Software Quality Engineering:
To Tame An Explosion

Inżynieria Jakości Oprogramowania:
Okiełznać Eksplozję

Witold Suryn
École de technologie supérieure
Montreal, Canada

Agenda

- **Why Software Quality Engineering?**
- **What is Software Quality Engineering**
- **Feature vs quality. Who wins?**
- **Quality and maturity**
- **Software Quality Models**
- **Software Quality Life Cycle**
  - Quality Requirements
  - Quality Design
  - Quality Implementation
  - Quality Evaluation
  - Traceability
- **Software Trustworthiness**

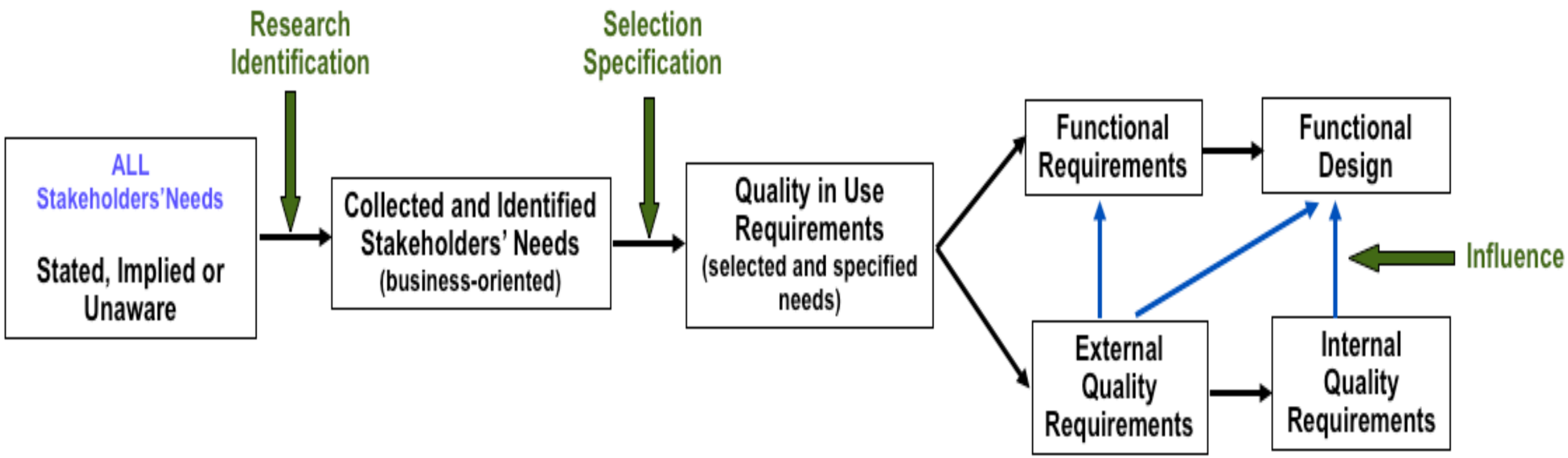
Why Software Quality Engineering ?

- **Why software?**
  - Because in contemporary social life software, systems and services rendered by them are omnipresent, beginning with watches we wear, ending with nuclear electricity plants or spaceships.
- **Why quality?**
  - Because if the above “instances” of software work without required quality we may be late, dead or lost in space.
- **Why engineering?**
  - As in every technical domain, it is the engineering that transforms ideas into products, *it is the verified and validated set of “to-dos”* that help develop the product that not only has required functionalities but also executes them correctly

Five perspectives of quality

- The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality, it is "something toward which we strive as an ideal, but may never implement completely";
- The user perspective is concerned with the appropriateness of the product for a given context of use;
- The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO 9001 or models, like the Capability Maturity Model (CMMI, 2002);
- The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product;
- The final perspective of quality is value-based. This perspective recognizes that the different perspectives of quality may have a different importance, or value, to various stakeholders.

A User and Quality

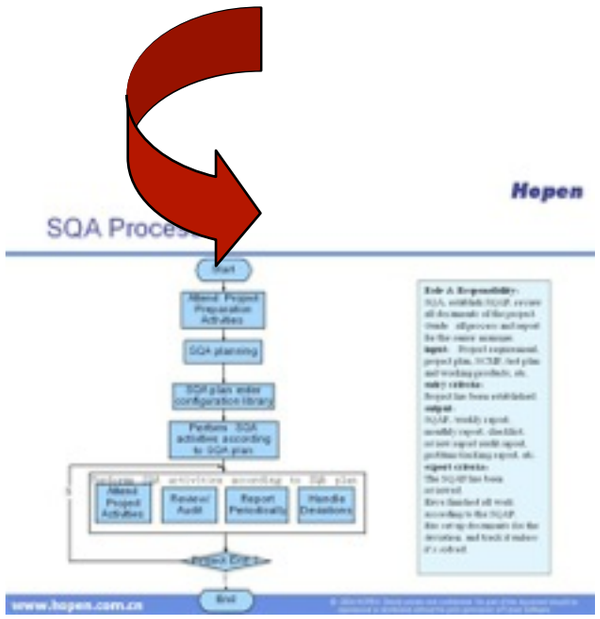


What is Software Quality Engineering?

Is Software Quality Engineering

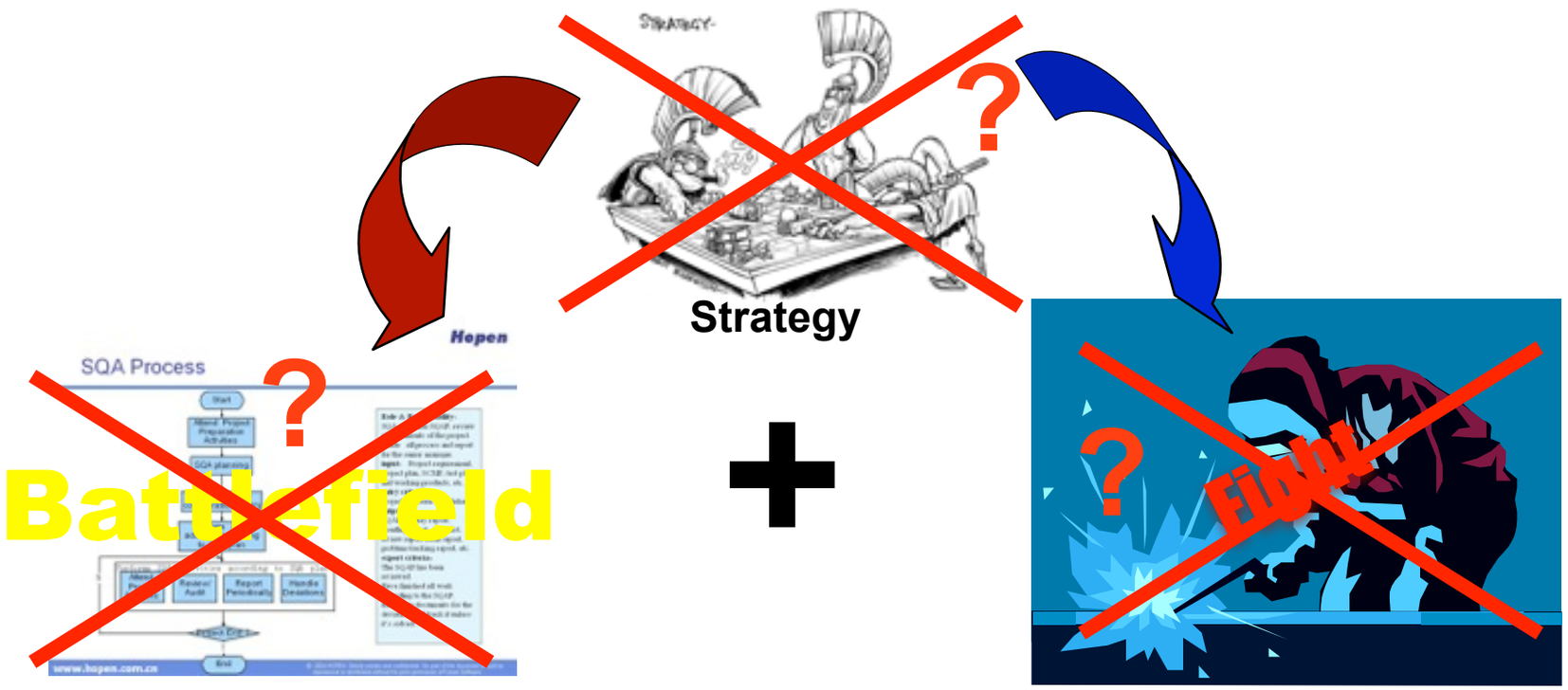
just another term for

Software Quality Assurance?



Battlefield

What is SUCCESSFUL Software Quality Engineering?



Software Quality Engineering DEFINITION

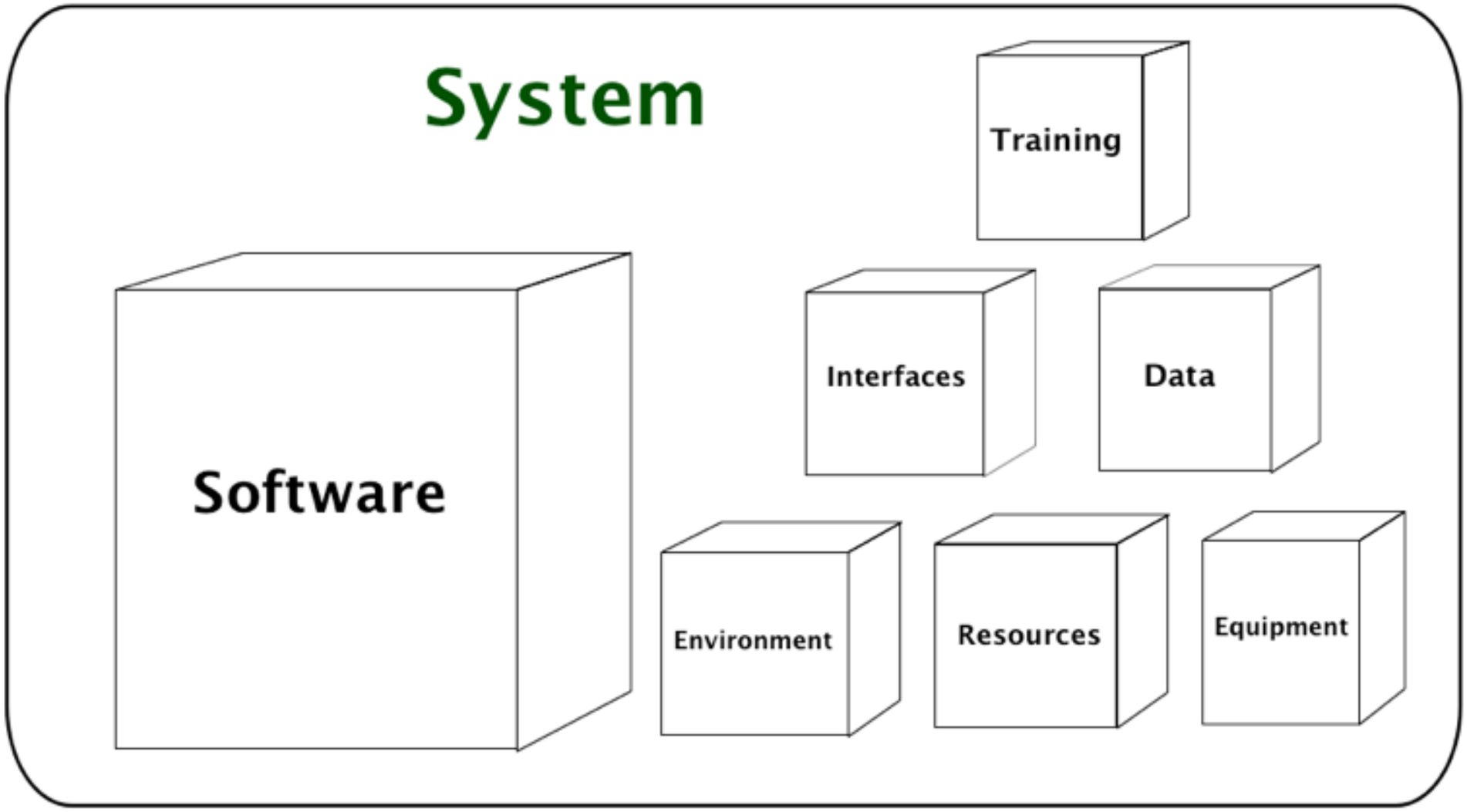
1. The application of a continuous, systematic, disciplined, quantifiable approach to the development and maintenance of quality throughout the *whole life cycle* of software products and systems; that is, the application of quality engineering to software,
2. The study of approaches as in (1).

