Radboud Universiteit

Testing Challenges for Cyber Physical Systems

ESI

Jan Tretmans

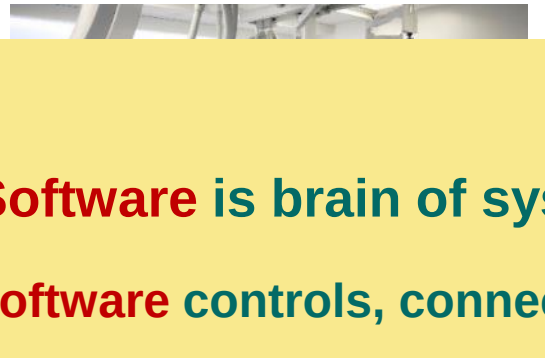
TNO - ESI – Embedded Systems Innovation at TNO
Radboud University Nijmegen
jan.tretmans@tno.nl

TNO innovation for life

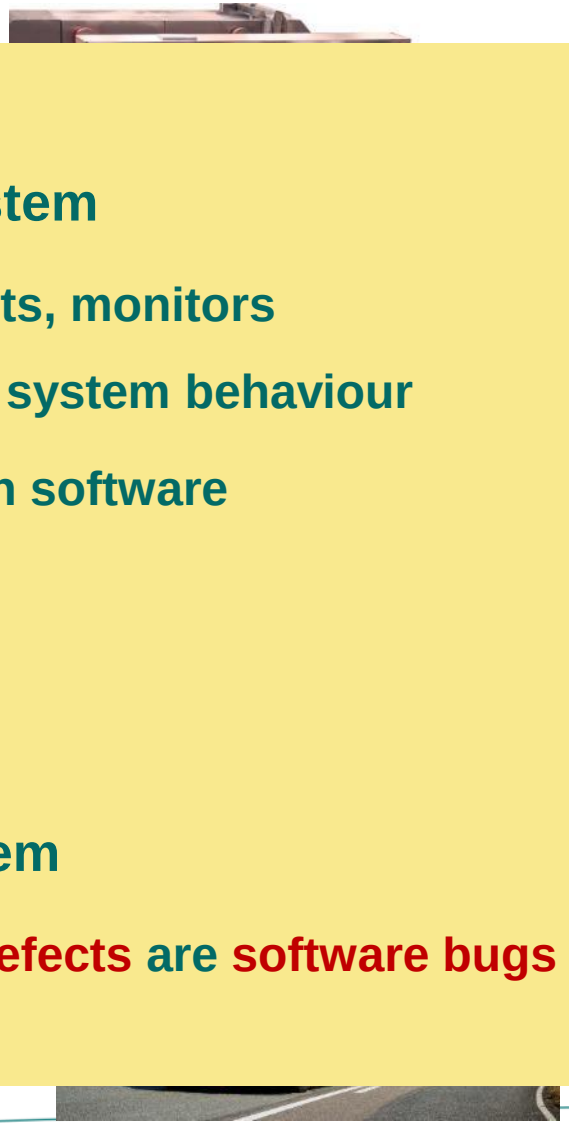
CPS : Cyber Physical Systems

Cyber-Physical Systems

Semiconductor manufacturing equipm



Industrial printers

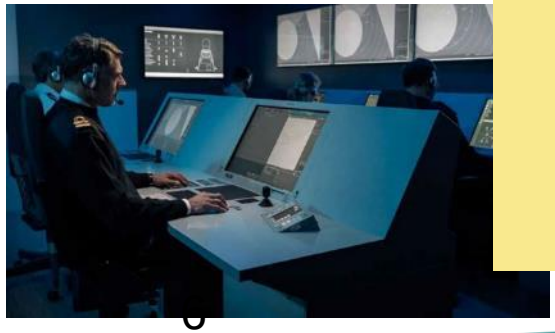


Automotive

Agricultural robots

Traffic management

Robotized warehousing



Combat management systems

Dike

Software is brain of system

- **software controls, connects, monitors**
  almost any aspect of CPS system behaviour
- **majority of innovation is in software**

Software determines
quality and reliability
of Cyber-Physical System

- **often > 50 % of system defects are software bugs**

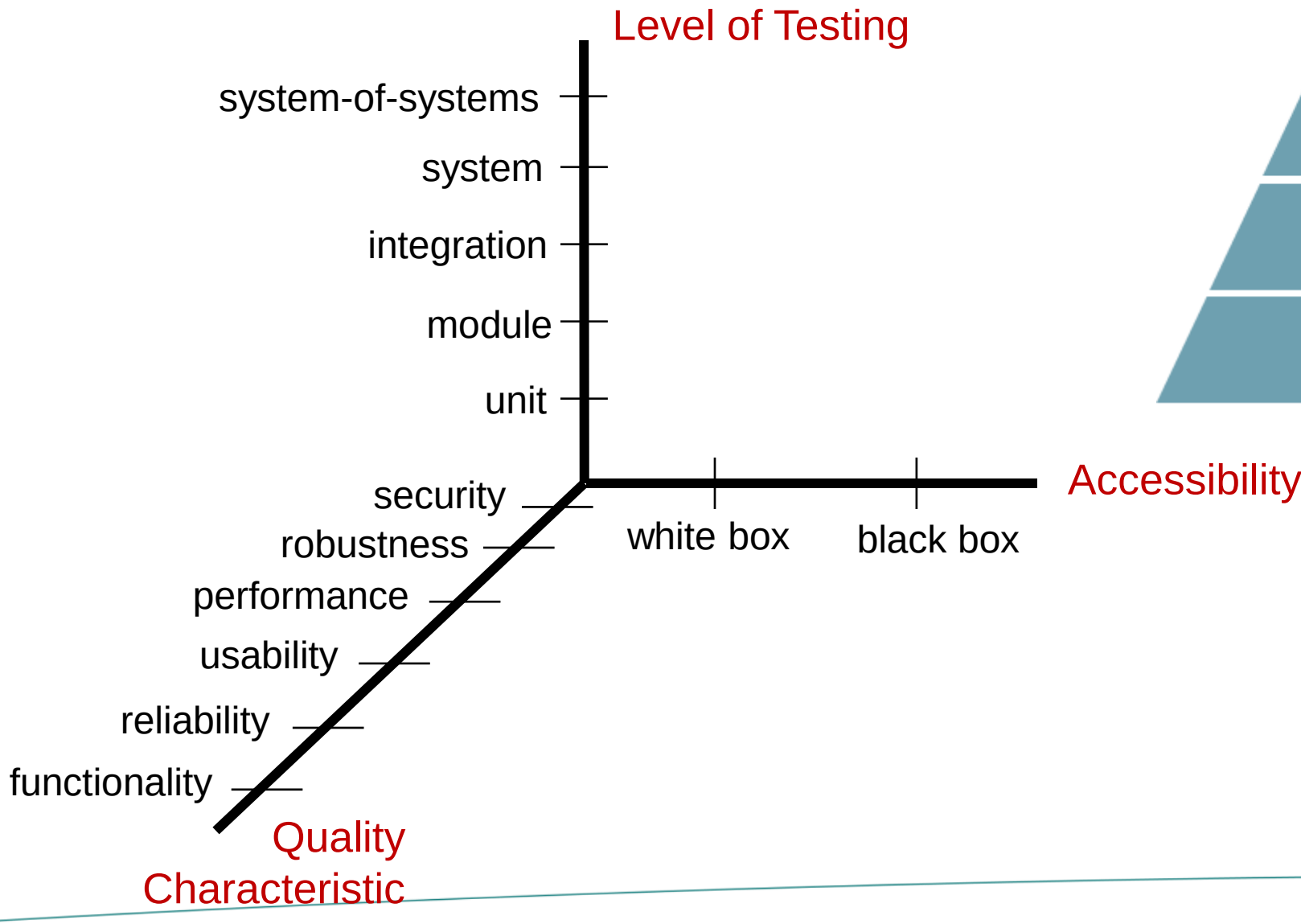
Software Testing

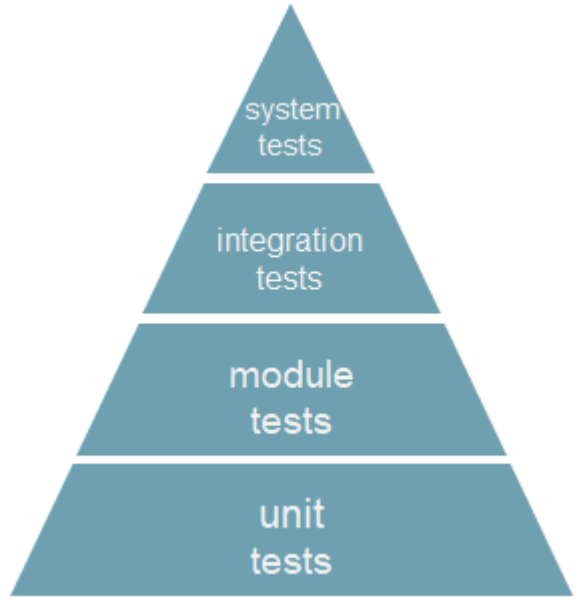
Testing: A Definition

Software testing is:

- a technical process,
- performed by executing / experimenting with a product,
- in a controlled environment, following a specified procedure,
- with the intent of measuring one or more characteristics / quality of the software product
- by demonstrating the deviation of the actual status of the product
  from the required status / specification.

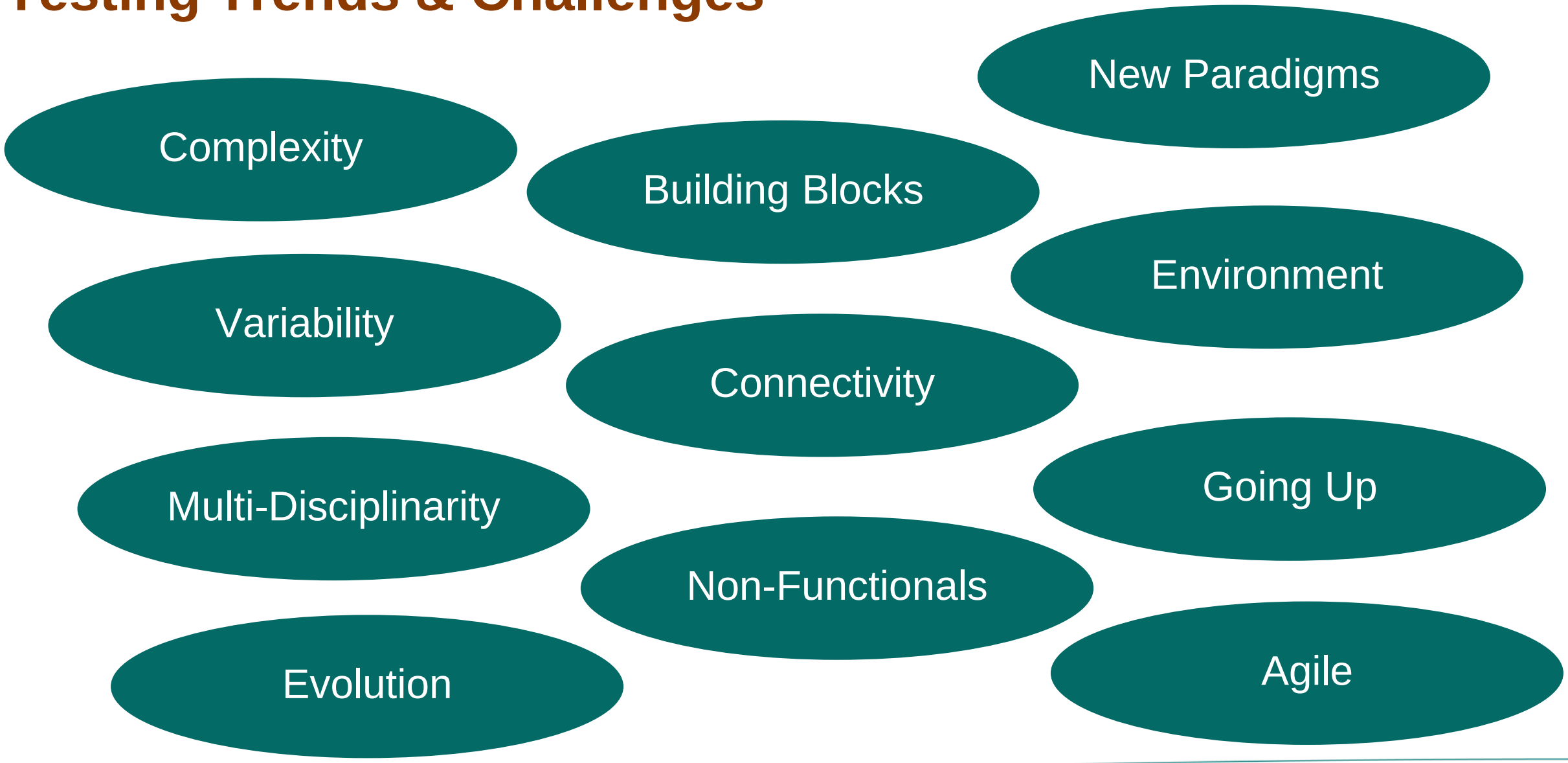
Sorts of Testing



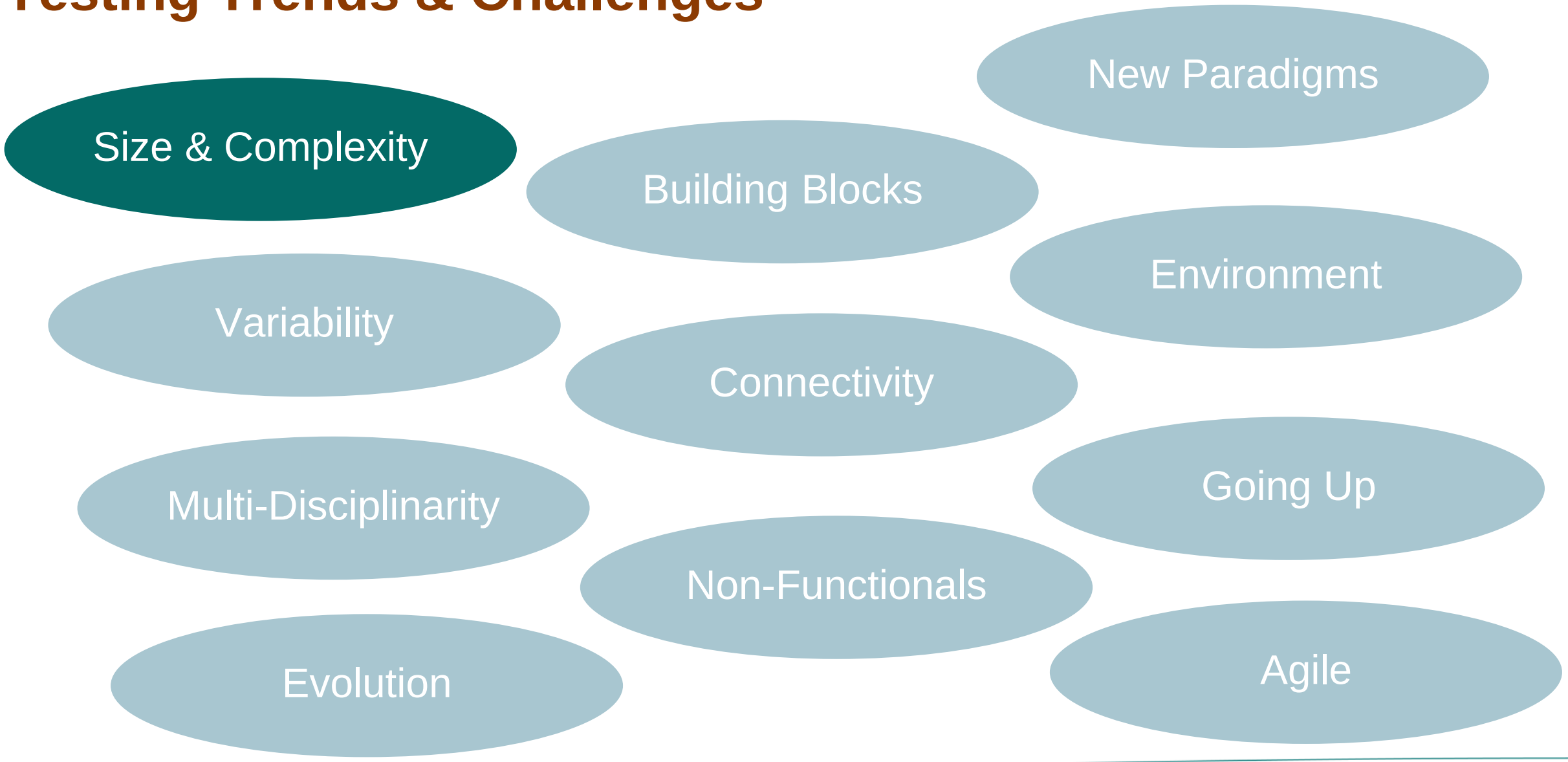


Testing Challenges for Cyber-Physical Systems

Testing Trends & Challenges



Testing Trends & Challenges



Size & Complexity

Completely testing ‘ + ’ for 32-bit Int

- $2^{32} * 2^{32} = 10^{19}$ test cases
- 1 nsec / test = 585 years of testing

Car

- 100,000,000 LoC
- 40,000 parts
- 4,000 manufactured components

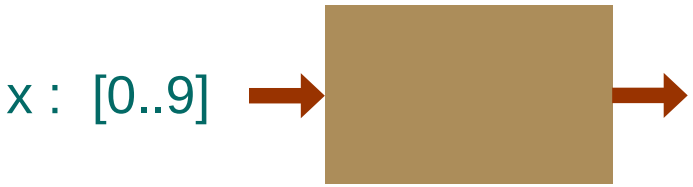
Machine with 300 parameters

- $2^{300} = 10^{90}$ different configurations
- #atoms on earth = 10^{50}, #atoms in known universe = 10^{80}

