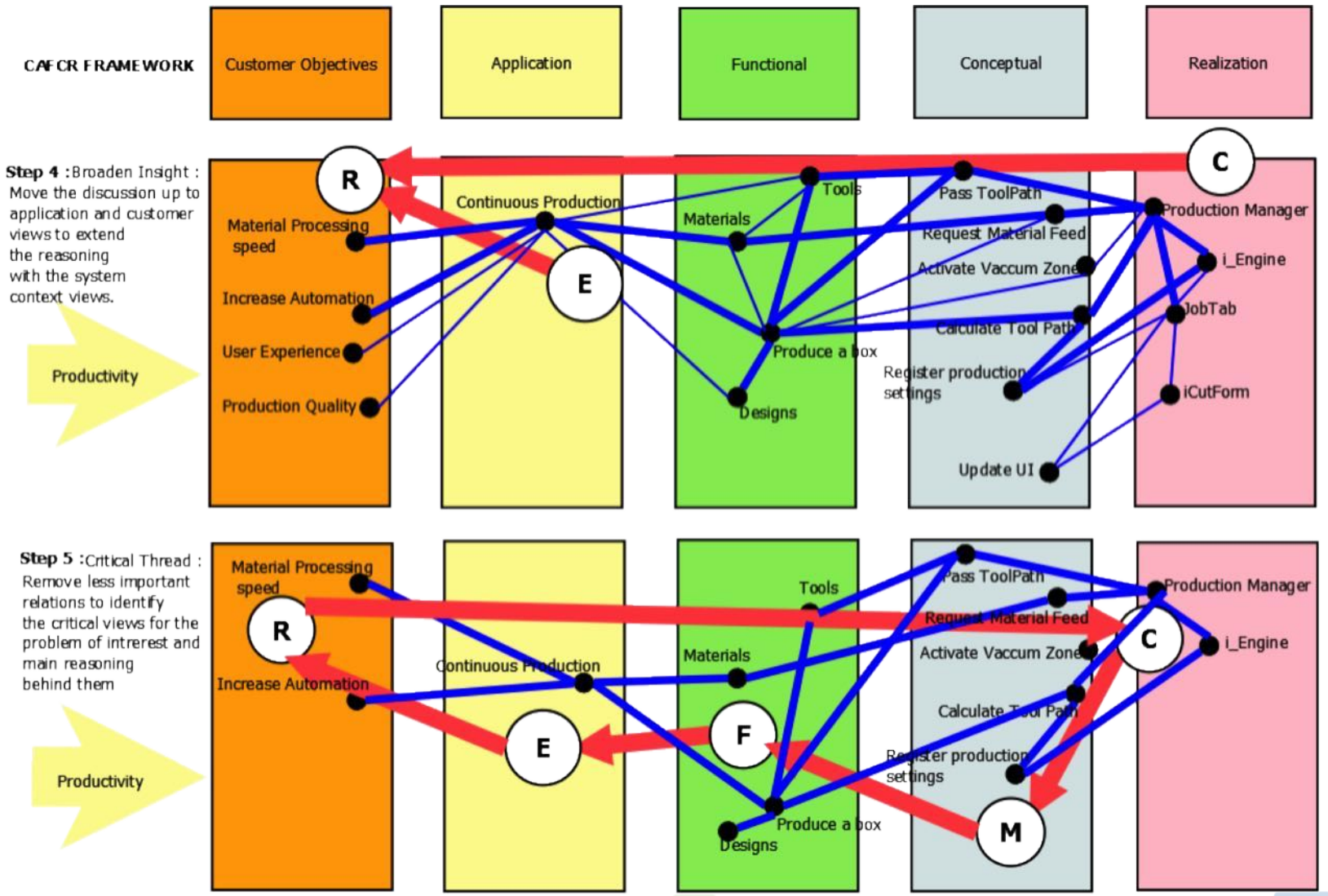
Case Analysis - Activity 1.4: Integrate via Qualities



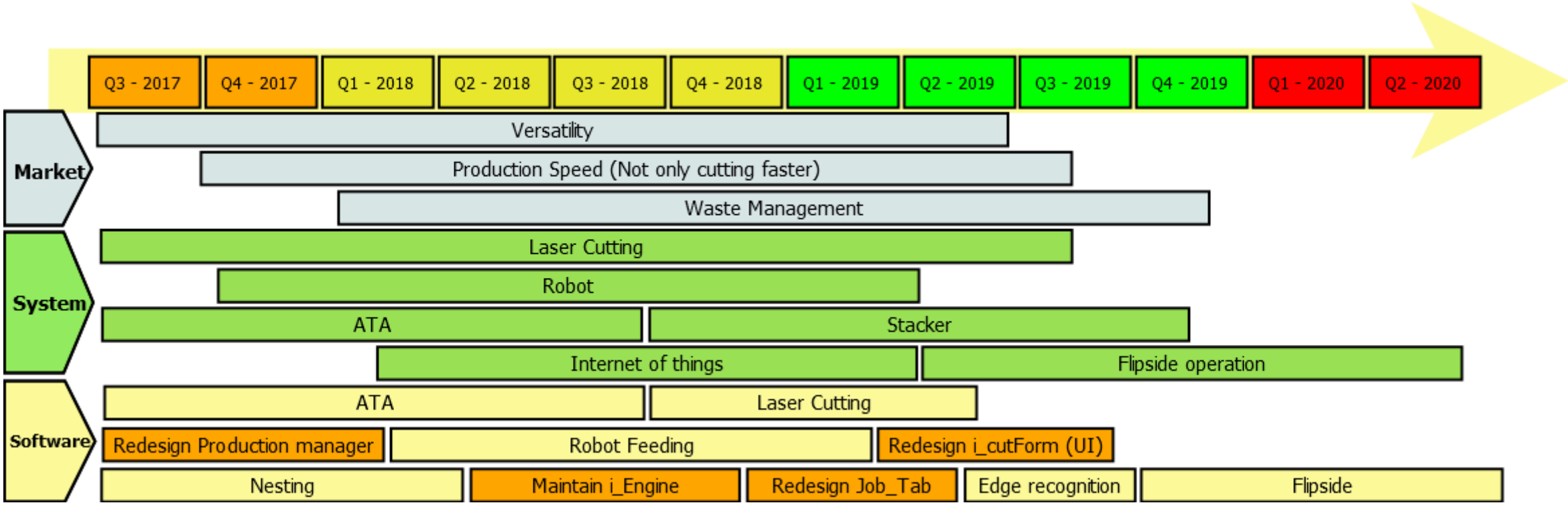
Case Analysis - Activity 1.5: Identify Threads of Reasoning



Case Analysis - Activity 1.5: Identify Threads of Reasoning



Case Analysis - Activity 2.1-2.2: Roadmaps



Results and Verification

- The case study verifies the application of the conceptual solution with an industry problem and illustrates the usage details.
- Based on the expert assessment, it is found over an 80% of the participants agrees or strongly agrees the solution can help the company to solve the problem in the continuous development of the legacy software.

Q&A

Contacts: moraga.max@gmail.com
yangyang.zhao@usn.no

28th Annual INCOSE international symposium

Washington, DC, USA

July 7 - 12, 2018

www.incose.org/symp2018