Software Quality Engineering OBJECTS (1)

- Software is
  - (1) instructions (computer programs) that when executed provide desired function and performance,
  - (2) data structures that enable the programs to adequately manipulate information, and
  - (3) documents that describe the operations and use of the programs.

Software Quality Engineering OBJECTS (2)

- System is
  - A collection of functionally arranged software components, related artifacts, resources and services that are organized to accomplish a predefined goal by processing information.



Software Quality Engineering OBJECTS (3)

- IT service
  - Cloud Computing:
    - IaaS (**HW** Infrastructure as a Service)
    - PaaS (**OS+programming environment** Platform as a Service)
    - SaaS (**Complete suite** Software as a Service)
    - NaaS (Network as a Service)
  - Service-oriented architecture (SOA)

Feature-quality relationship (1)

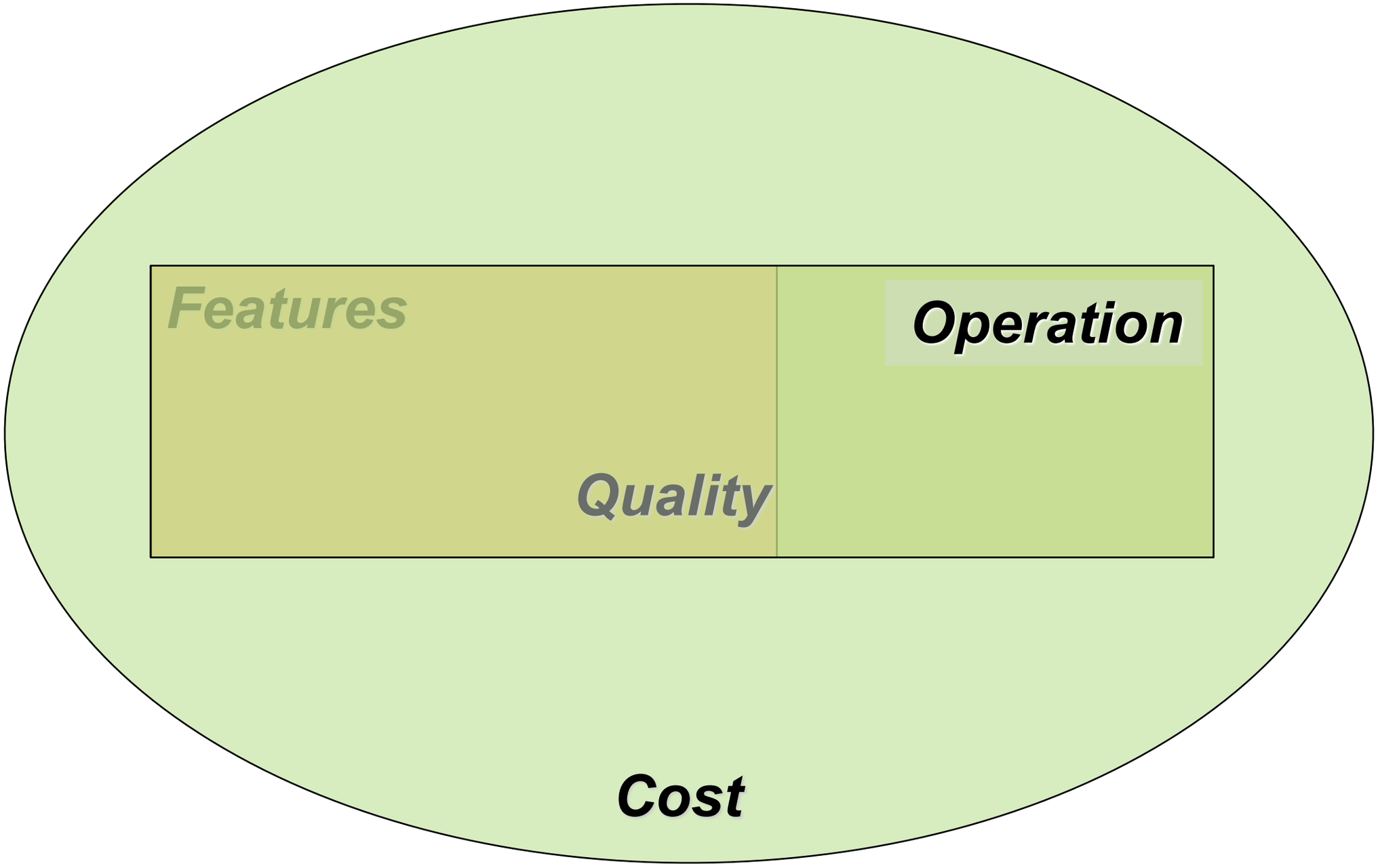
- **Everything in software engineering boils down to user's satisfaction**
- **Satisfaction is conditional to the overall behaviour of the system, with software product in the first place**
- **The behaviour of any software product is perceived through features (see [1]) and quality**
- **Features and quality of software product are expressed through requirements**
- **Any behaviour-related requirement for software product may only be realized through code**

<u>Conclusion</u>: functional, non-functional (feature) and quality requirements are inter-related

<u>Hypothesis</u>: the relationship between quality requirements and functional and non-functional (feature) requirements is bi-directional

[1] A feature "is a set of logically related functional requirements that provides a capability to the user and enables the satisfaction of a business requirement" [Wiegers, 1999]