Size & Complexity

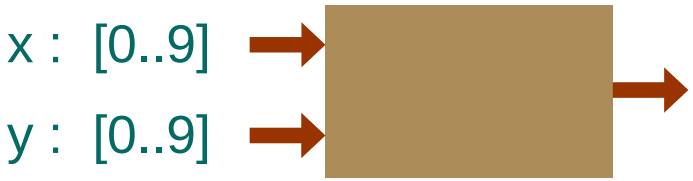
Testing effort grows exponentially with system size

Testing cannot keep pace with development



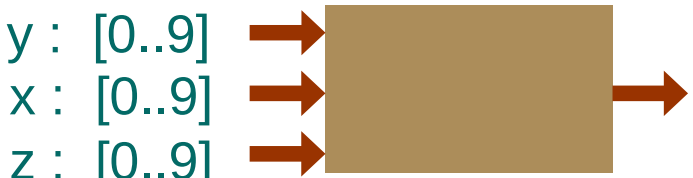
10 ways that it can go wrong

10 combinations of inputs to check



100 ways that it can go wrong

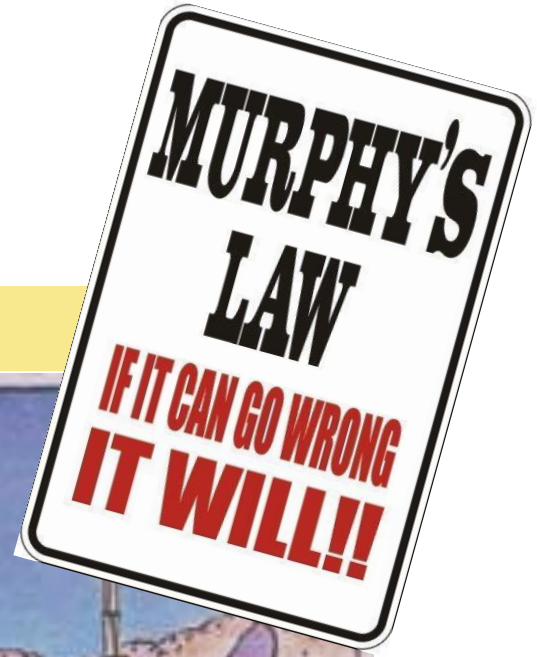
100 combinations of inputs to check



1000 ways that it can go wrong

1000 combinations of inputs to check

Size & Complexity

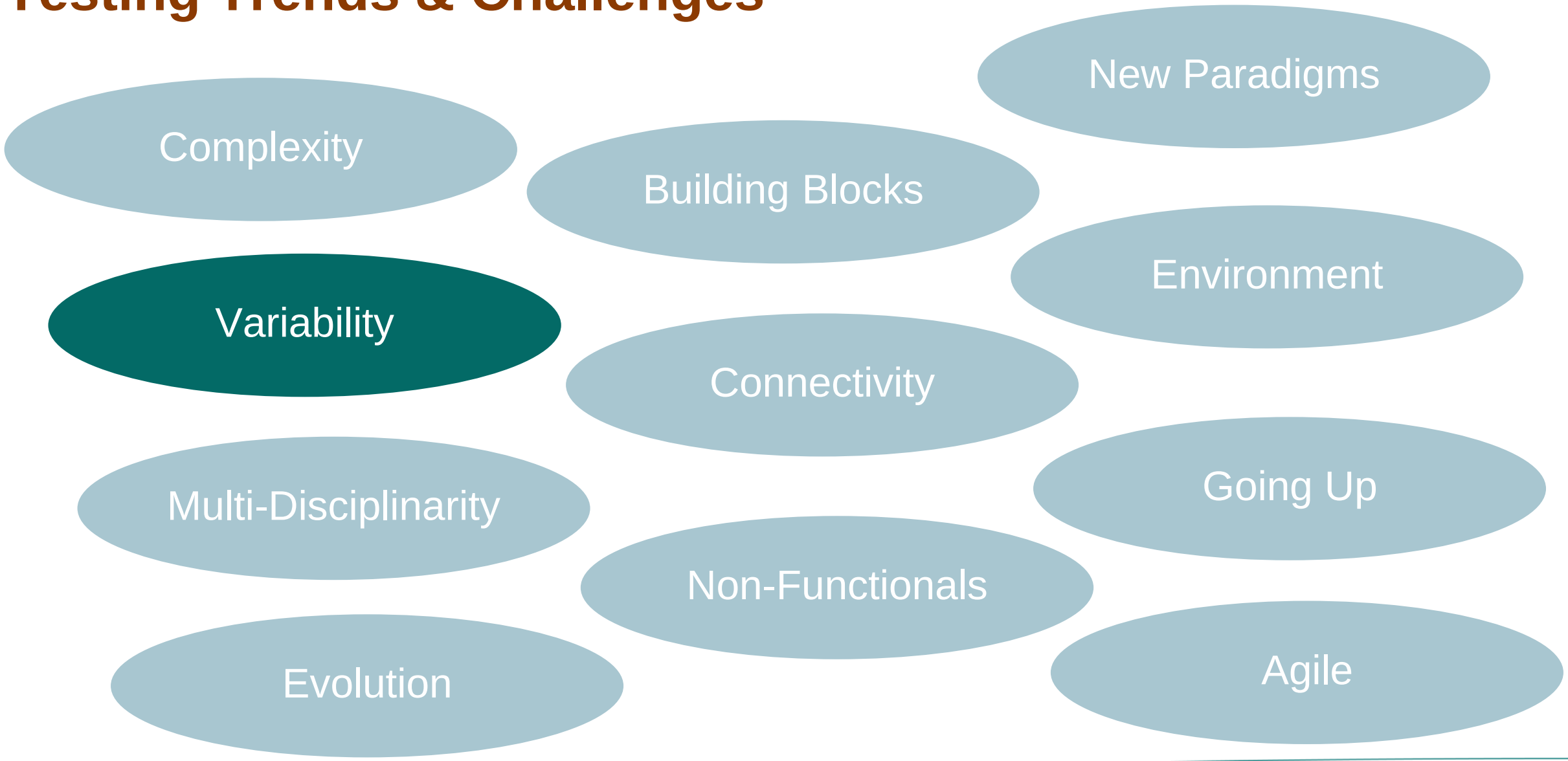


Testing effort grows exponentially with system size

Testing cannot keep pace with development

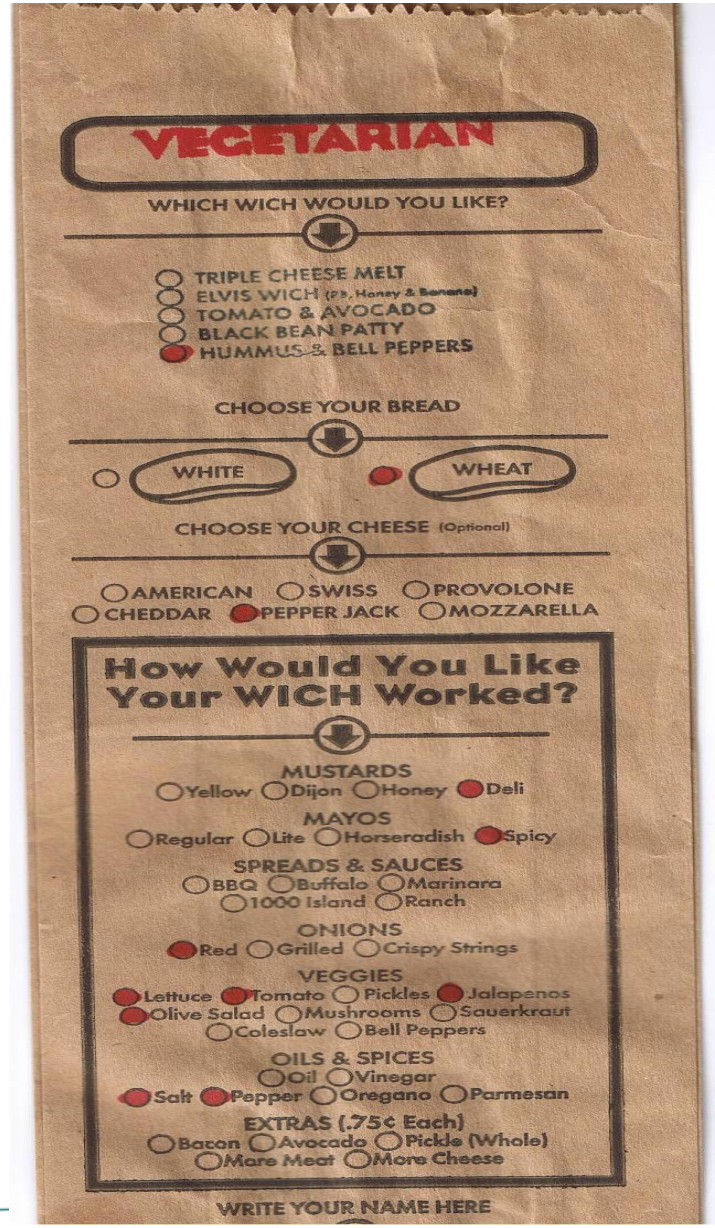
→ combinatorial explosion of required testing effort

Testing Trends & Challenges

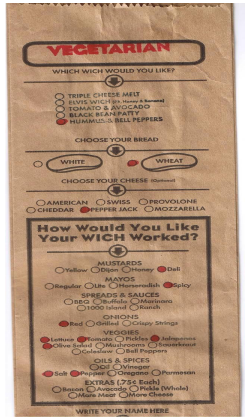


Variability & Product Lines

or: How to Select your Sandwich

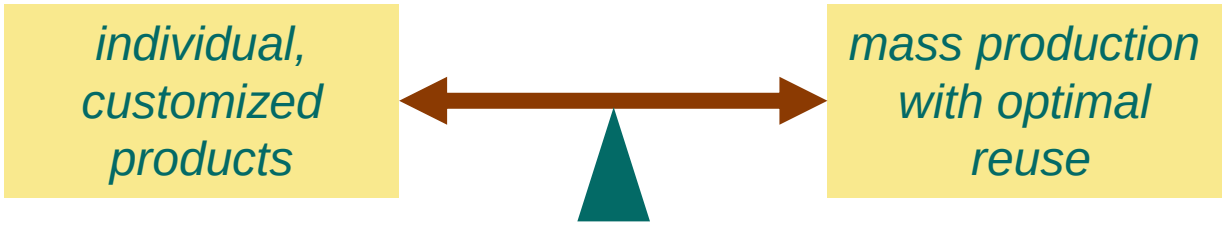


Variability



- Highly configurable sandwich*: exponential number of choices*, **a different sandwich for everybody on the planet!**
- Not all combinations make sense: *dependencies*
- **How to taste / test all of them?**
- Sandwich product line = family of sandwiches
- Also for high-tech systems
  Linux, cars, . . .

Variability Engineering

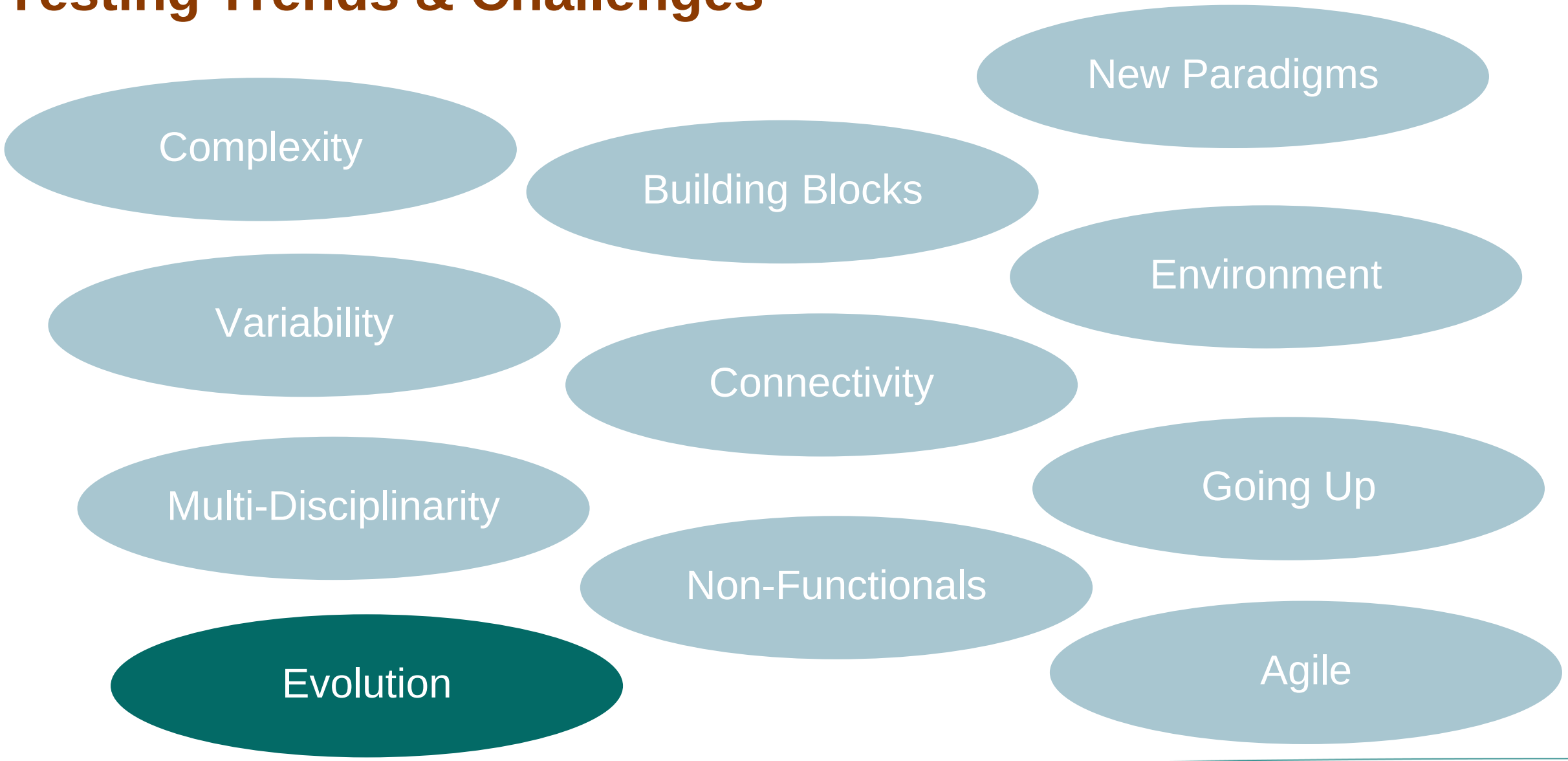


- Customization & reuse by developing families of 'similar' products
  - → identify variation points
  - → instantiate to different configurations = products