Feature-Quality
Ideal Inter-relationships

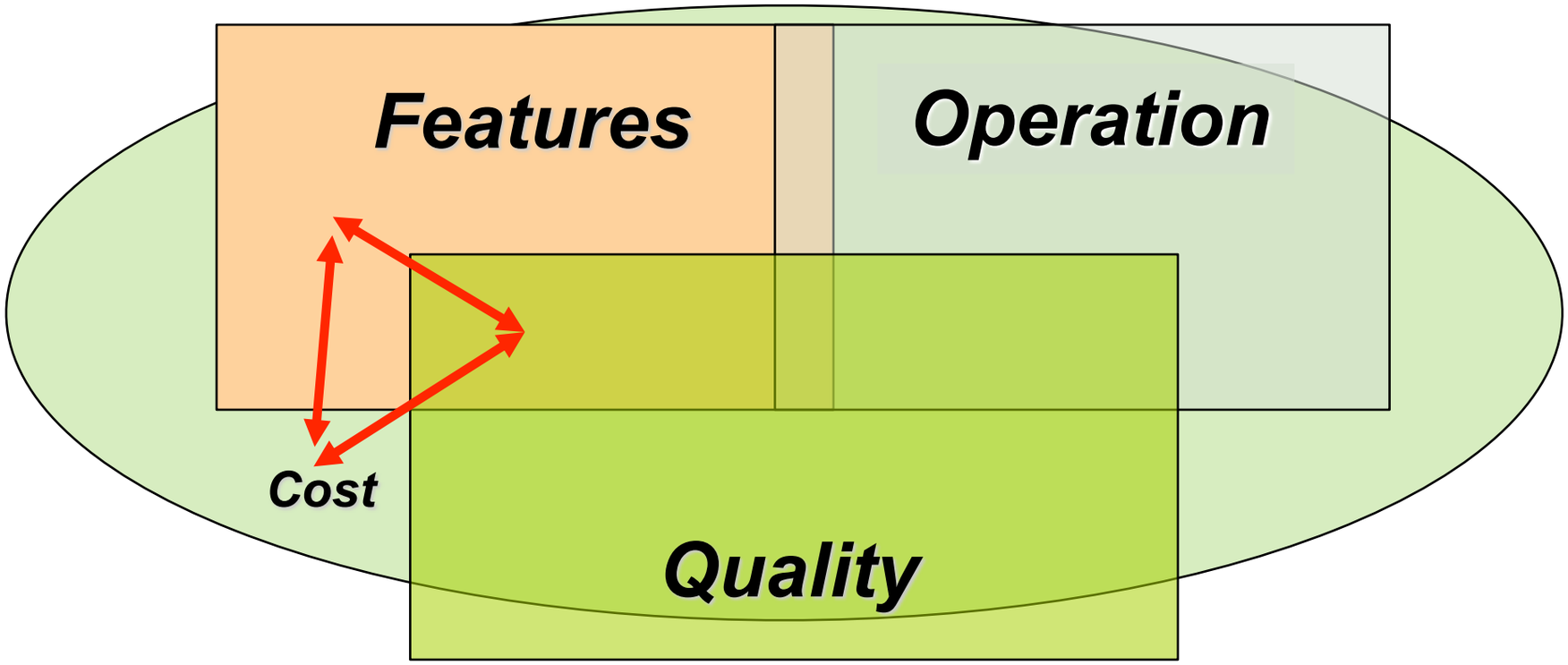


Stakeholder's Concerns

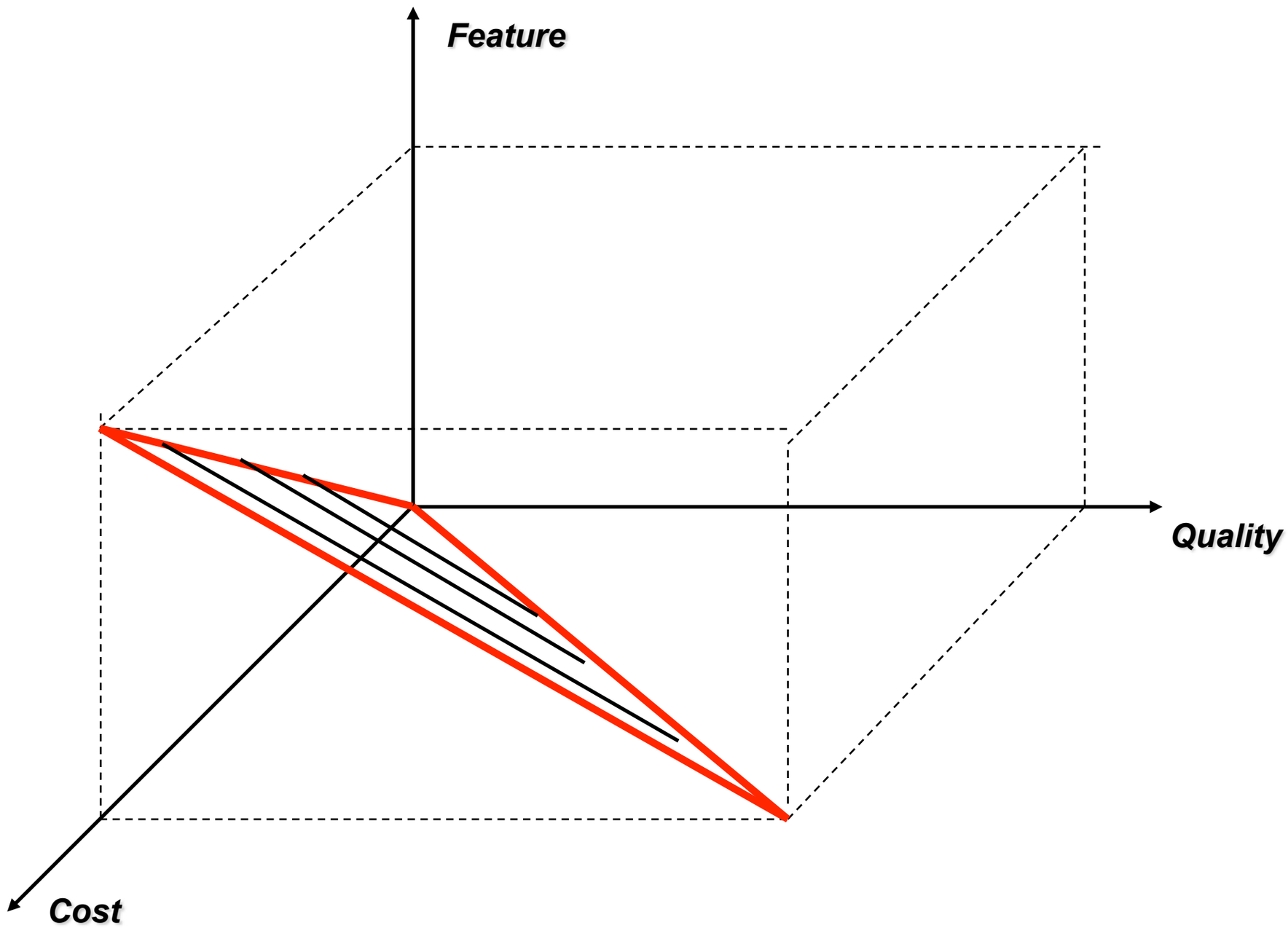
- **Four dimensions of software product evaluation and appreciation**
  - *Features*: service contents. A static view on what the software offers to the user
  - *Operation*: service execution. A dynamic view that covers both non-functional and maintenance aspects
  - *Quality*: a static and dynamic view of software *fitness* to do its job (see [2])
  - *Cost*: a static and dynamic view of necessary investments

[2] Quality is the delivery of a product that satisfies user expectations of functionality, performance, and security at previously agreed upon costs and timeliness. Global Tester, 2003

Stakeholder's Concerns Real Inter-relationships



Feature-Quality-Cost (FQC) Relationship



Feature-Quality-Cost Relationship

- **Economic perspective**

$$\text{Cost} = a_0 f_0 + a \sum \text{features} + b \sum \text{quality aspects}$$

or

$$C = a_0 f_0 + af + bq$$

Where:

a, b – proportions of investment

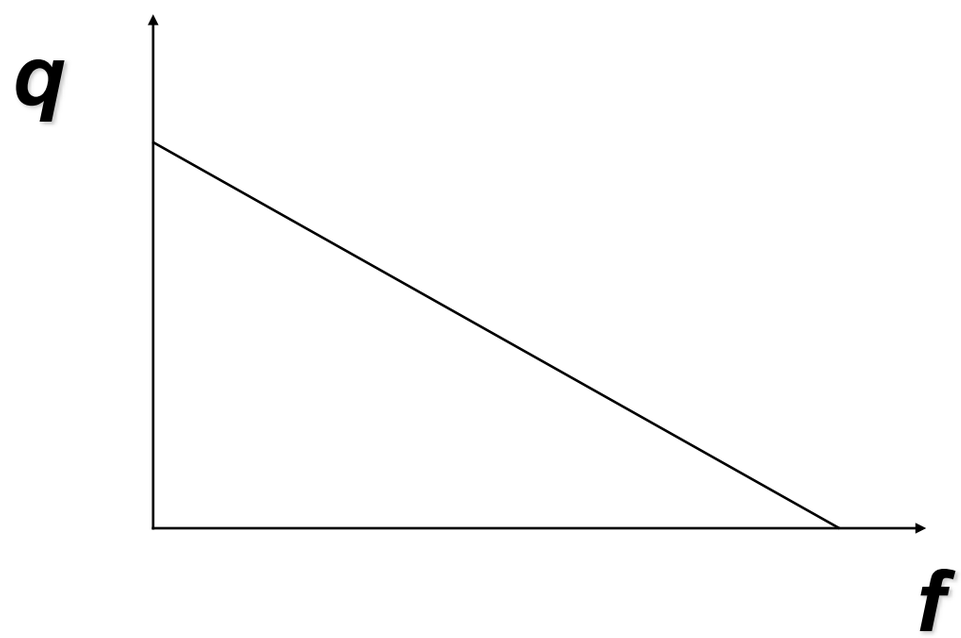
a_0 - initial investment level

f_0 - initial set of features

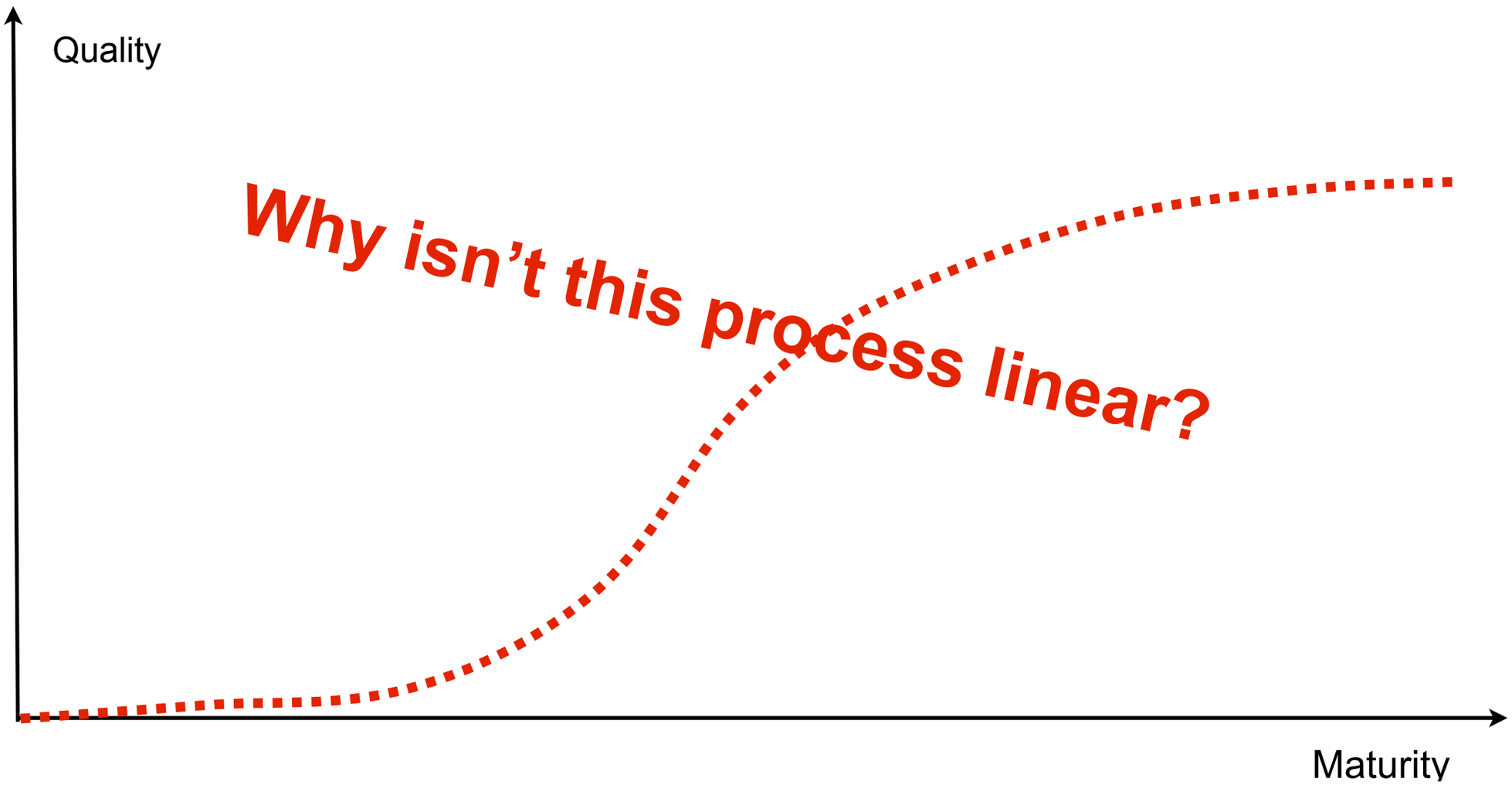
Feature-Quality Financial Relationship

For any given Cost = const

$$q = (const - a_0 f_0 - af)/b$$



Quality vs Maturity

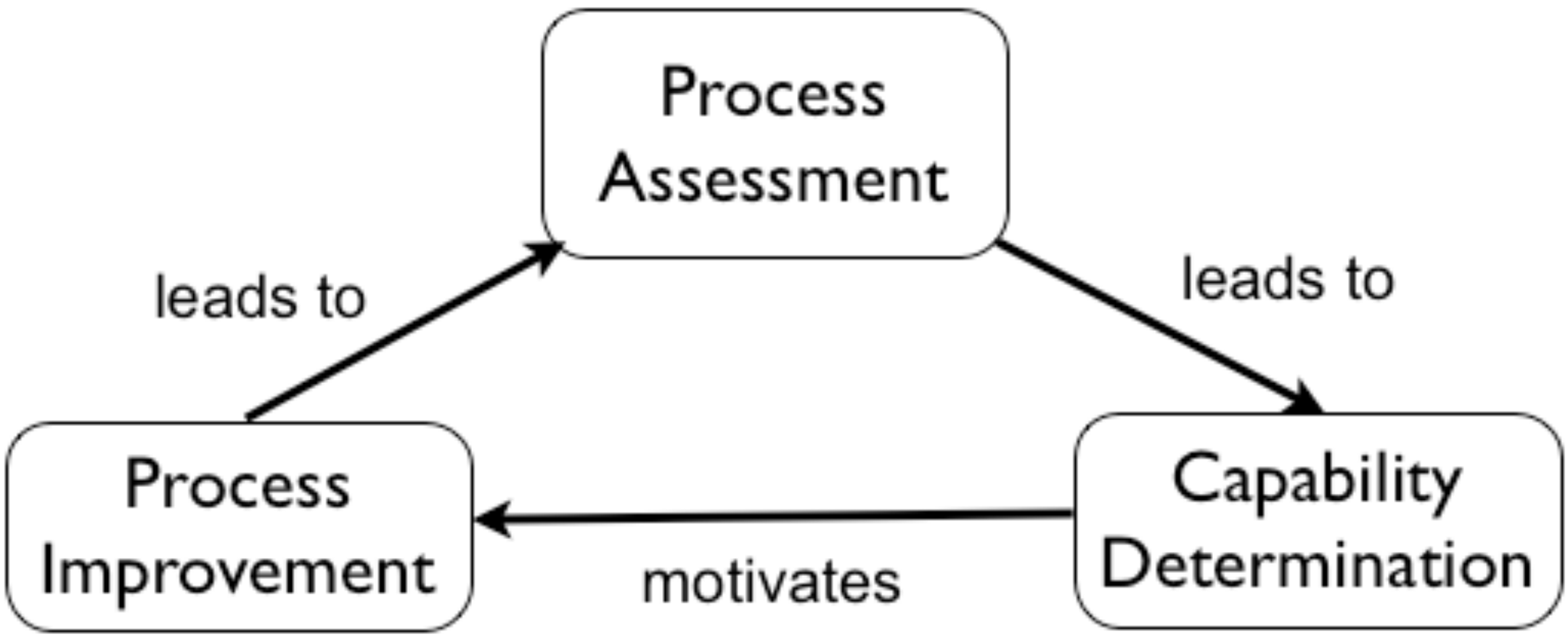


Quality vs Maturity CMMI

	Level	Capability	Result
5 Optim-izing	Continuous Process Improvement	Organizational Innovation & Deployment Causal Analysis & Resolution	Productivity & Quality
4 Quanti-tatively Managed	Quantitative Management	Quantitative Process Management Software Quality Management	
3 Defined	Process Standardization	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Product Management Risk Management Integrated Teaming Integrated Supplier Management Decision Analysis & Resolution Organizational Environment for Integration	
2 Managed	Basic Project Management	Requirements Management Project Planning Project Monitoring & Control Supplier Agreement Management Measurement & Analysis Product & Process Quality Assurance Configuration Management	
1 Initial	Heroic Efforts	Design Develop Integrate Test	Risk & Waste

Quality vs Maturity
SPICE

- SPICE (Software Process Improvement and Capability dEtermination) is an international initiative to support the development of an International Standard for Software Process Assessment



Process Assessment Relationship
(adapted from ISO/IEC 15504-1)

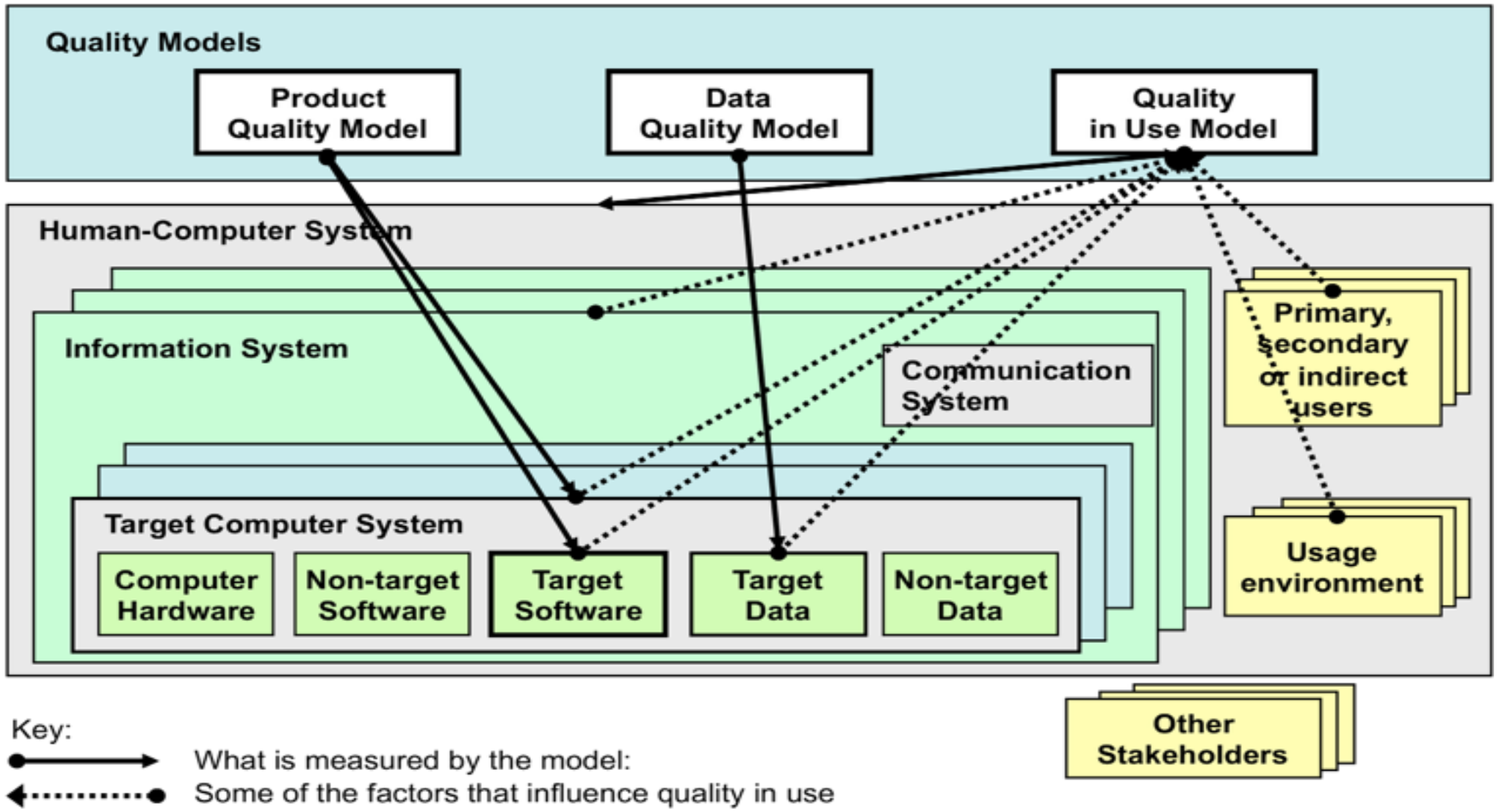
Quality vs Maturity
SPICE (cntd)

- **SPICE** provides a structured approach for the assessment of processes by or on behalf of an organization for the following purposes:
  - understanding the state of its own processes for process improvement;
  - determining the suitability of its own processes for a particular requirement or class of requirements;
  - determining the suitability of another organization's processes for a particular contract or class of contracts.

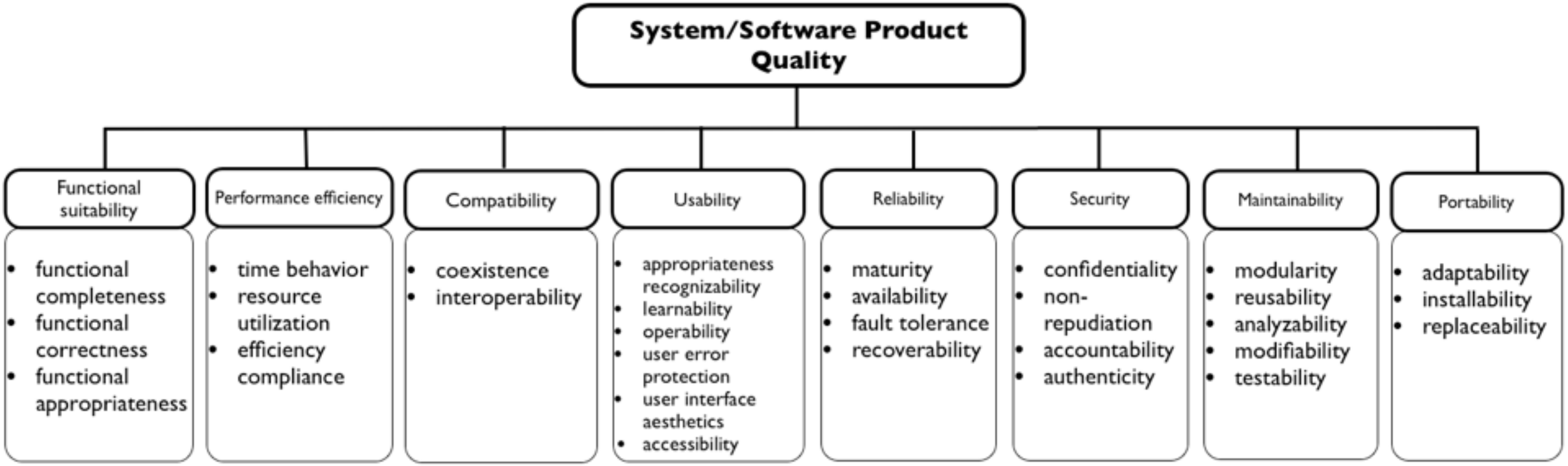
Software Quality Model(s)

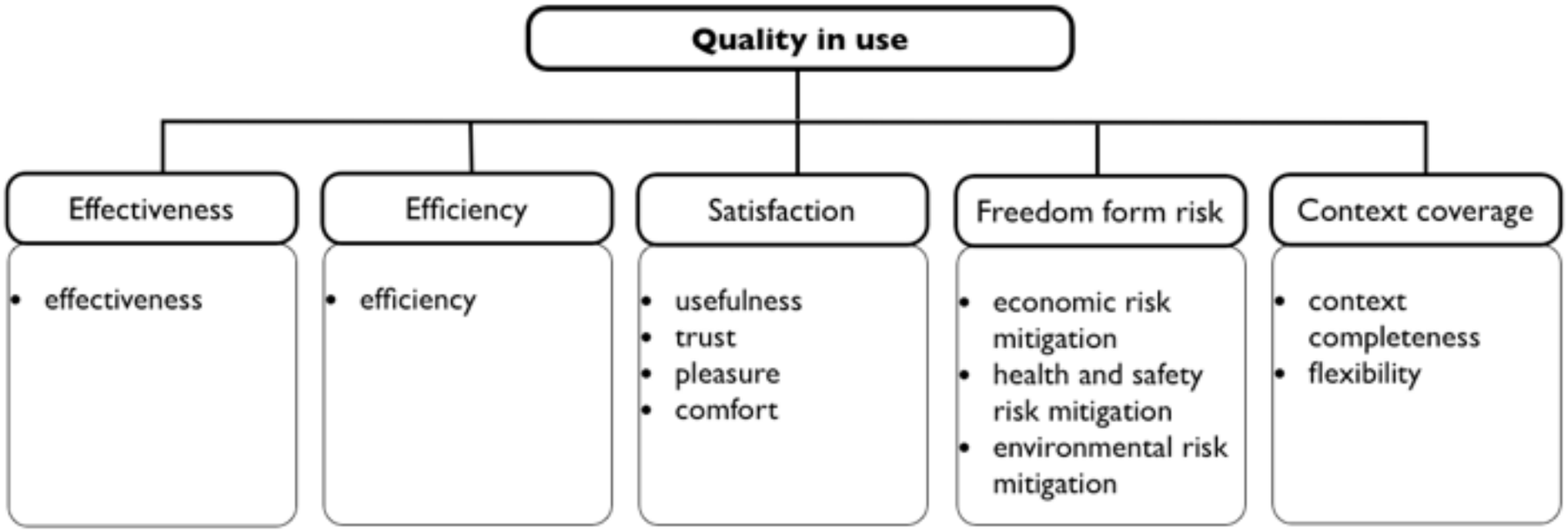
- **Quality models present an approach to tie together different quality attributes with basic objectives to:**
  - help understand how the several facets of quality contribute to the whole,
  - emphasize clearly that software quality is much more than simply faults and failures,
  - help to navigate through the map of quality characteristics, sub-characteristics and appropriate measures (measurement formulas and scales)
  - help to define our evaluation profile (what precisely we want to evaluate)
- **Best known models: McCall, Boehm, Dromey, ISO 9126, ISO 25010**
- **Practically used: ISO 9126 (declining) and ISO 25010 (gaining popularity)**

Software Quality Model(s) Targets

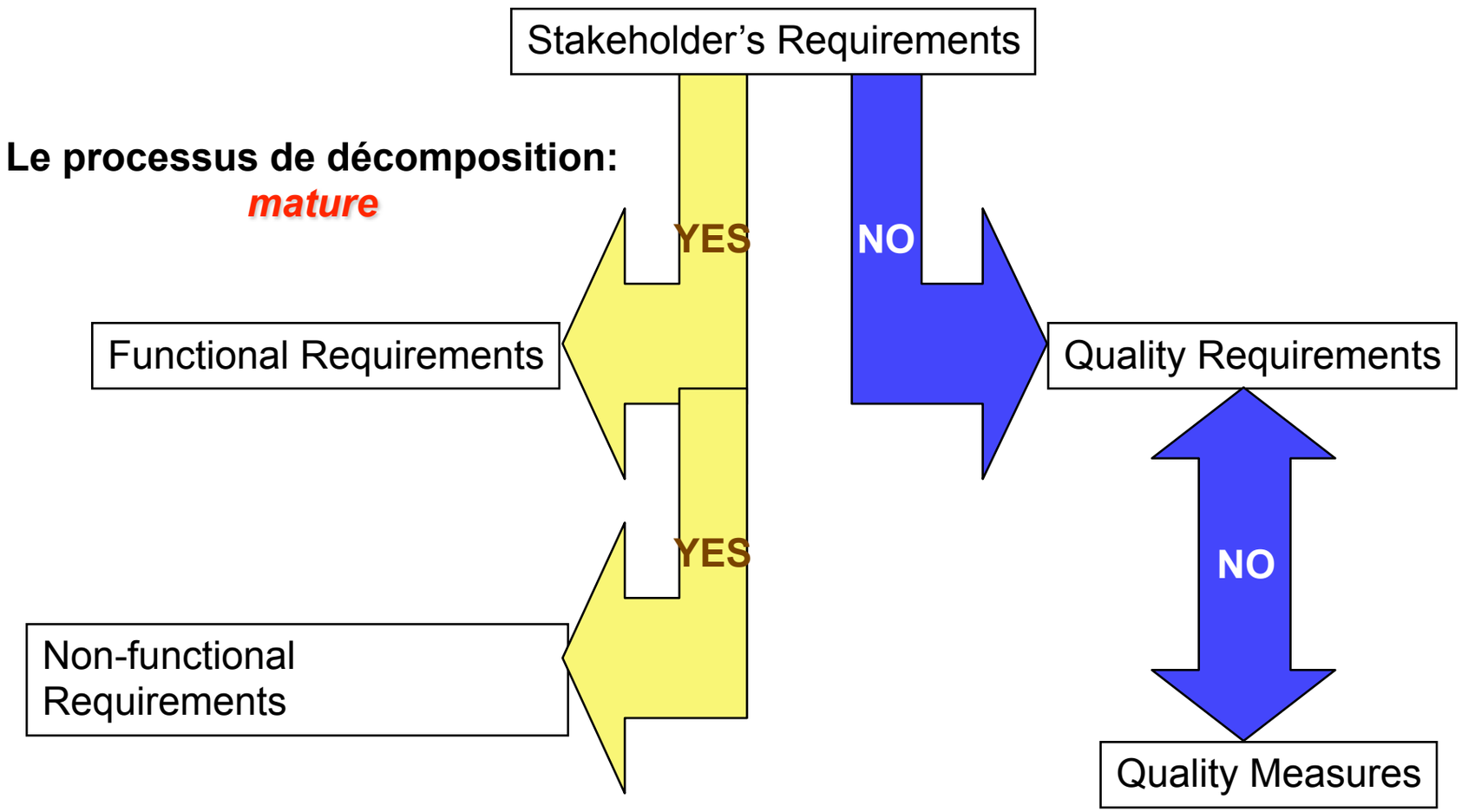


Software Quality Model(s) ISO 25010

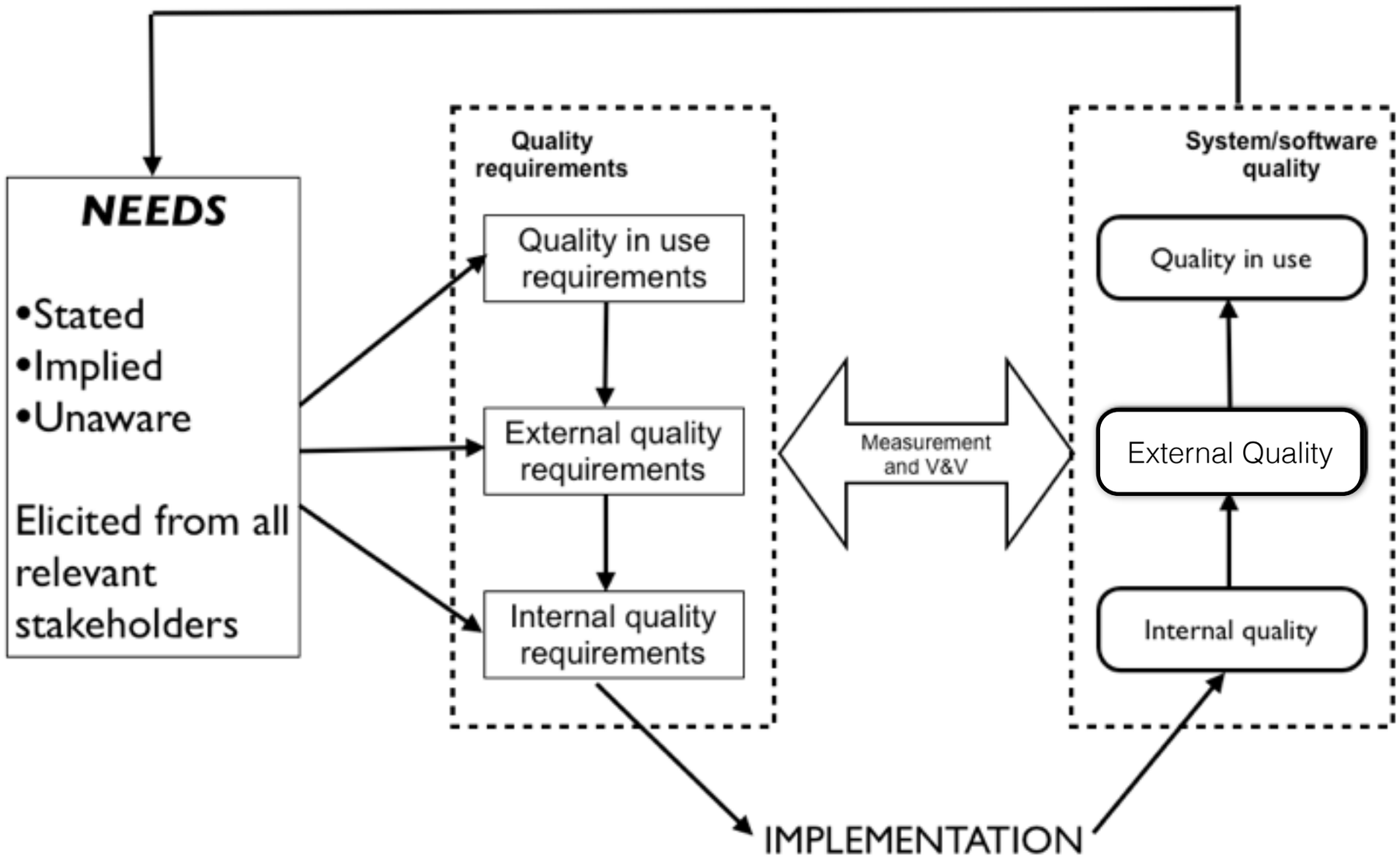




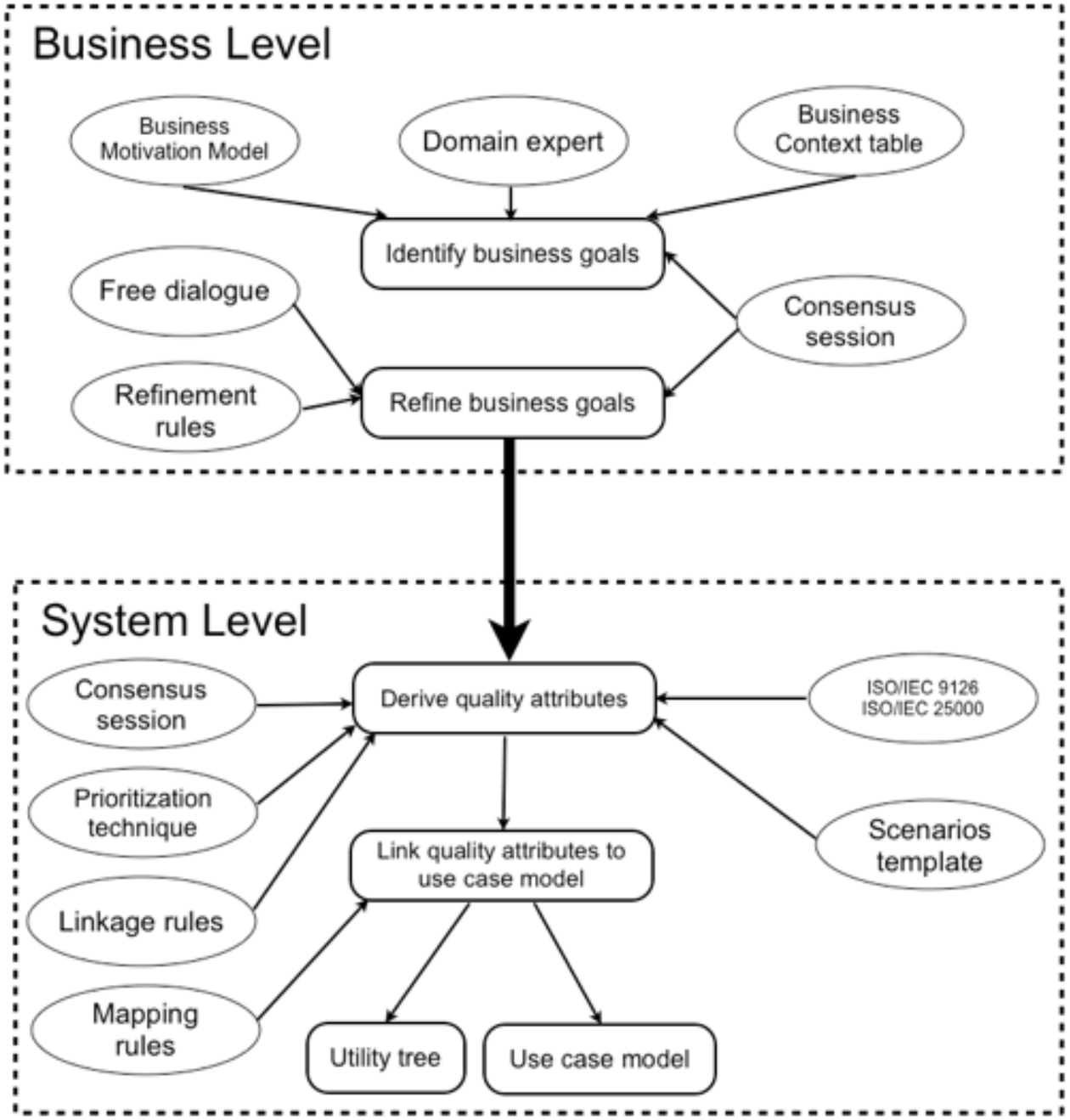
Software Requirements vs Software Quality Requirements



Software Quality Requirements



Software Quality Requirements SOQUAREM



Software Quality Design

- *Quality design* is the phase in which quality engineer translates quality requirements gathered earlier into their technical representations (or in simpler form: executable to-dos) applicable in next phases

- In this sense the design of quality means the identifying, specifying and indicating to the development team all quality related to-dos that should be taken care of in the system design phase.