- *Aim*: instantiate as late as possible,
  to perform design, analysis, …, on the product family
  and not on each individual product

- *But*: testing is always on an individual product
  - → **how to select configurations for testing ?**

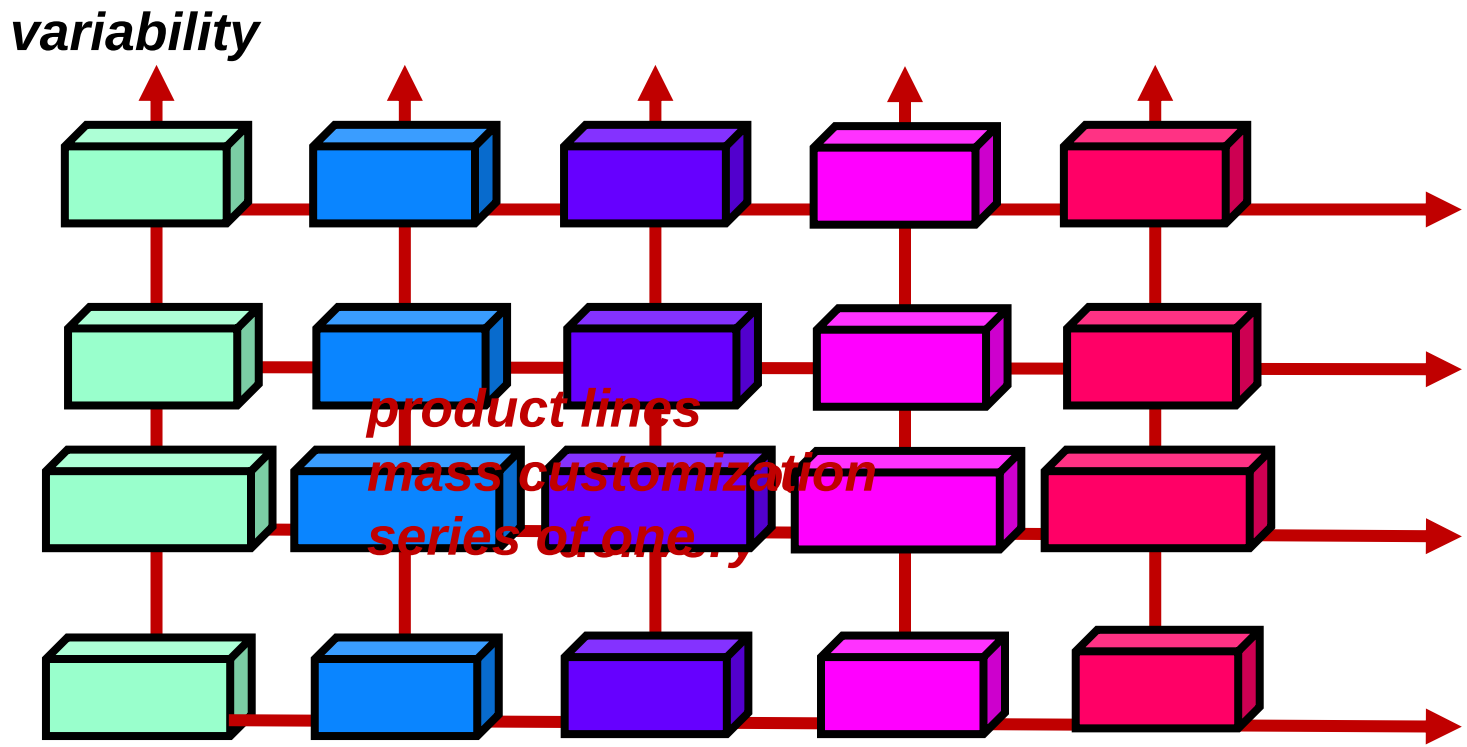
Testing Trends & Challenges



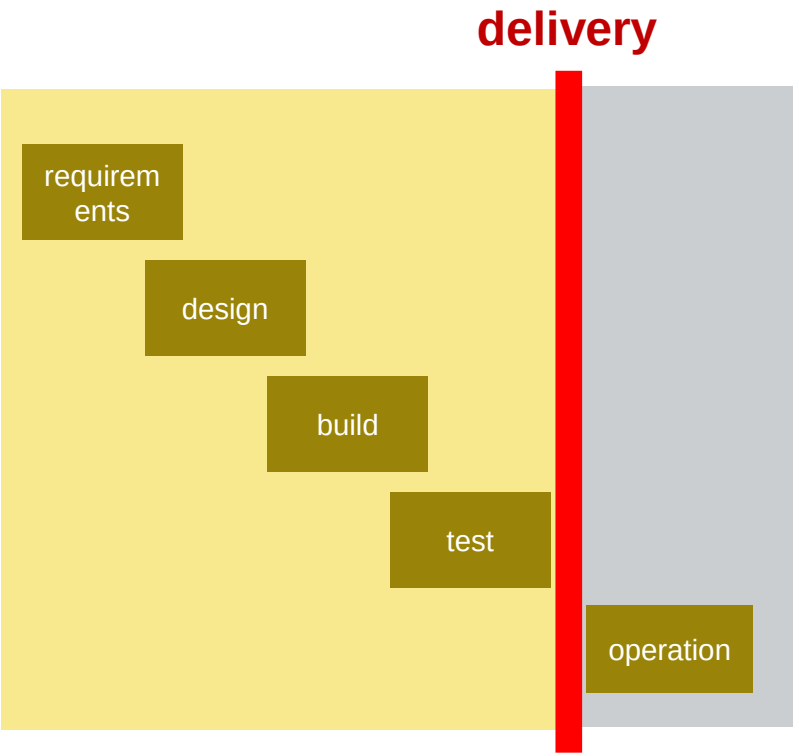
Evolution : Change over Time

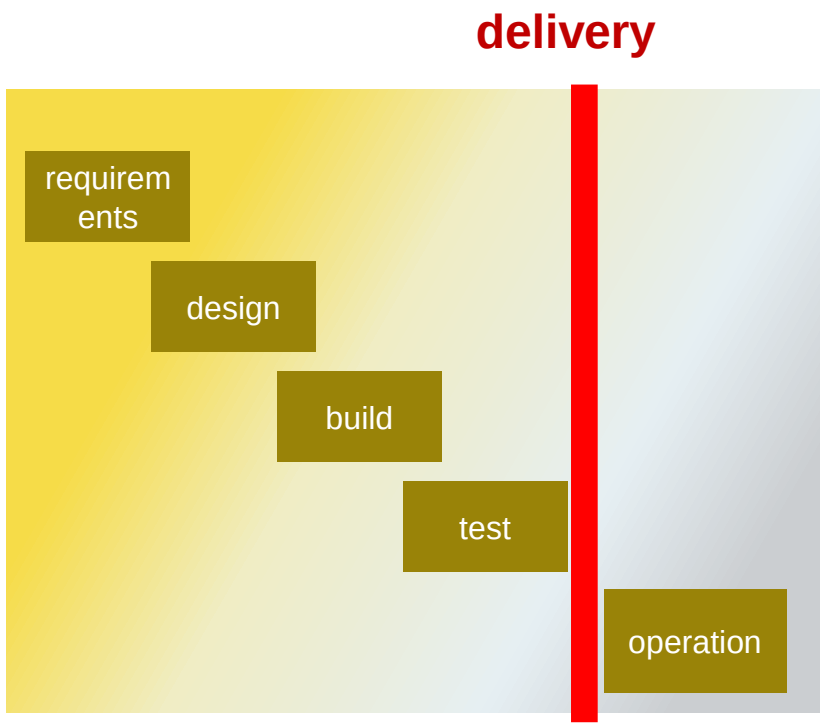
- system never comes alone: *variability*
- systems continuously change: *evolution*