Software Quality Design
Small practice

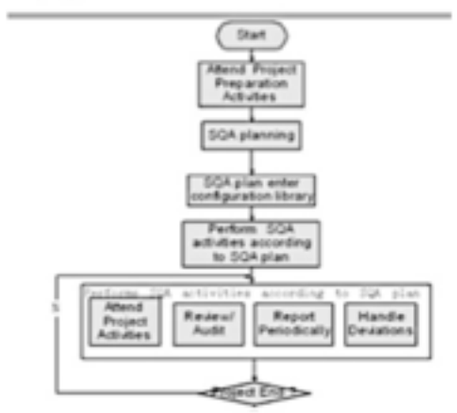
- ***Quality Engineer Bob*: from my quality requirements definition phase I have identified the following, system-level quality attributes:**
  - *system availability* has to be minimum 98%
  - *the mean recovery time* should not exceed 1 min/failure
  - the customer requests that the *relative user efficiency* be at least 0.9.
- ***System Engineer Frank*: Well, it sounds Ok, but what I am supposed to do with it?**
- ***Quality Engineer Bob*: these attributes have the following impact on your design:**
  - *the availability* of 98% means that the system must be available 59 minutes out of every operational hour at any time, so your design should strive to exhibit high reliability. One of the options you may take into account is using stable technologies and enhanced application of reuse.
  - *recovery time* below 1 min/failure may request applying super efficient recovery algorithms together with extra fast hardware.
  - *user efficiency* of 0.9 and above translates into such a functional, user interface and on-line help design that there will be virtually no difference in operation between and average user and an expert.

Software Quality Implementation (1)

Corporate
Software Quality Strategy

Software Quality
Assurance Processes

SQA Process



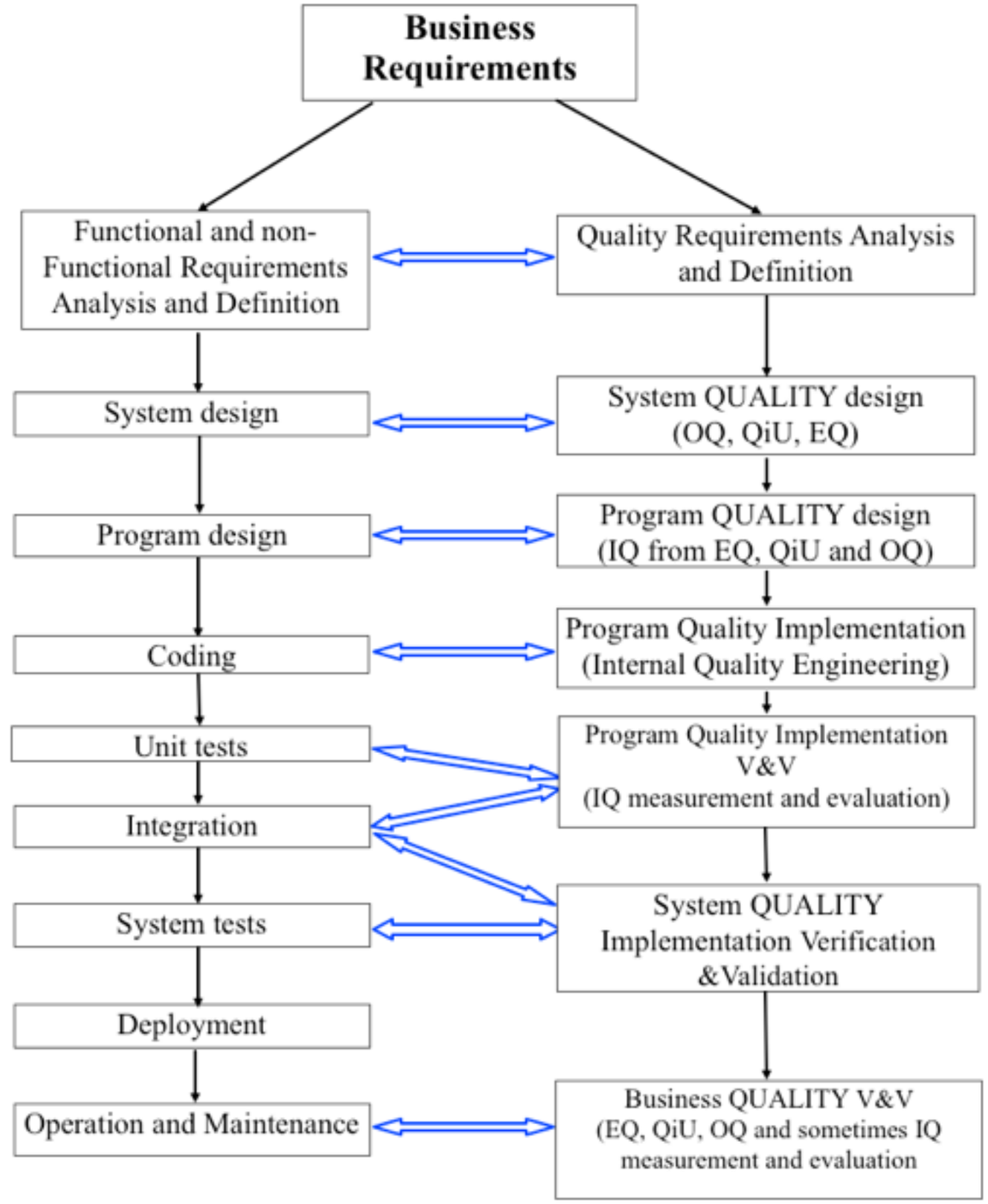
+

Software Quality Engineering

Software Quality Implementation (2)

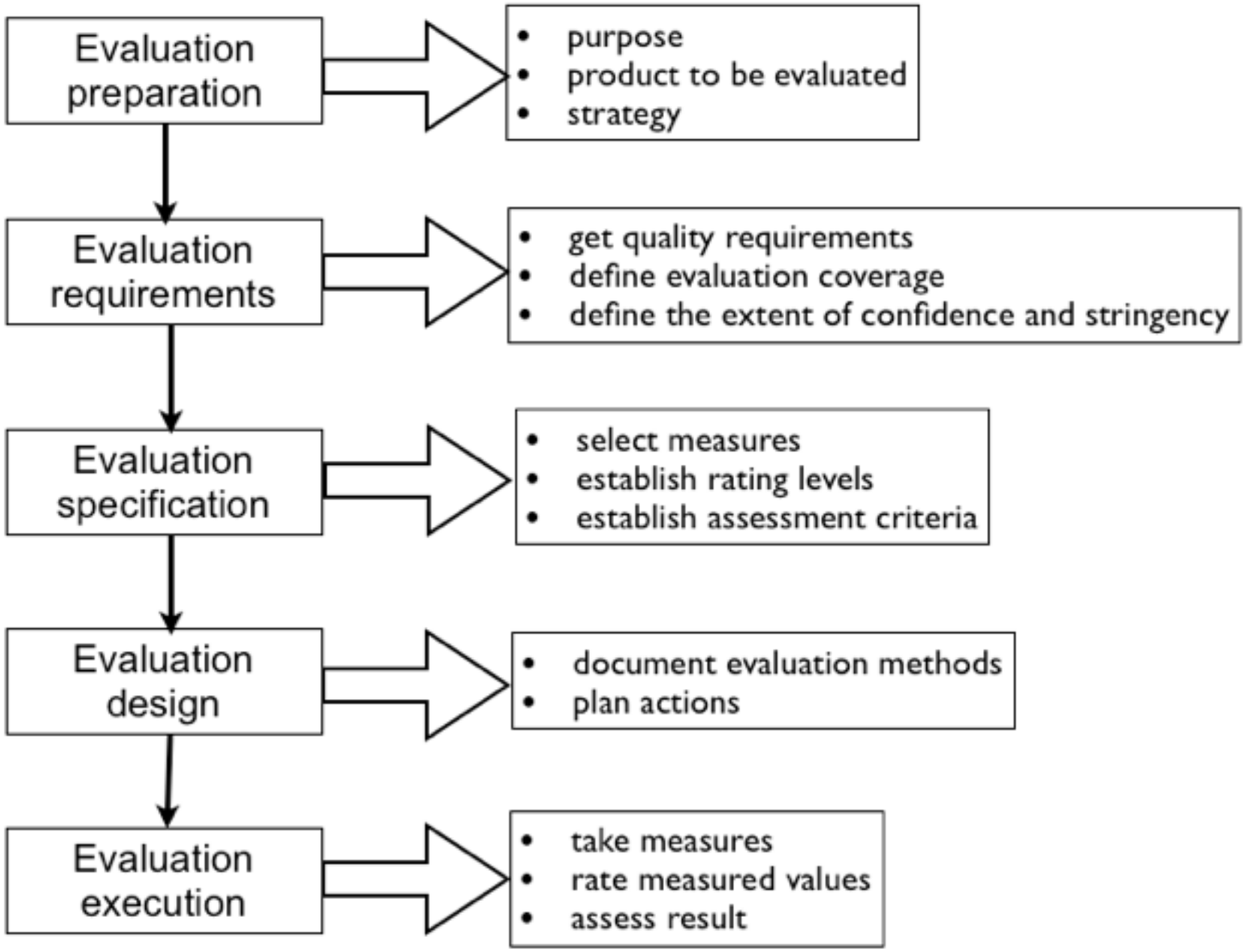
- **Missing strategy**
  - frequently observable state of small and medium companies
  - often quality assurance processes are not there yet too
  - so what are their chances?
  - Similar to these of winning the war, but as long as they have soldiers who can use arms (engineering), they still have chances to win the battle.
- **Missing process**
  - with or without strategy, surely can make things messier than required.
  - Their lack may slow down the development, increase costs, decrease productivity and generally do a lot of secondary harm to the project itself,
  - but again, as long as the engineering force is there, the final product has the chances for good quality.
- **Missing engineering**
  - is like trying to win the war having a superb strategy, exceptional management and control processes and no soldiers.
  - **There is practically no chance to build a quality product**