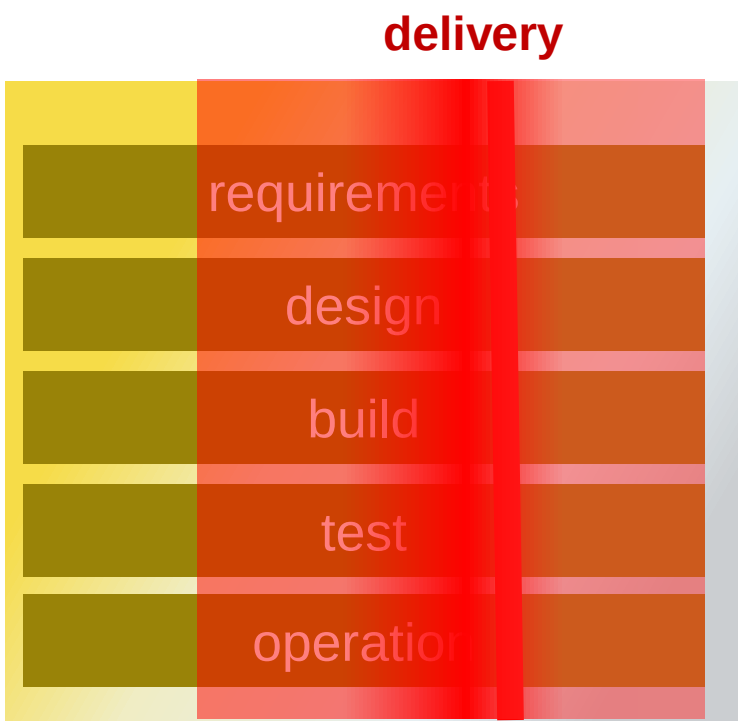
yet another source of
Test Explosion



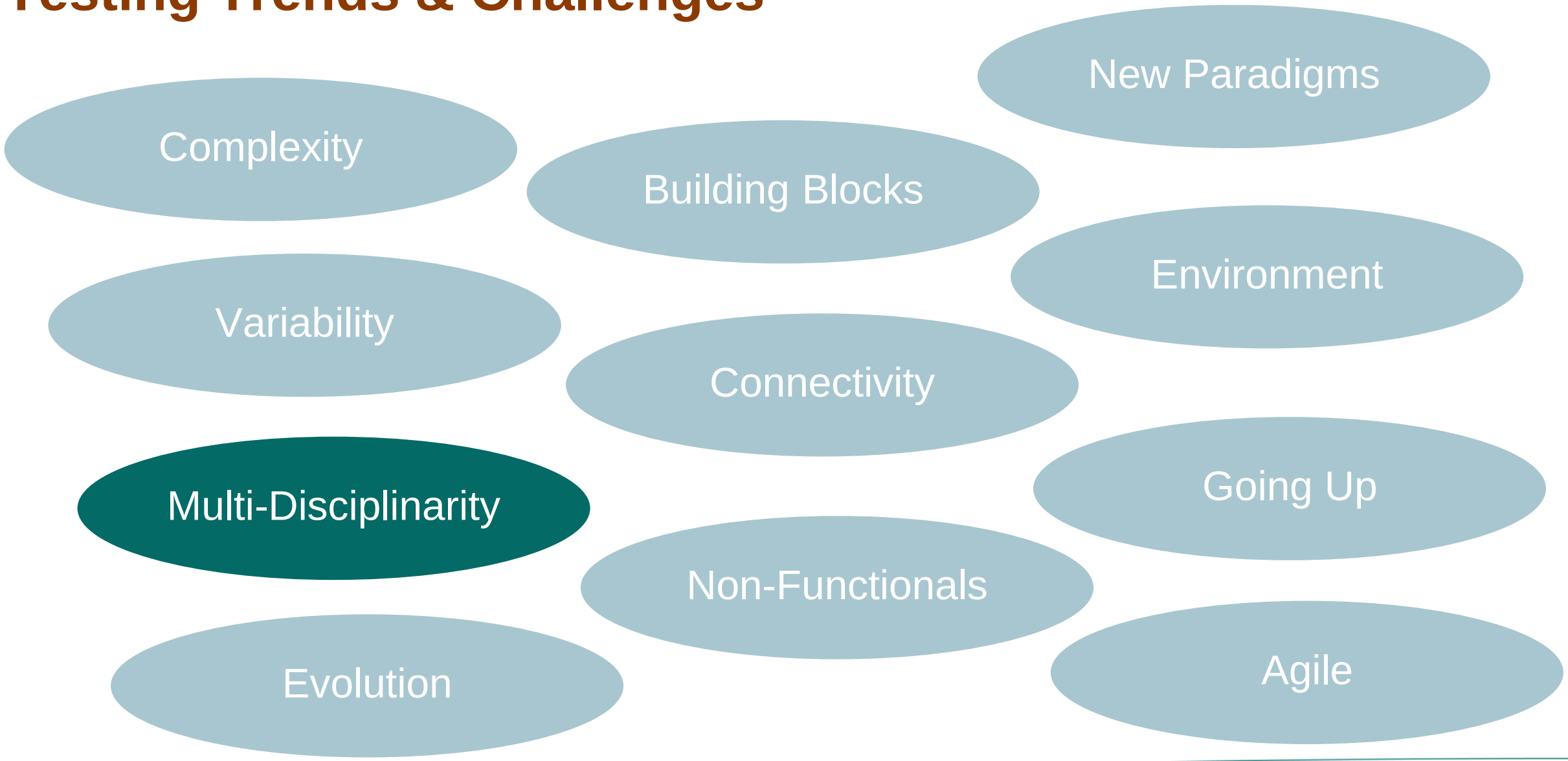
Evolution, Change : Fading Boundaries







Testing Trends & Challenges



Cyber-Physical Systems

Semiconductor manufacturing equipment

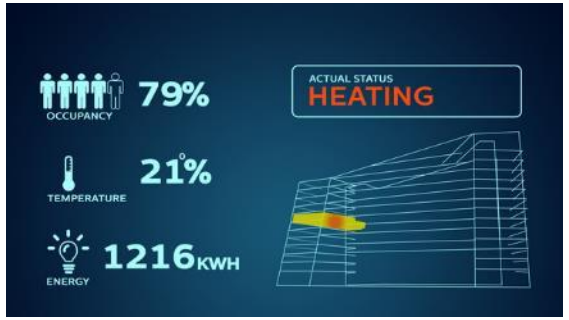
Medical systems

Food processing

Agricultural robots

Traffic management

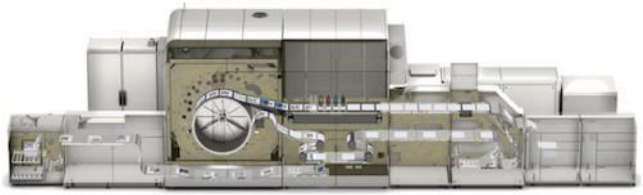
Electron microscopes



Building control

Robotized warehousing

Combat management systems



Industrial printers

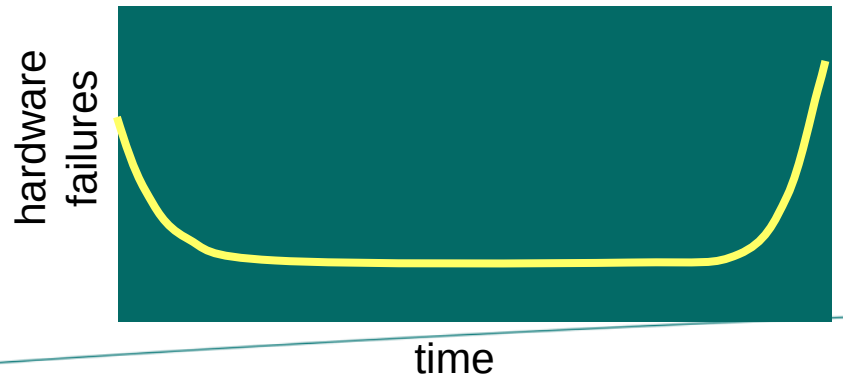
Automotive

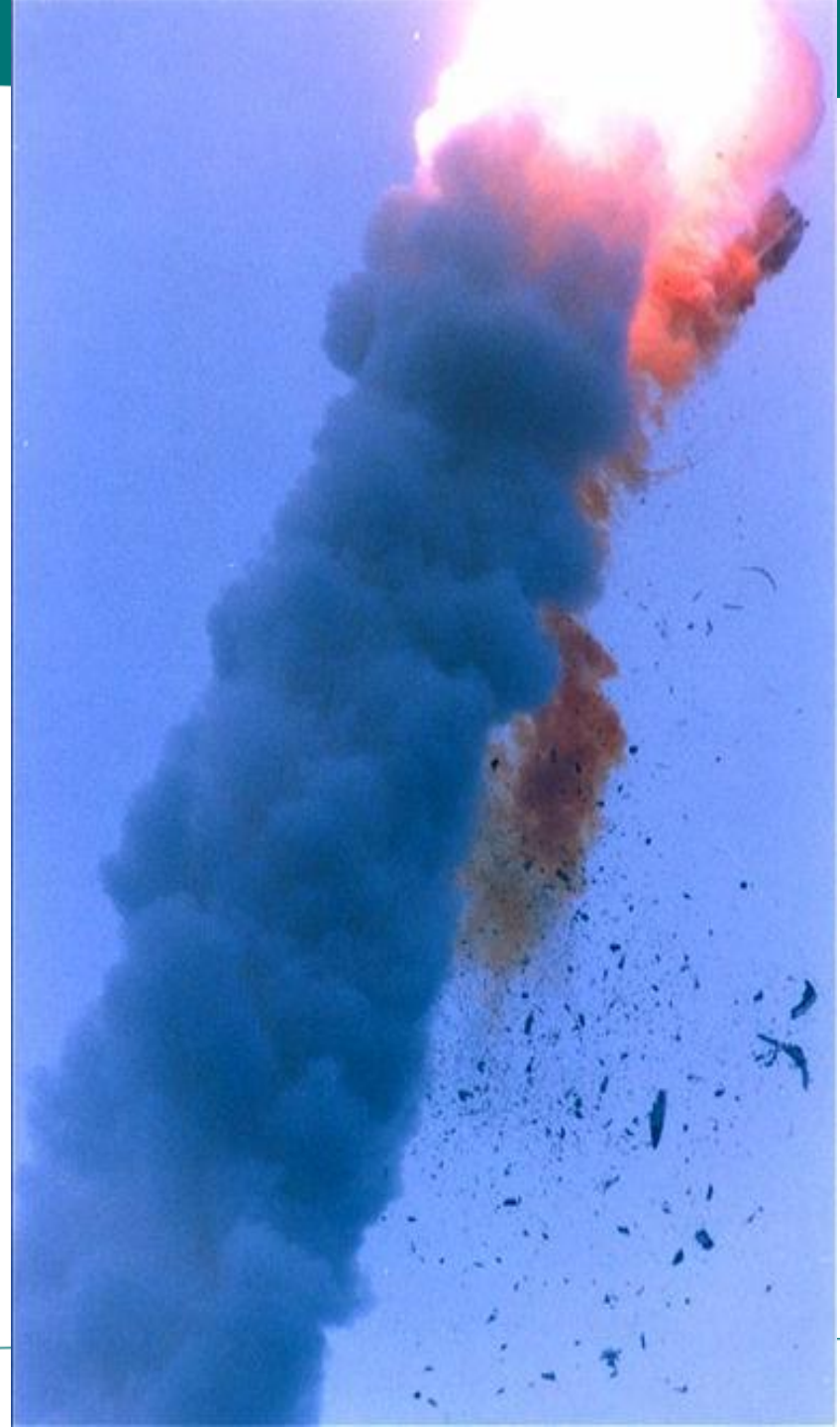
Dike

Software is Different

Software is different from hardware :

- non-continuous
- any bug is a design error
- adopting redundancy is useless
- no wear and tear
- no MTBF
- what is software reliability?

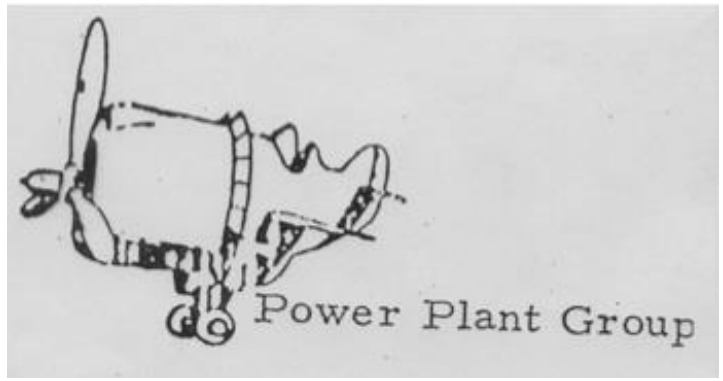


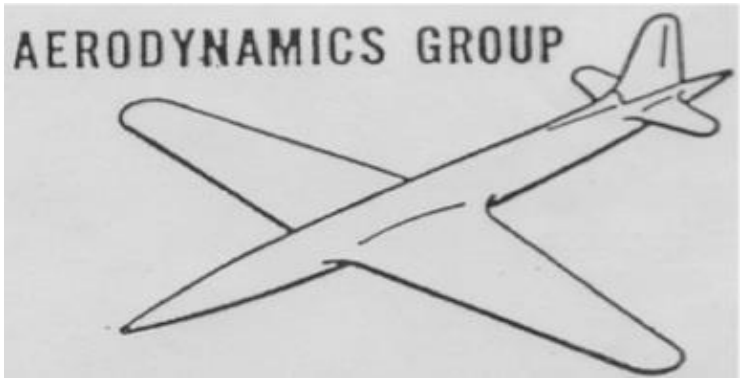


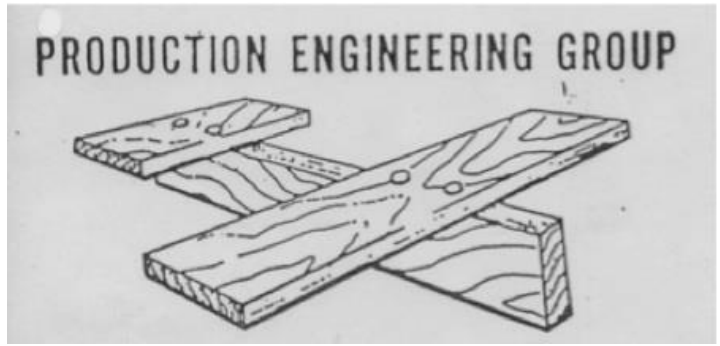
Multi-disciplinarity

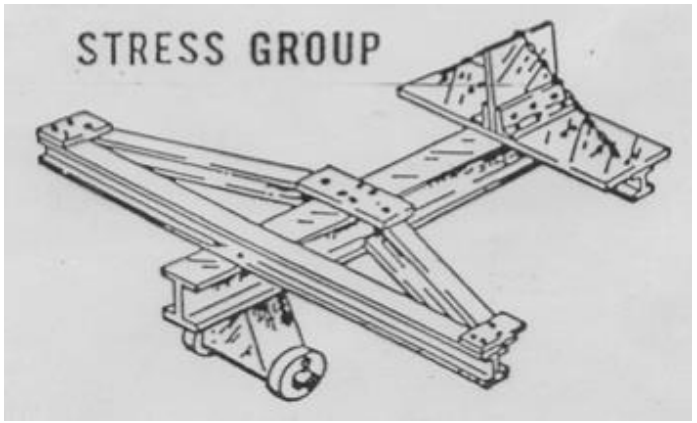
- **Virtualization**
  - models to simulate/emulate physical and environment in t
  - intelligent stub, in-the-loop testing
  - because real system is: expensive, infeasible, dangerous, too slow, too fast, cannot produce error scenarios, …
- **Modeling**
  - system ←→ physical part ←→ software ←→ environment
  - models for virtualization ←→ models for testing

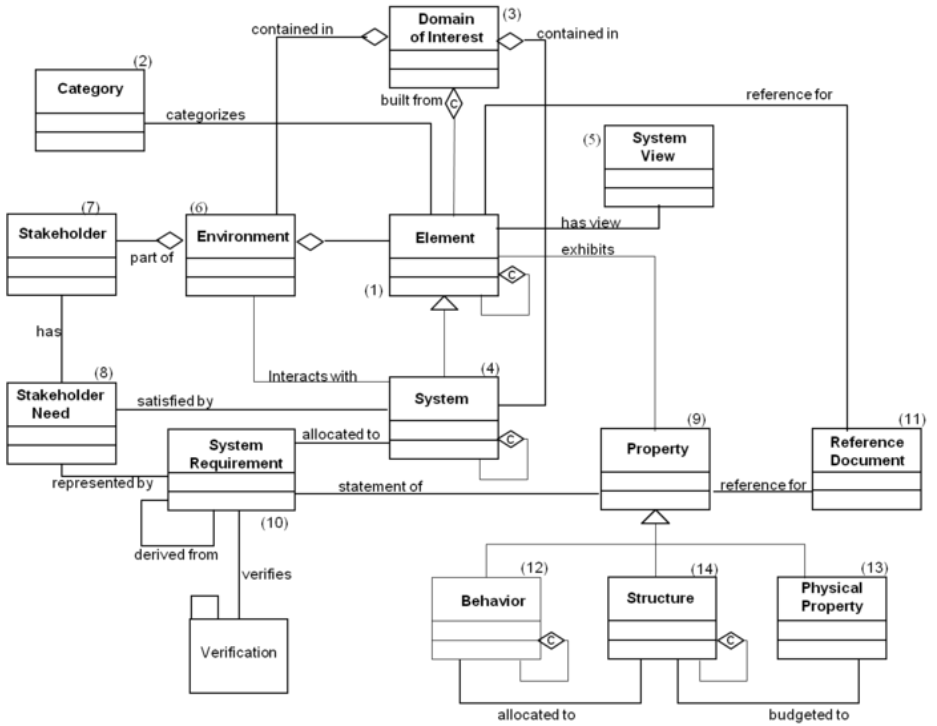
Multi-disciplinarity : Different Views on Systems