Software Quality Implementation (3) Model SQIM

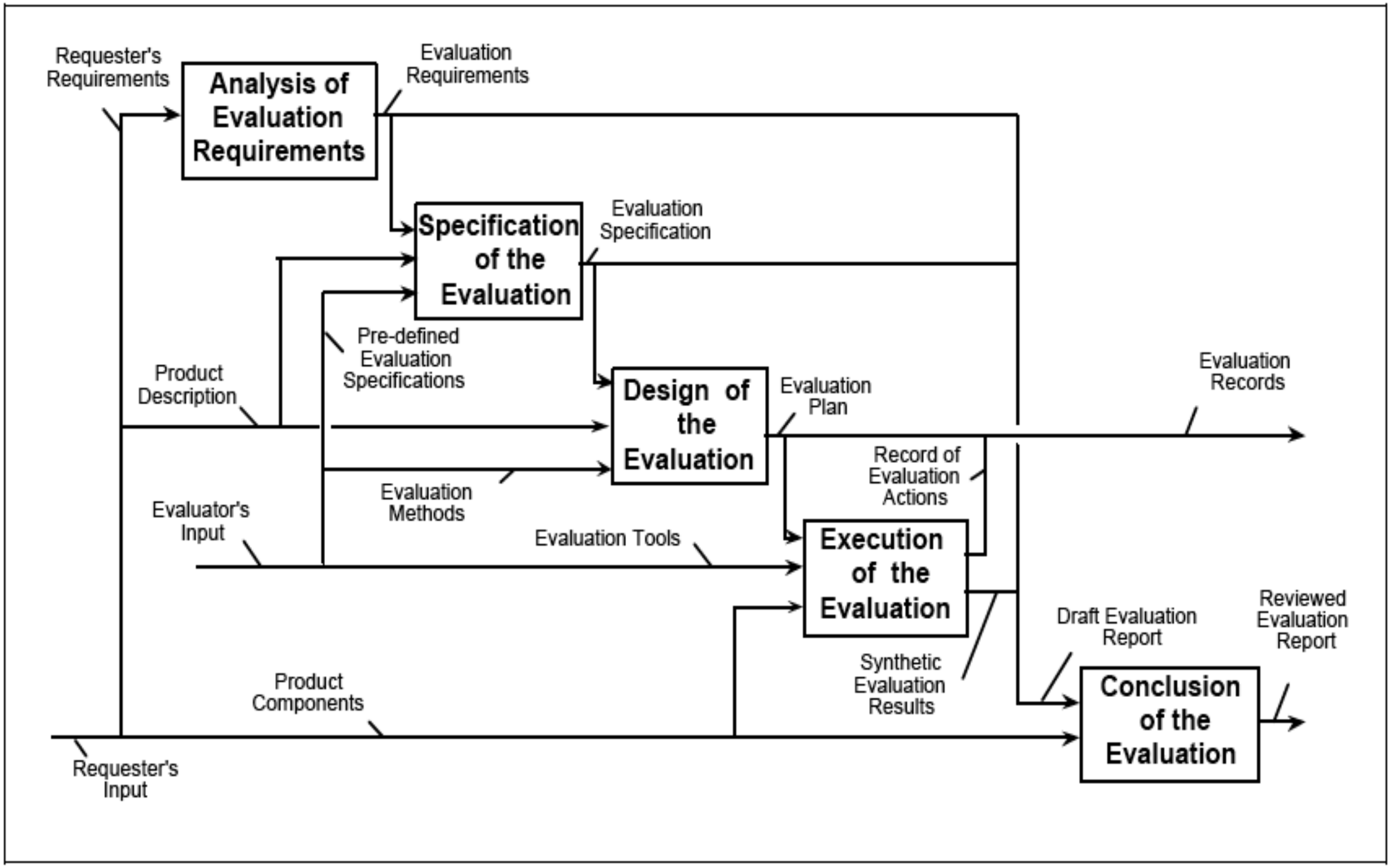


Where:
- OQ – operational quality (based on TL9000)
- QiU – quality in use (from ISO/IEC 9126 and ISO/IEC 25010)
- EQ – external quality (from ISO/IEC 9126- and ISO/IEC 25010)
- IQ – internal quality (from ISO/IEC 9126 and ISO/IEC 25010)

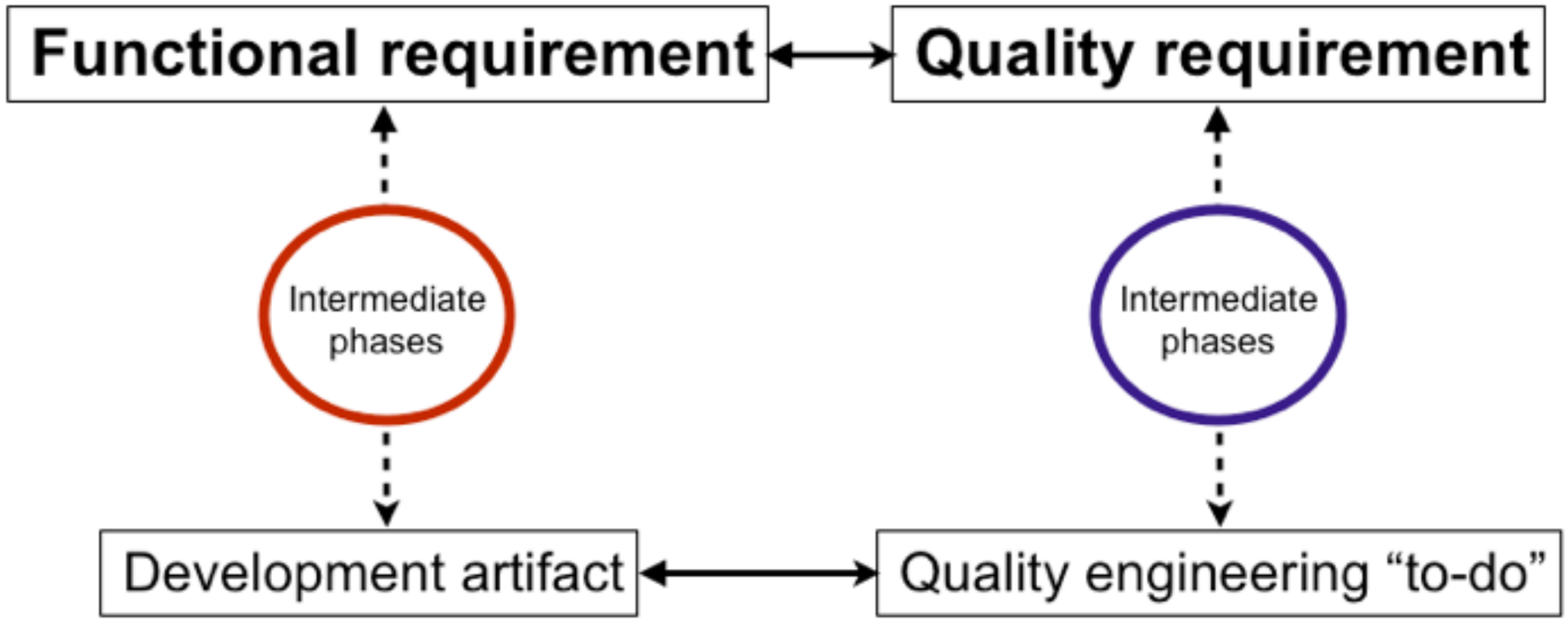
Software Quality Evaluation (1)



Software Quality Evaluation (2)



Software Quality Traceability (1)

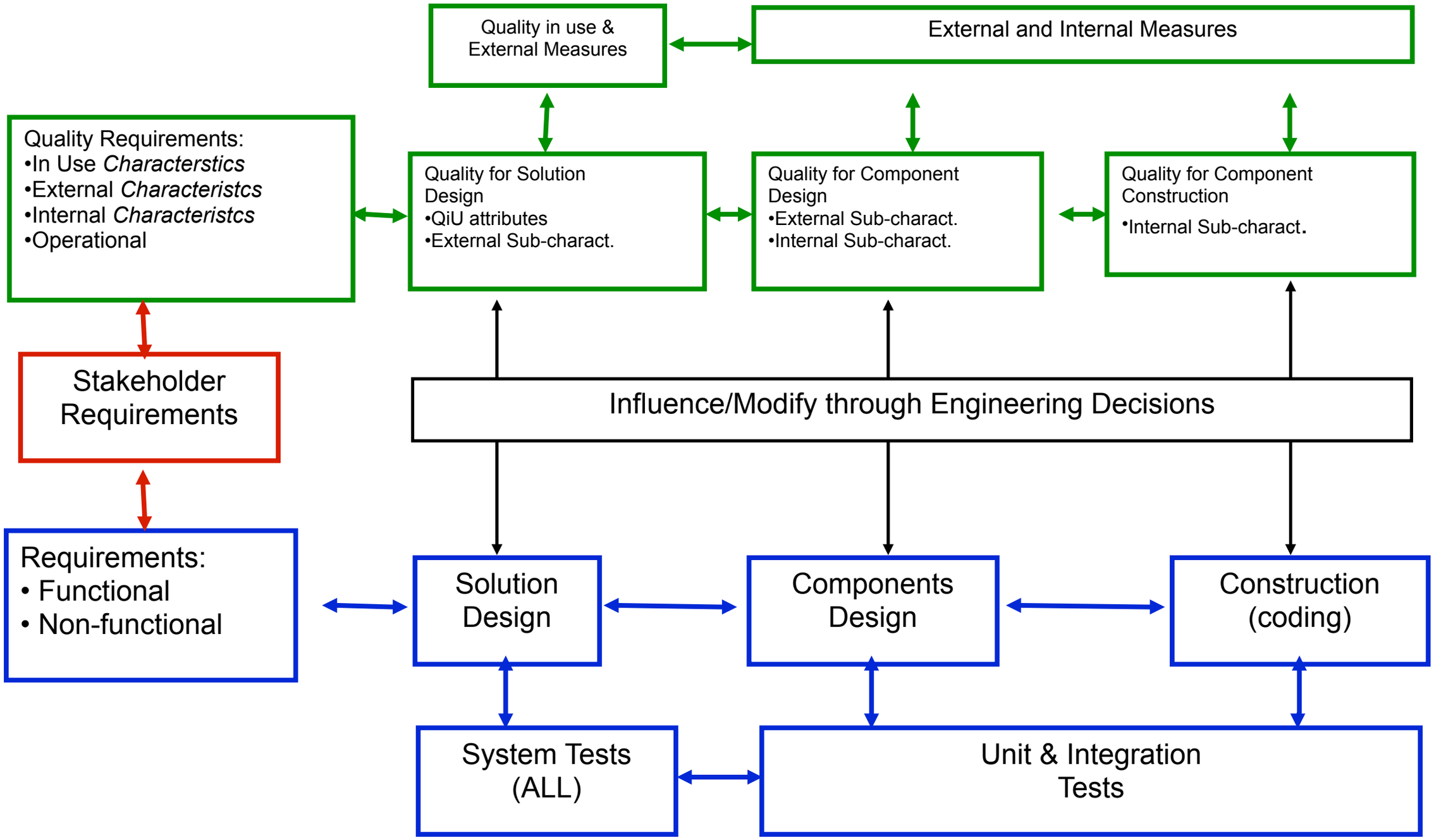


1. The diagram above could be read as follows: ***the development artifact A is originated by a functional requirement B that is associated with quality requirement C that is translated into quality engineering "to-do" D that is applied to development artifact A.***

2. In a correct traceability situation this phrase should be readable forward and backward (just try if for fun)

3. If you find yourself in a situation where this phrase goes in only one way (missing arrow or arrowhead), something is wrong with the artifact you are observing

Software Quality Traceability (2)



Software Quality - Trustworthiness

- **Definitions:**
  - *[human]* Trustworthiness is a moral value considered to be a virtue. A trustworthy person is someone in whom we can place our trust and rest assured that the trust will not be betrayed (Wikipedia)
  - *[history]* Refers to the reliability and authenticity of records (Minnesota State Archives)
  - *[engineering]* deserving of trust or confidence; dependable; reliable (Dictionary)
  - *[IT systems use]* An entity is trustworthy if there is sufficient credible evidence leading one to believe that the system will meet a set of given **requirements**. Trust is a measure of trustworthiness, relying on the evidence provided (M.Bishop, “Computer Security: Art and Science”).