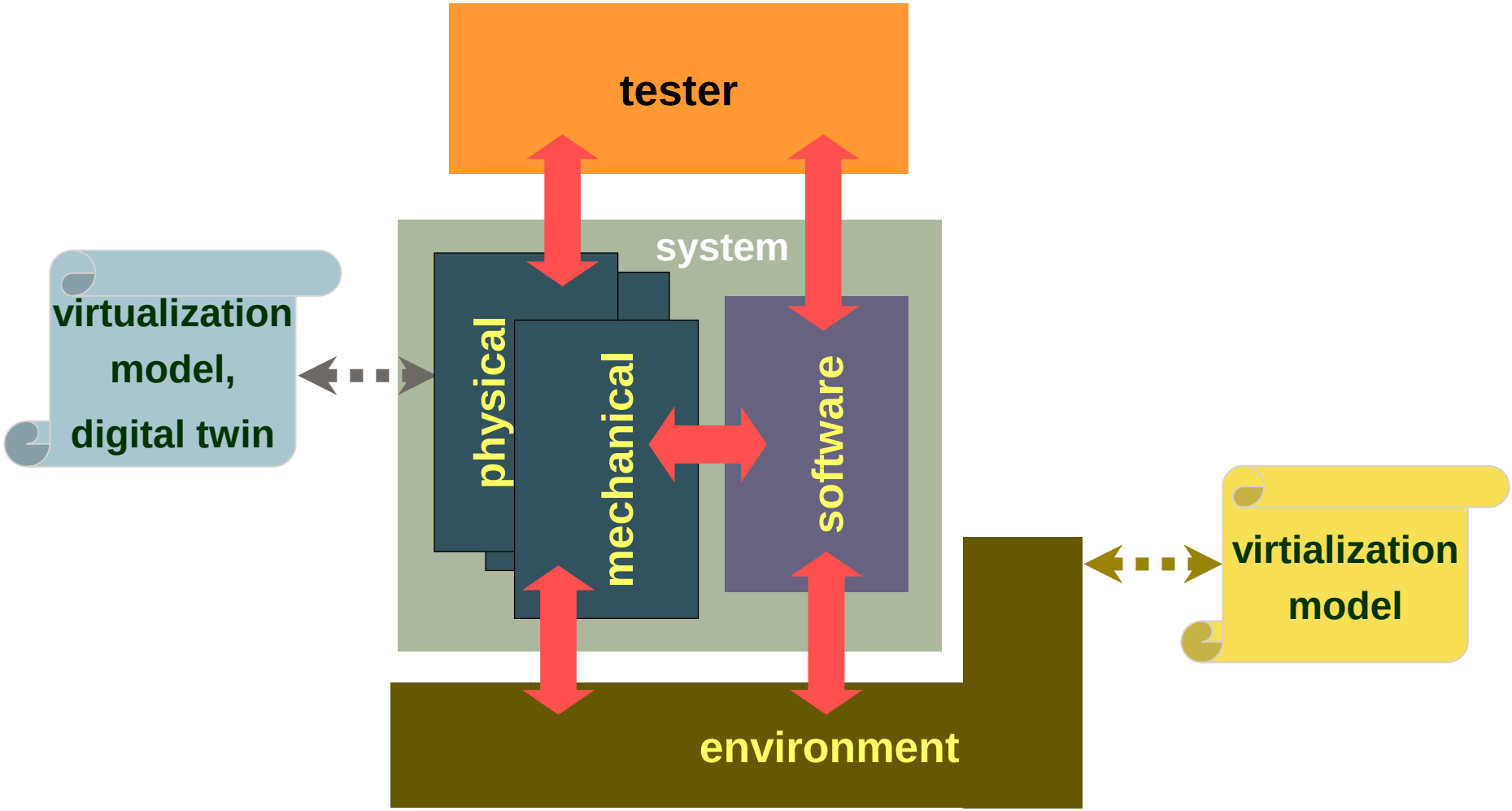




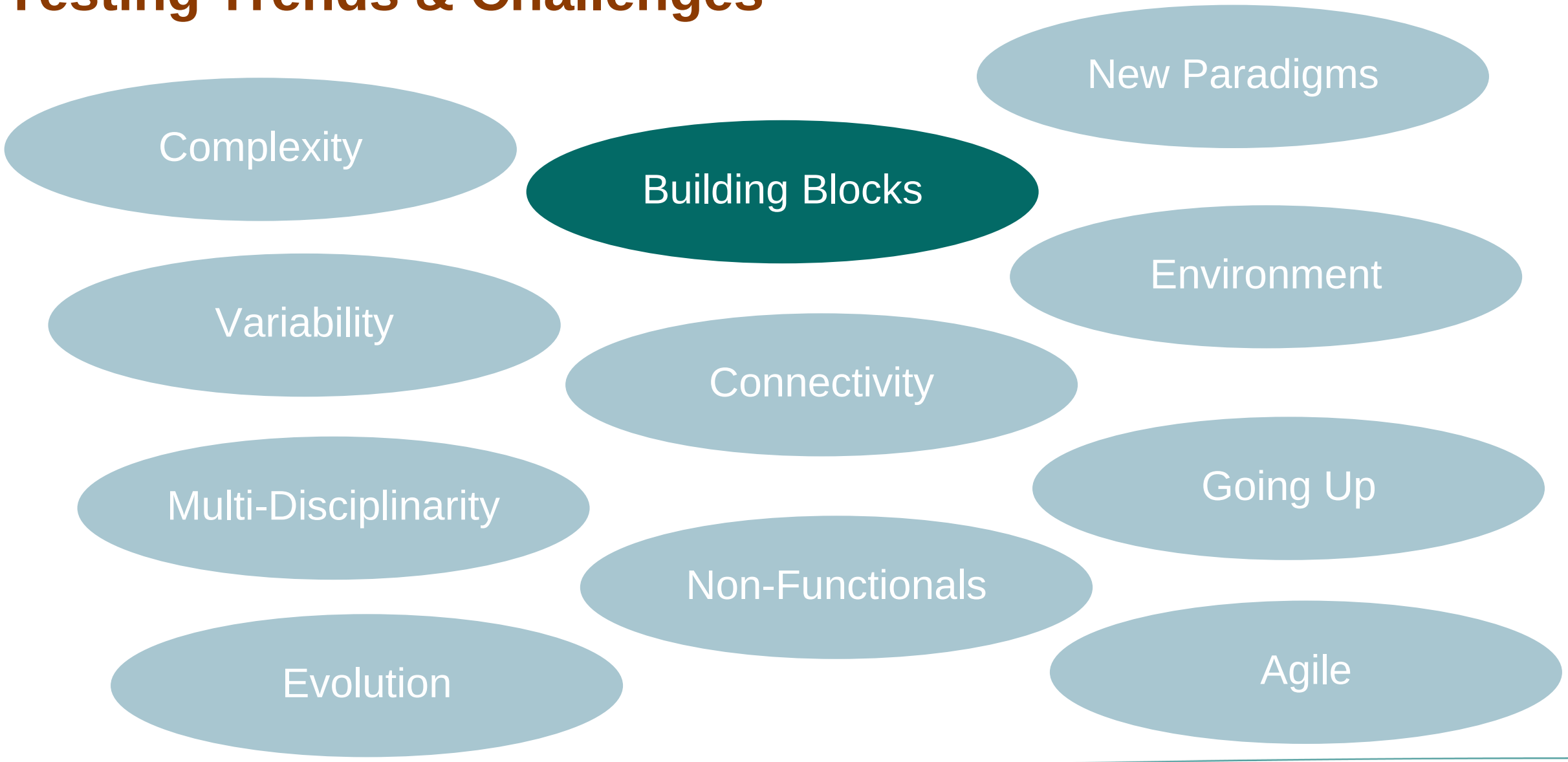


software group

Models for Multi-disciplinary Testing



Testing Trends & Challenges



Building Blocks : Components

Compon

TO REUSE, OR NOT TO REUSE

open source

IN PARTICULAR,
IN TIMES OF CONTINOUS CHANGES

Off The Shelf

ON & OFF SHORE

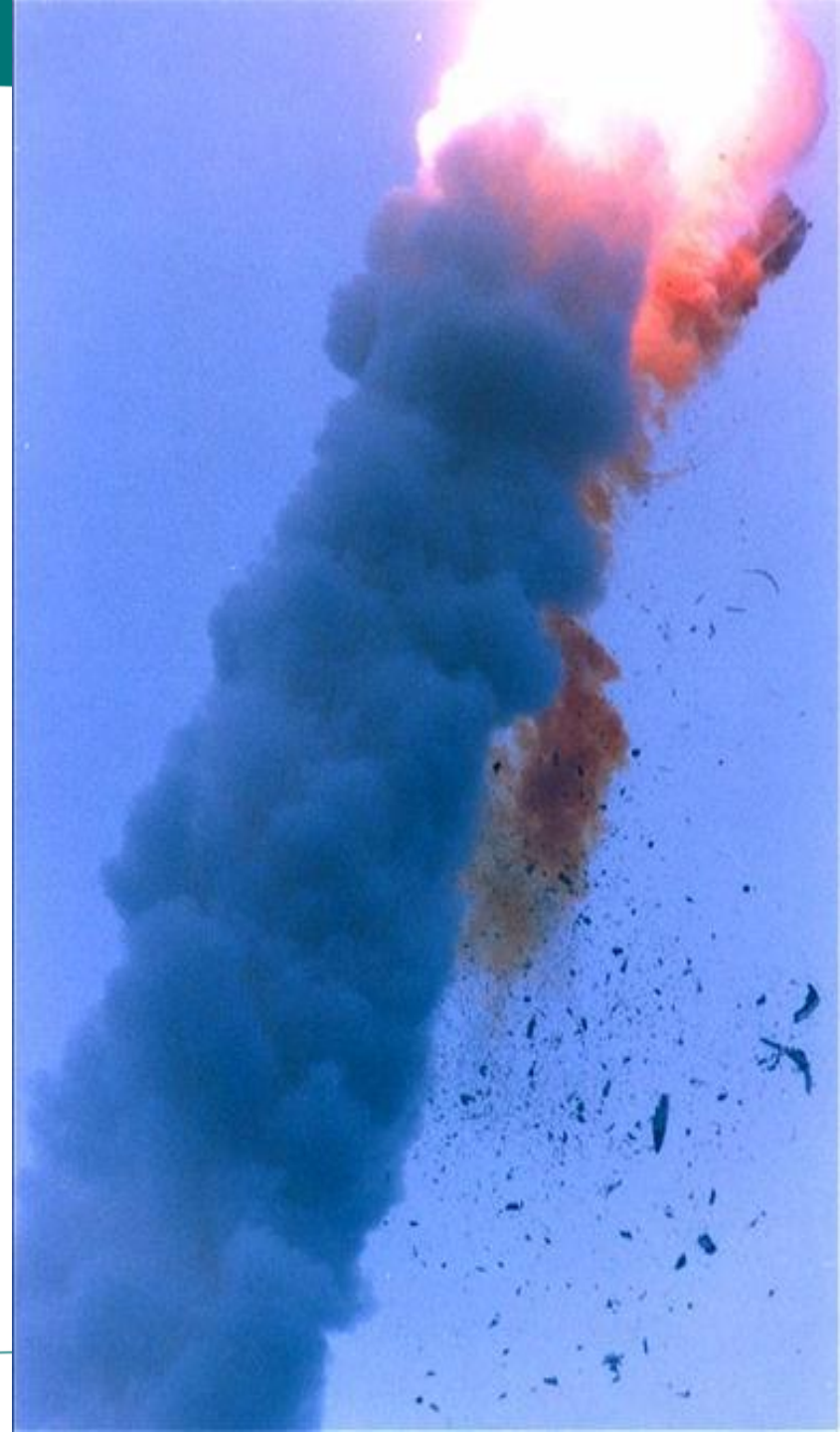
- reuse
- platform
- integration challenges
- dependencies
- when to test
- where to diagnose, repair

Components and Failures

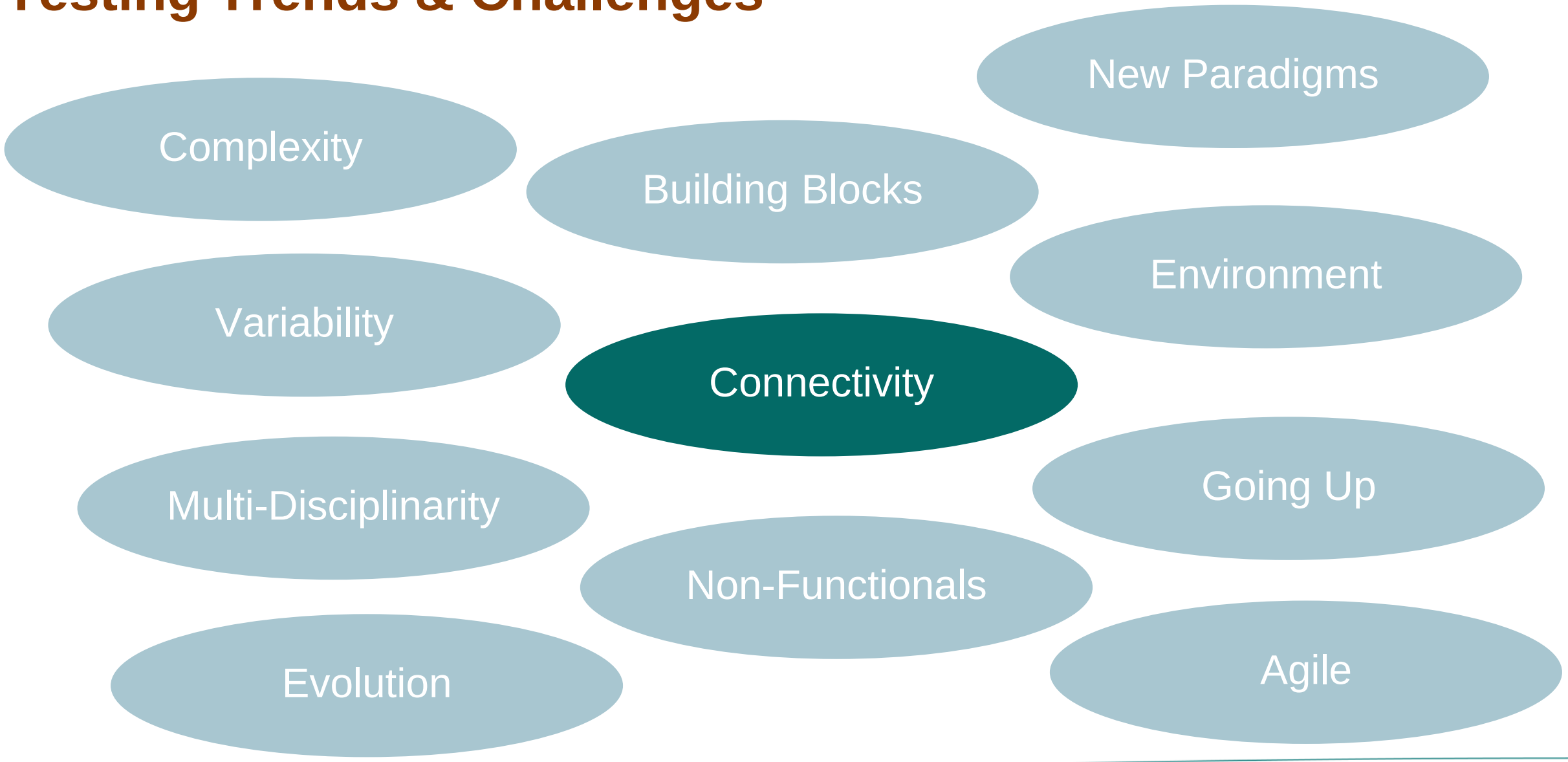
Ariane V rocket
- Design defects in control software

- Design
  - Exception handler assumed hardware errors only
  - Reuse of Ariane IV component in Ariane V without proper system testing

- Error
  - Software exception
- Failure
  - Mis-interpretation of diagnostic information

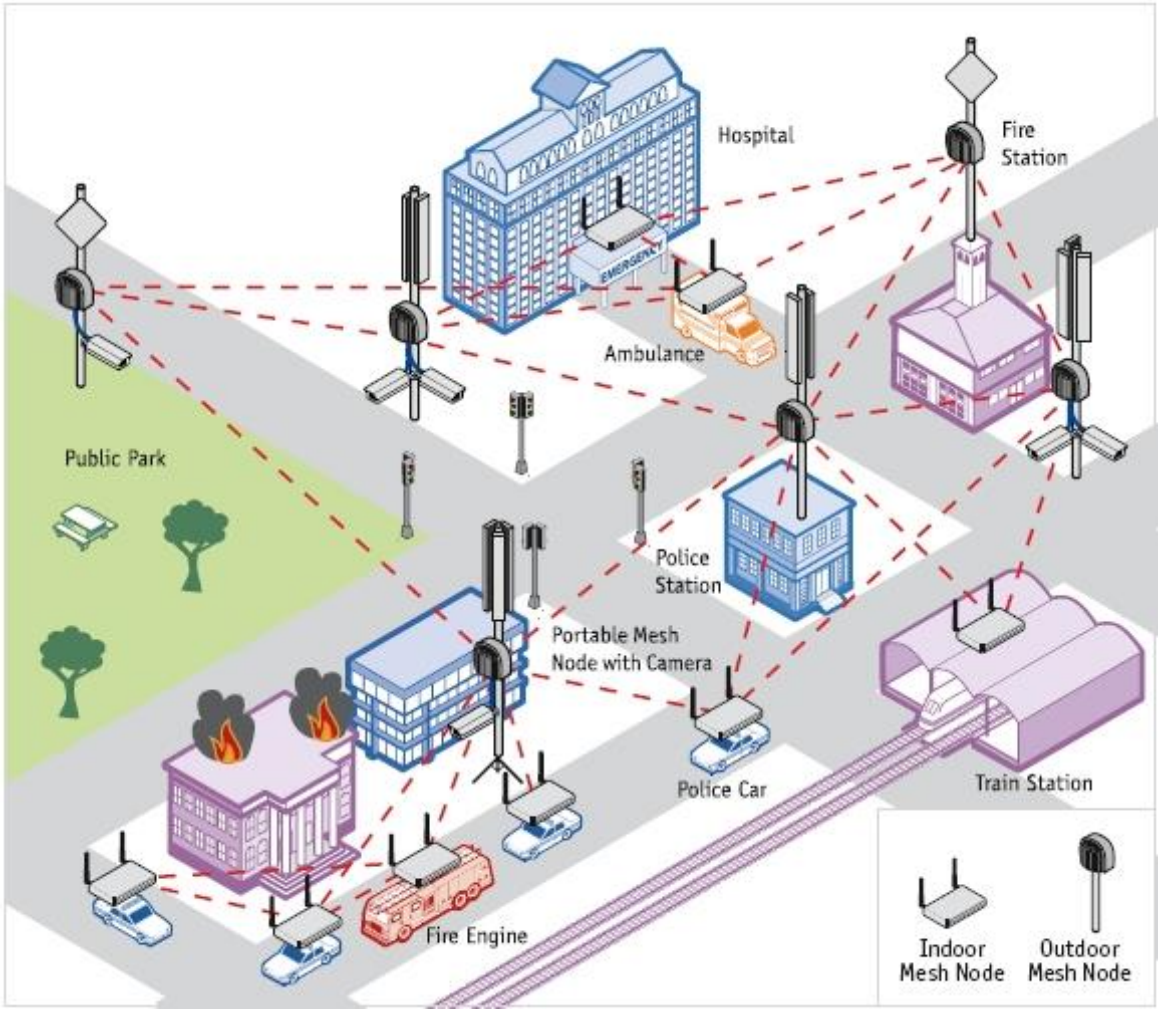


Testing Trends & Challenges



Connectivity

- Blurring boundaries of systems
  → everything connected
- Systems-Of-Systems
  - Dynamically connected systems
  - Not under own control
- Software is glue
  - with internal and external world
- Testing:
  - what is SUT ?
- Virtualization
  - which systems are available for testing ?
  - which systems must be virtualized?
- Dynamics
  - run-time testing and integration



Fading Boundaries

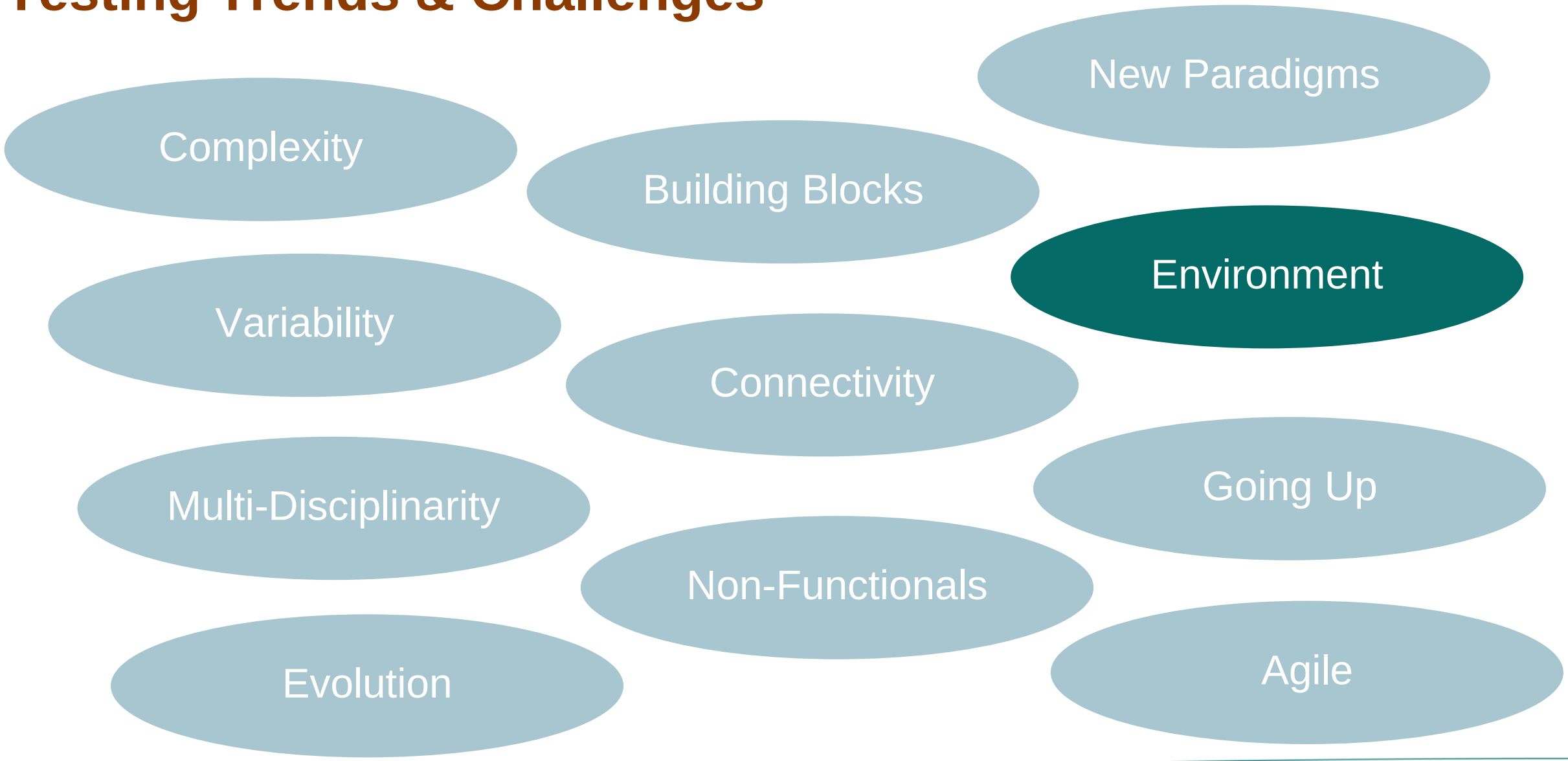
ENVIRONMENT

SYSTEM

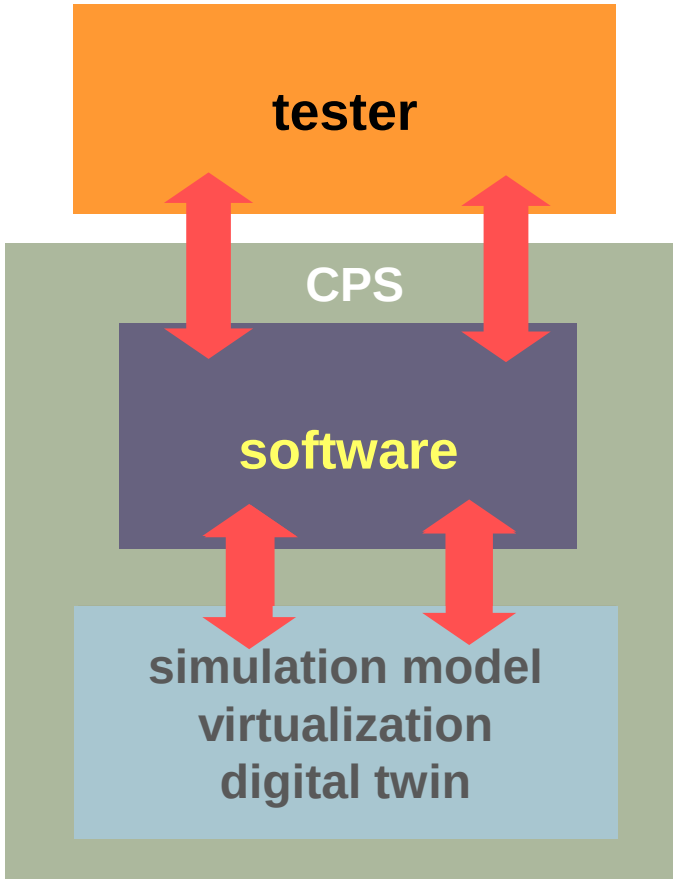
OF

SYSTEMS

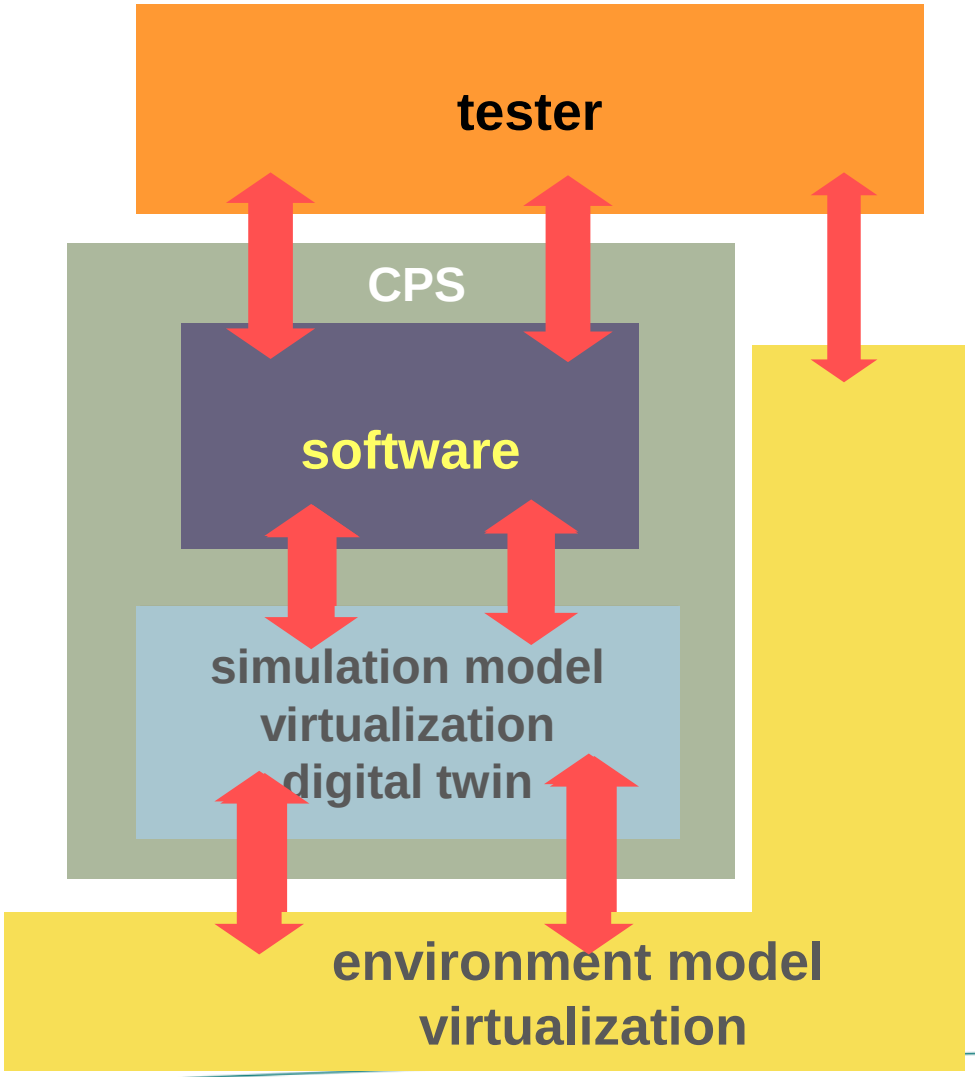
Testing Trends & Challenges



Environment

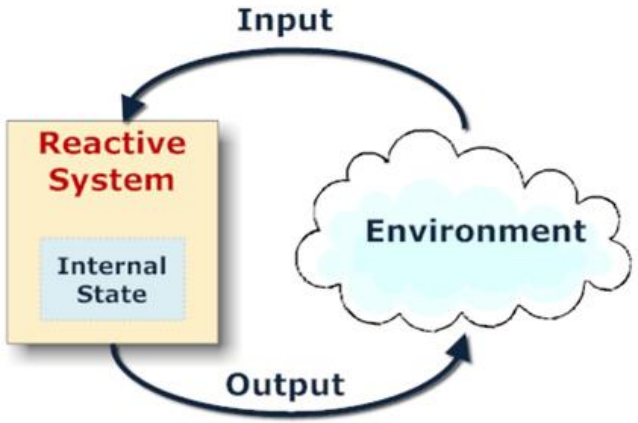


Environment



Environment

functional



state-based

autonomous

computation : *I* → *O*

reactive : *I, S* → *O, S'*

autonomous : *I, S, E* → *O, S', E'*

tests over *I*

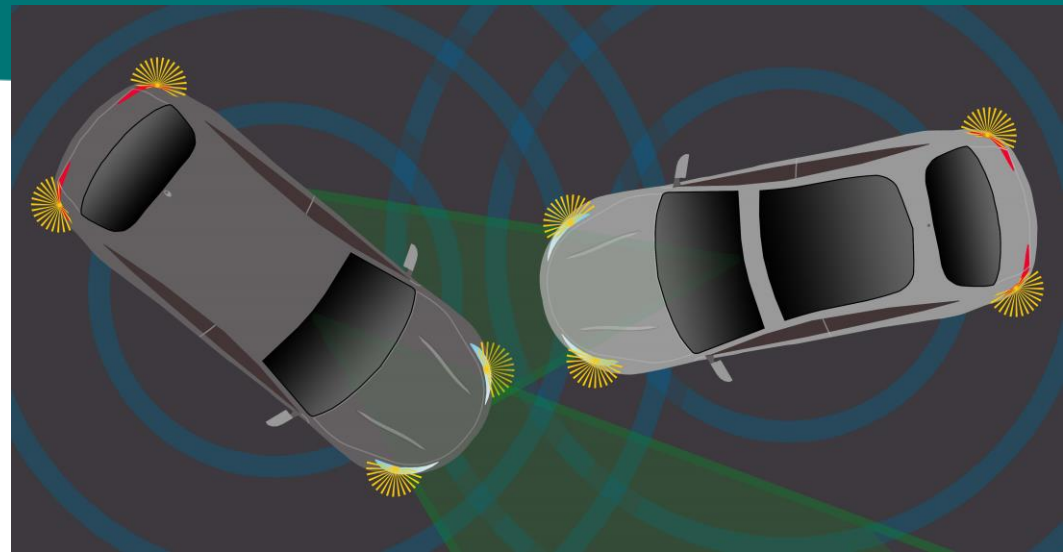
tests over *I, S*

tests over *I, S, E*

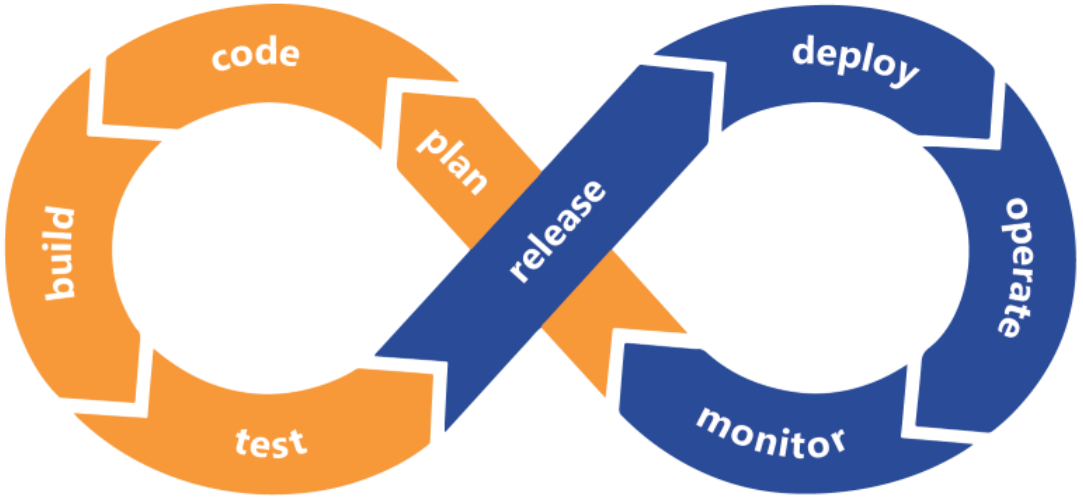
for *safety*, *trustworthiness*, *dependability,*
the **environment** must be taken into account

who determines *I* ?

Environment

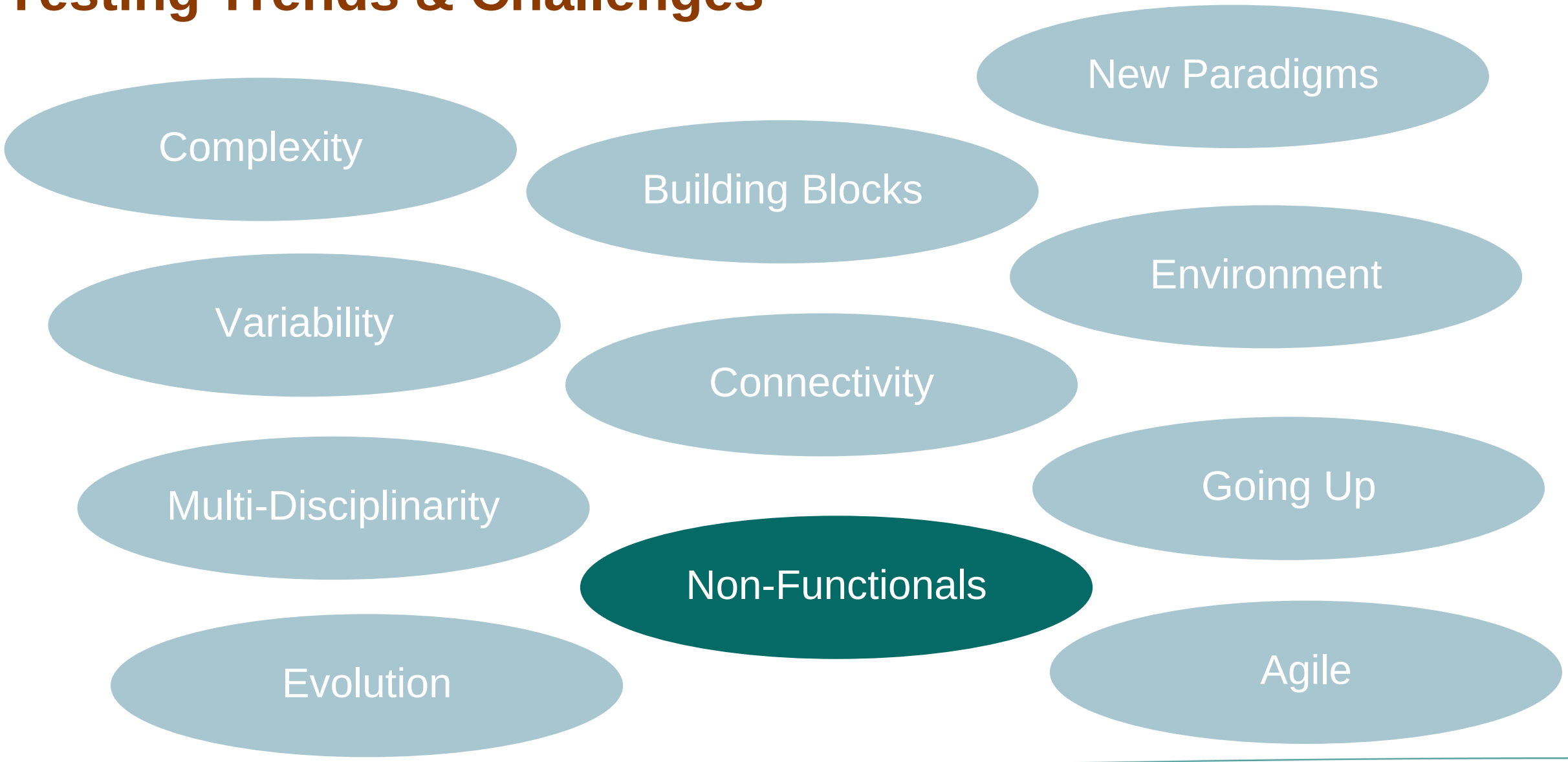


- Autonomous
  - → take part in environment
- Environment
  - → test in all possible environments
  - → not, or limited, under (test) control
  - → can change
- No specification of car as reactive system in isolation
  - → specification of car in environment : *safety*