What trustworthiness means in IT domain (1)

- **User *perception* of trustworthiness:**
  - Quality
  - Reliability
  - Completeness of required functionalities
  - Proper quality-cost ratio (we do not overpay)
  - Post-sale maintenance and service
  - Pre and post-sale training
  - Documentation
  - Responsibility for the product

What trustworthiness means in IT domain (2)

- **Let us recall the "now"(1):**
  - COTS (Commercial Off-The-Shelf) applications:
    - applications for massive customer with "no face", with predefined set of functionalities
  - CM (Commercial Modular) systems:
    - systems built from pre-developed modules or subsystems for a known user, having large, configurable yet finished set of functionalities. In most cases functionalities are linked into *services*
  - Individual systems:
    - Developed to meet individual and unique requirements of the known user. The system may offer functionalities, services or both
  - What do they have in common?
    - we own them (or at least the license to use them)
    - we know who build them
    - we know whom to pursue (or sue) if something is wrong
    - the trustworthiness in this context is ***manageable***

What trustworthiness means in IT domain (3)

- **Let us recall the "now"(2):**
  - SOA - service oriented architecture
    - A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve two or more services coordinating some activity in order to "do a job"
    - The technology of Web services is the most likely connection technology of service-oriented architectures
    - Is it new? No. DCOM, CORBA are younger brothers
  - Cloud computing
    - Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS):
      - IaaS - virtual server instances with unique IP addresses and blocks of storage on demand
      - PaaS - set of software and product development tools hosted on the provider's infrastructure
      - SaaS - vendor supplies the hardware infrastructure, the software product and interacts with the user through a front-end portal

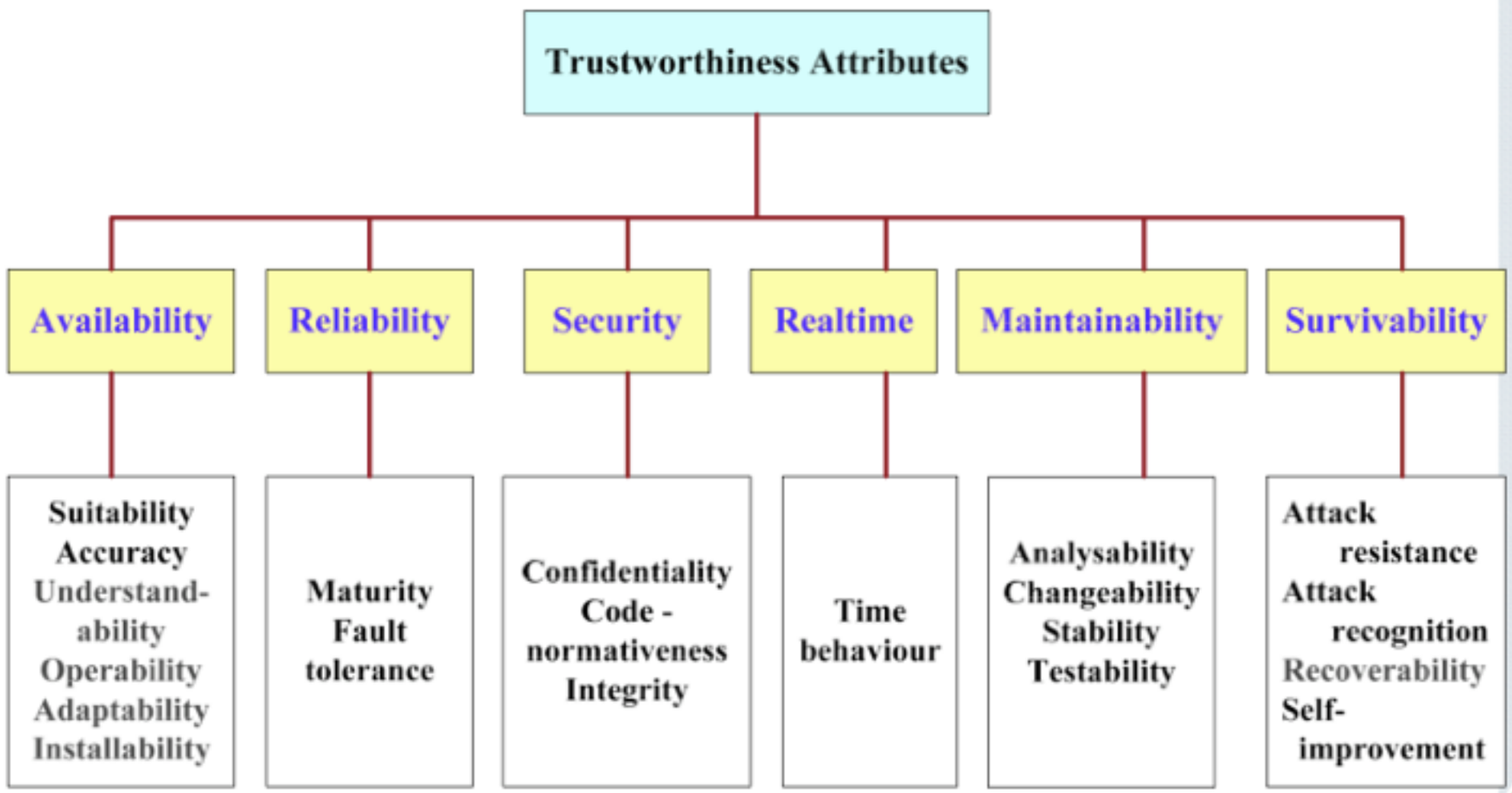
Trustworthiness' place in today's world

- **Some data**
  - In 2006, the amount of digital information created, captured, and replicated was 1,288 x 1018 bits. In computer parlance, that's 161 exabytes or 161 billion gigabytes. This is about 3 million times the information in all the books ever written.
  - Between 2006 and 2010, the information added annually to the digital universe increased more than six fold from 161 exabytes to 988 exabytes.
- **What happens if only 0.001% of important information becomes untrustworthy?**
- **What happens if some of the systems involved in processing these data go "untrustworthy"?**

Trustworthiness - classical approaches to evaluation (example)

Trustworthiness attribute model

- The model is the collection of the key properties that describe software trustworthiness



www.trustie.net

Trustworthiness “in new” (1)

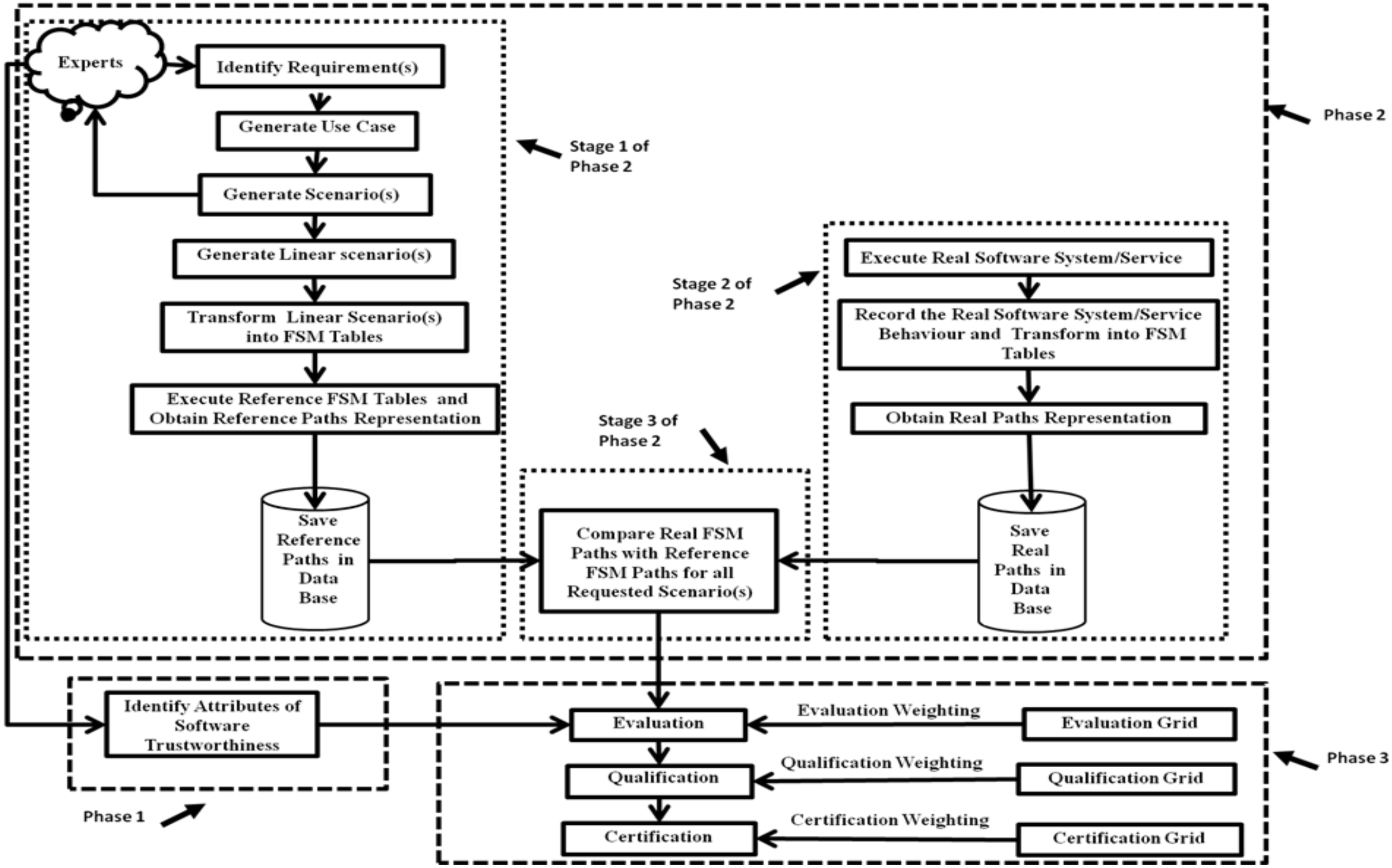
Behaviouristic Method of Software, System and Service Trustworthiness Certification

- Developed in international cooperation by:
  - Dr. **Yuyu Yuan** (Beijing University of Post and Telecom, China)
  - Dr. **Witold Suryn** (École de technologie supérieure, Montreal, Canada)
  - Mr. **Jørgen Bøegh** (Terma A/S, Denmark)

Trustworthiness “in new” (2)

- **New hypotheses**:
  - Trustworthiness ≠ quality
    - Quality makes a very important component of trust, but alone is not enough (like high quality system that misses some services)
  - Trustworthiness does not have to be “*total*”
    - The context of use shall target the area where the service has to be trustworthy (like a friendly thief who can be trusted to never steal from us)
  - Trustworthiness does not have to be “*parametric*”
    - Classic approaches identify and measure attributes (“parameters”) to evaluate software or system trustworthiness
  - Trust given to IT system can be based on the way it serves our purpose or, with more suitable vocabulary, **HOW IT BEHAVES**.
  - System **behavior** can be described and defined using one of known formal methods, like for example Finite State Machine (**FSM**)

Trustworthiness “in new” (3) BEMSET



Więcej na ten temat…