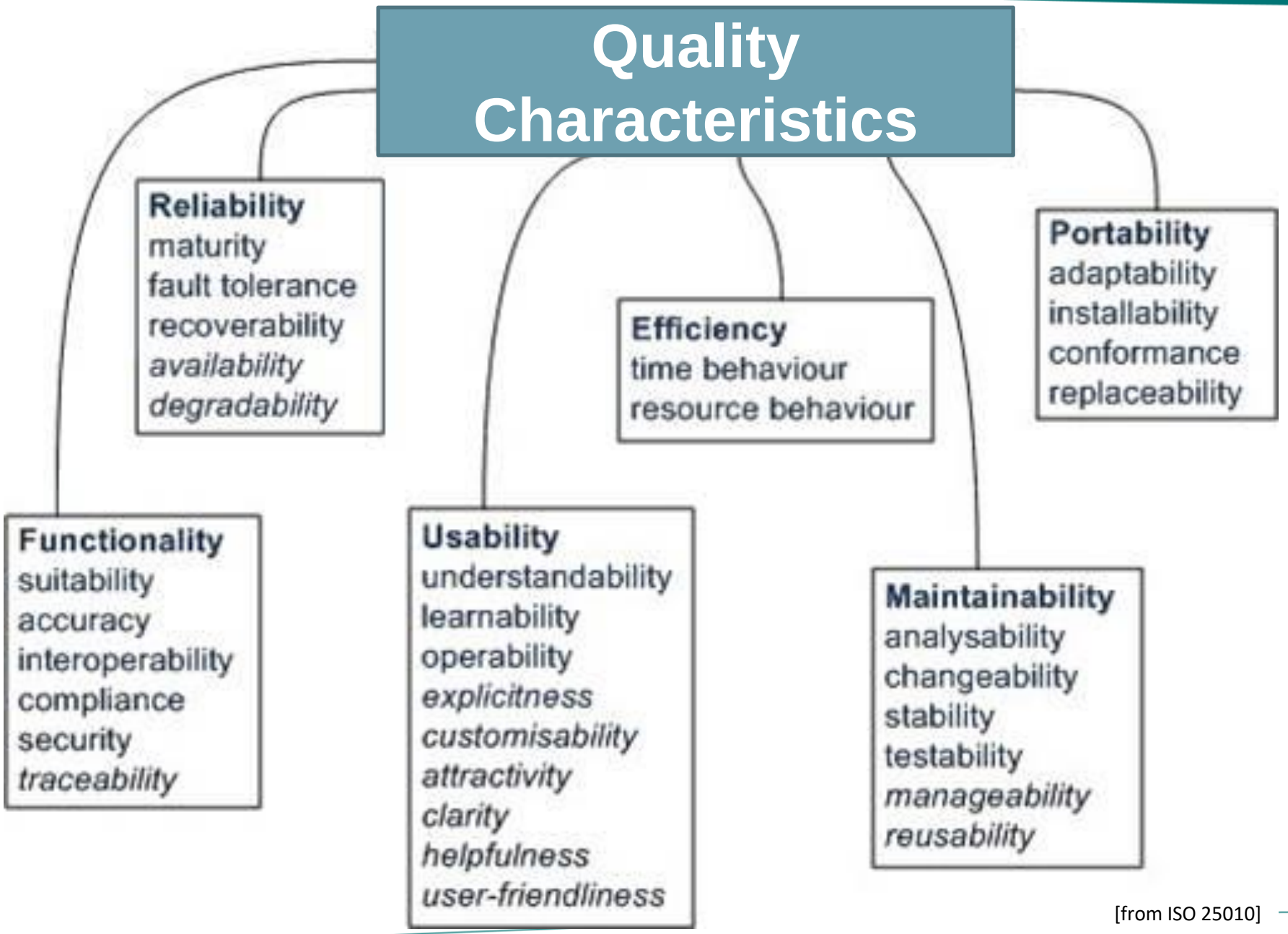


Testing everything before release is an illusion
→ continue quality control after release

Testing Trends & Challenges

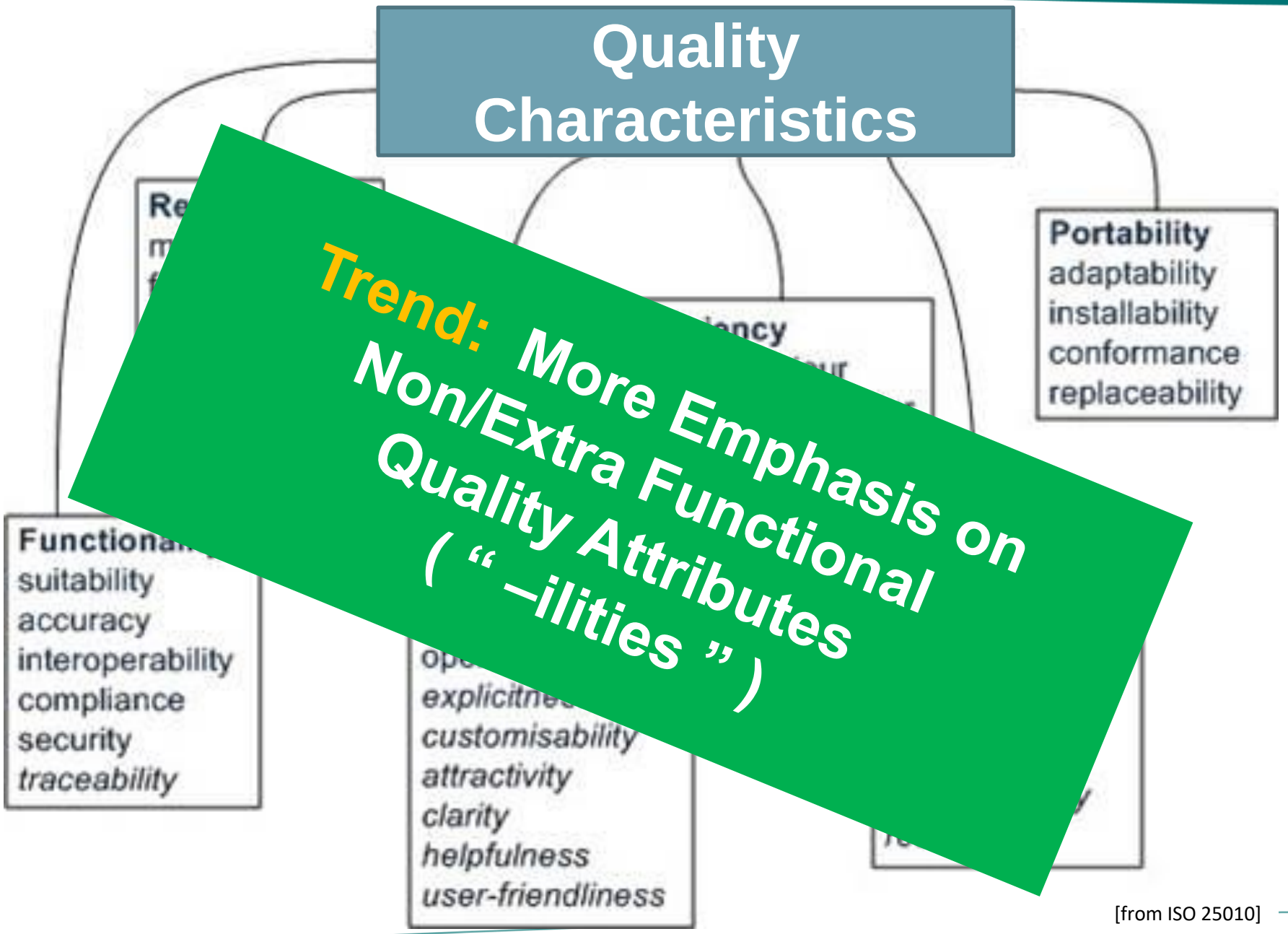


Quality Characteristics



[from ISO 25010]

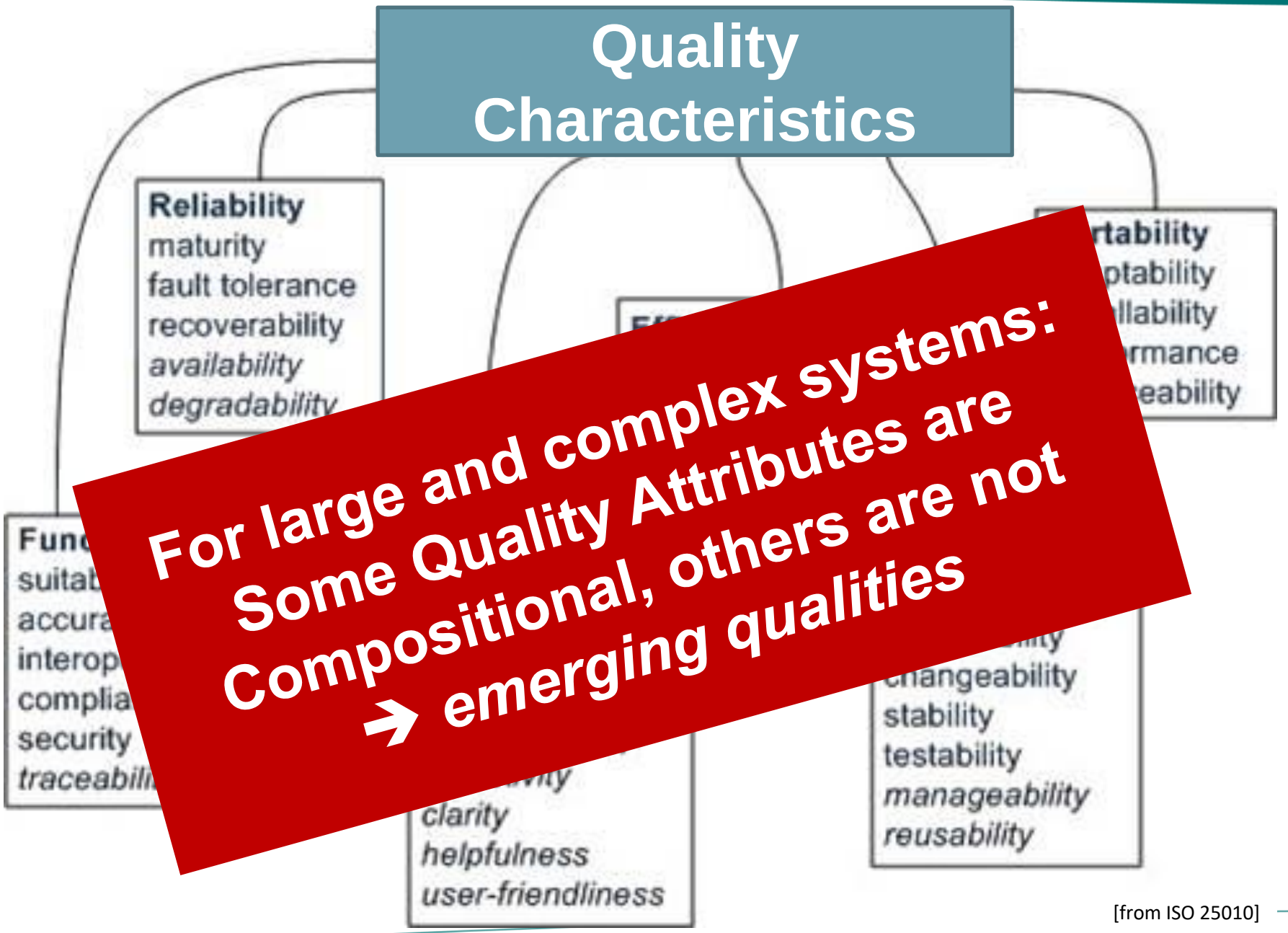
Quality Characteristics



Trend: More Emphasis on Non/Extra Functional Quality Attributes (" –ilities ")

[from ISO 25010]

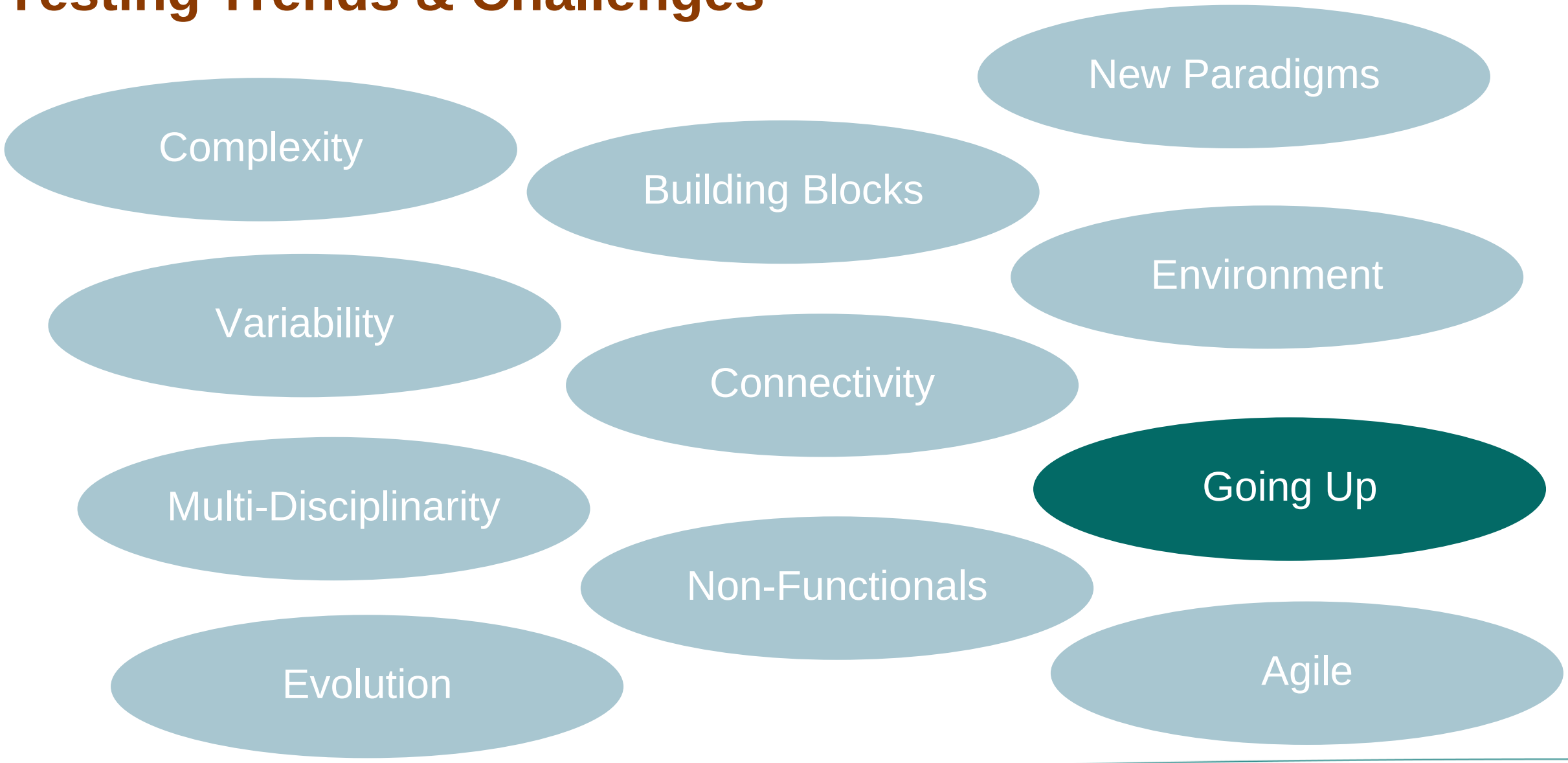
Quality Characteristics



For large and complex systems: Some Quality Attributes are Compositional, others are not ➔ emerging qualities

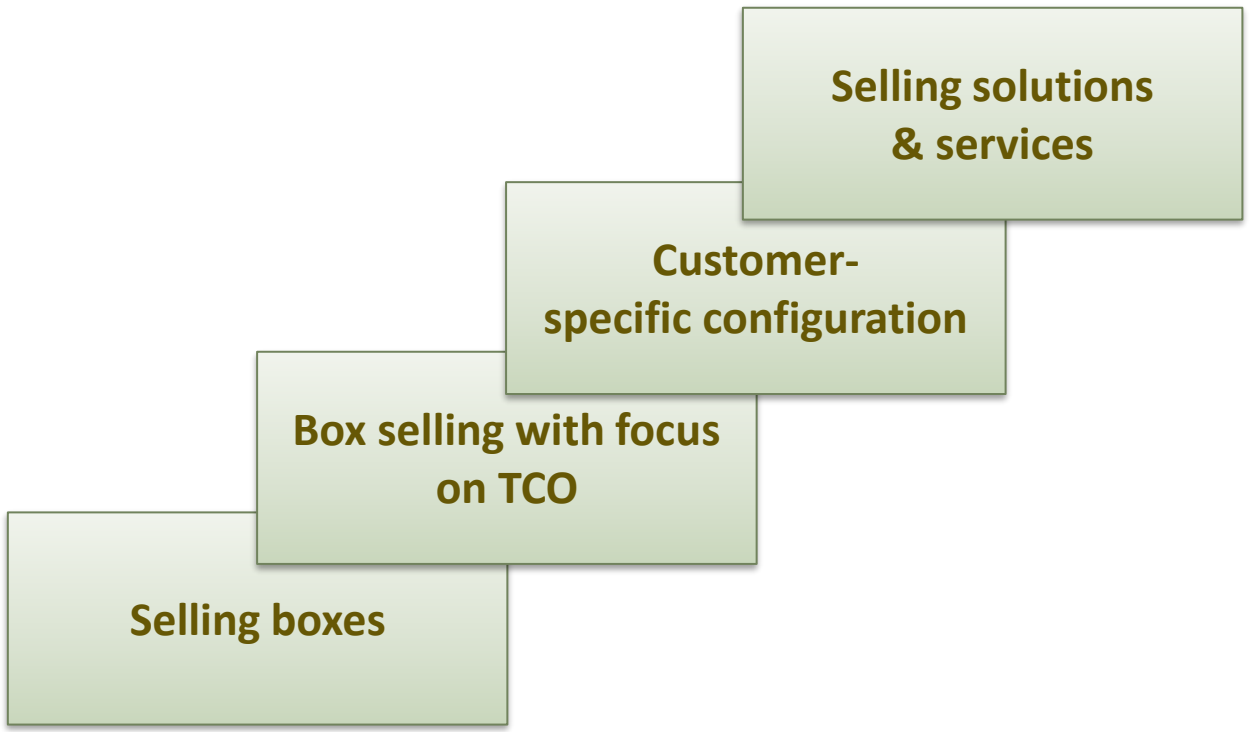
[from ISO 25010]

Testing Trends & Challenges

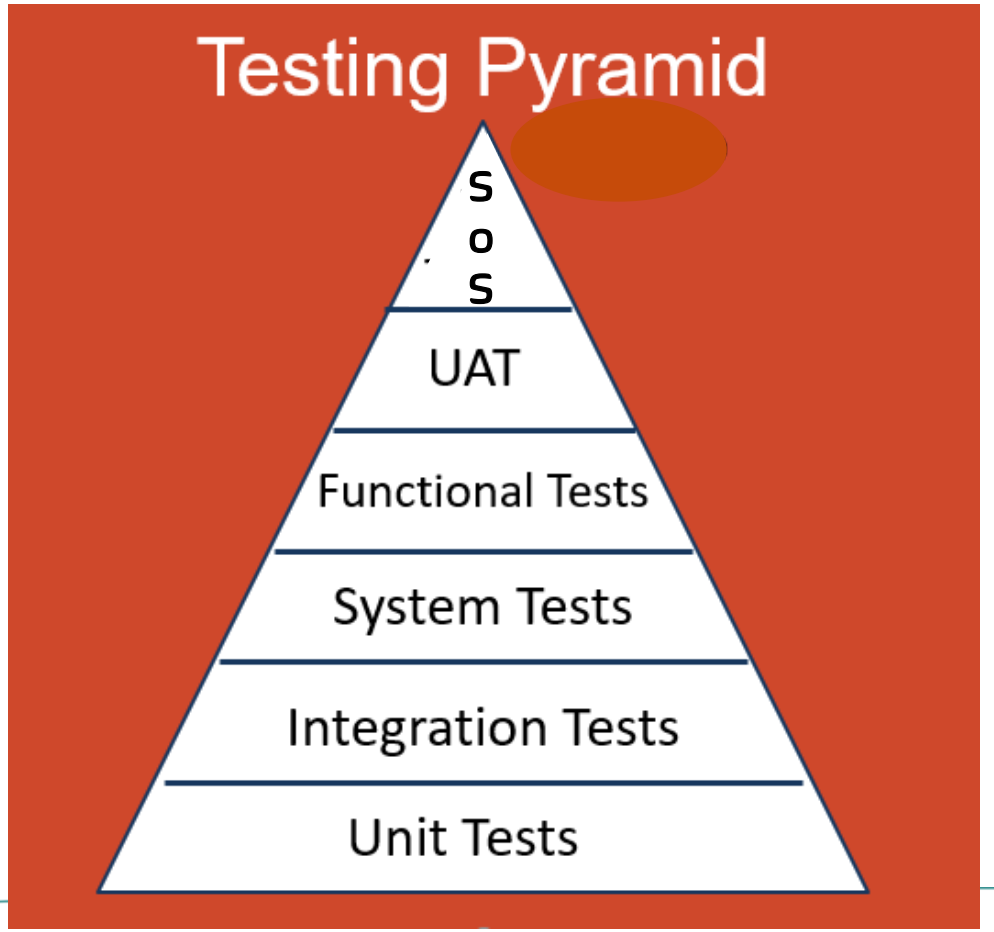


Going Up

- In the Value Chain
  - new business models
  - testing quality-of-service



- In the Test Pyramid
  - everybody does unit tests
  - bugs are on the higher levels



Going Up

- In Coding
  - from software to meta-software: build tools, build scripts, configuration setting, . . .

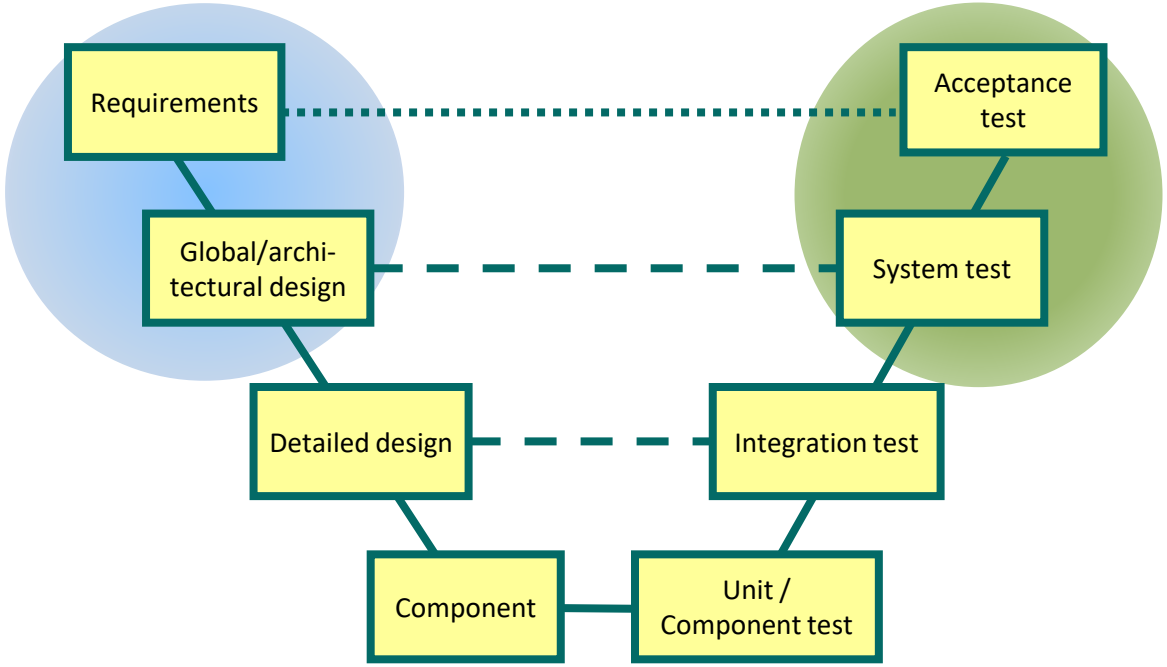
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="htt
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- In the V-Model
  - requirements, design, system test
  - detailed design, coding, unit tests outsourced

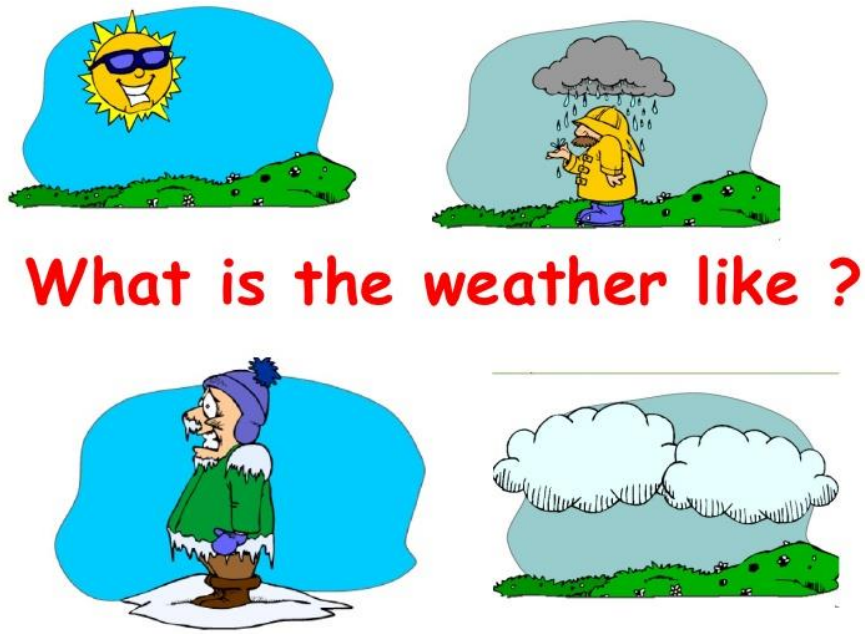


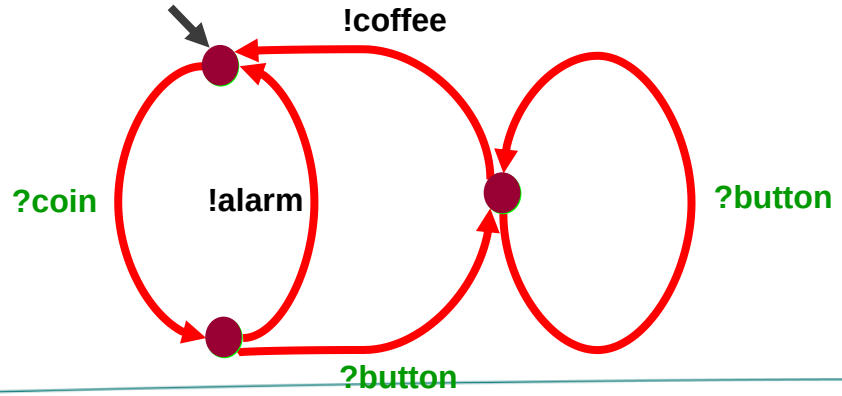
Going Up Consequence : Uncertainty & Non-Determinism

- Sometimes you don't know …..
  - testing a search engine, weather forecast, …
  - systems-of-systems, big data, ...
- Sometimes you don't want to know …..
  - no details
  - abstraction
  - particular view

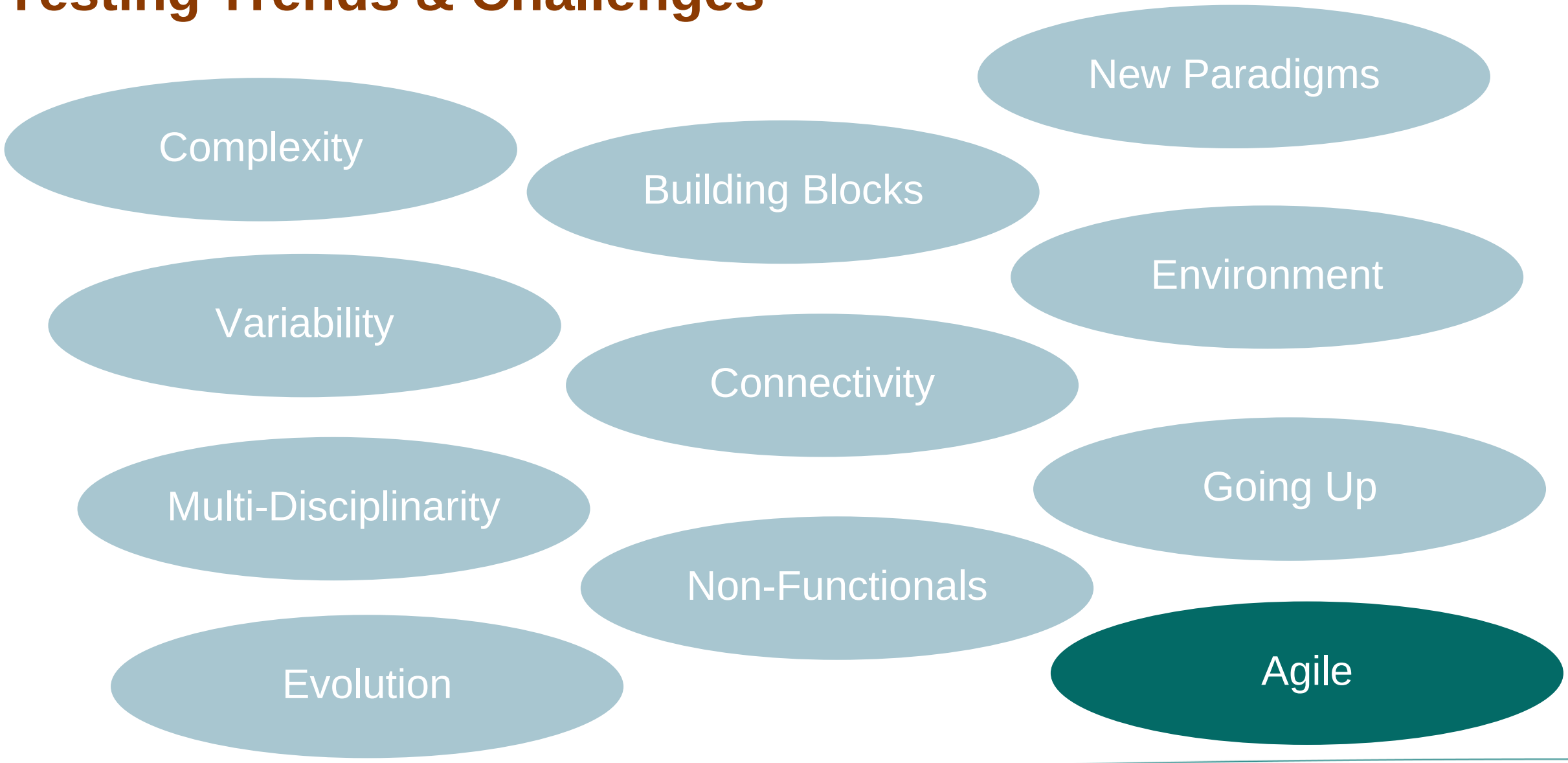
Uncertainty of test outcomes & oracles

- **non-determinism**
- **probabilities**





Testing Trends & Challenges



Agile ?


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com scottadams@aol.com
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

Agile

- Agile → test automation
  - test execution automation
  - test fast and often

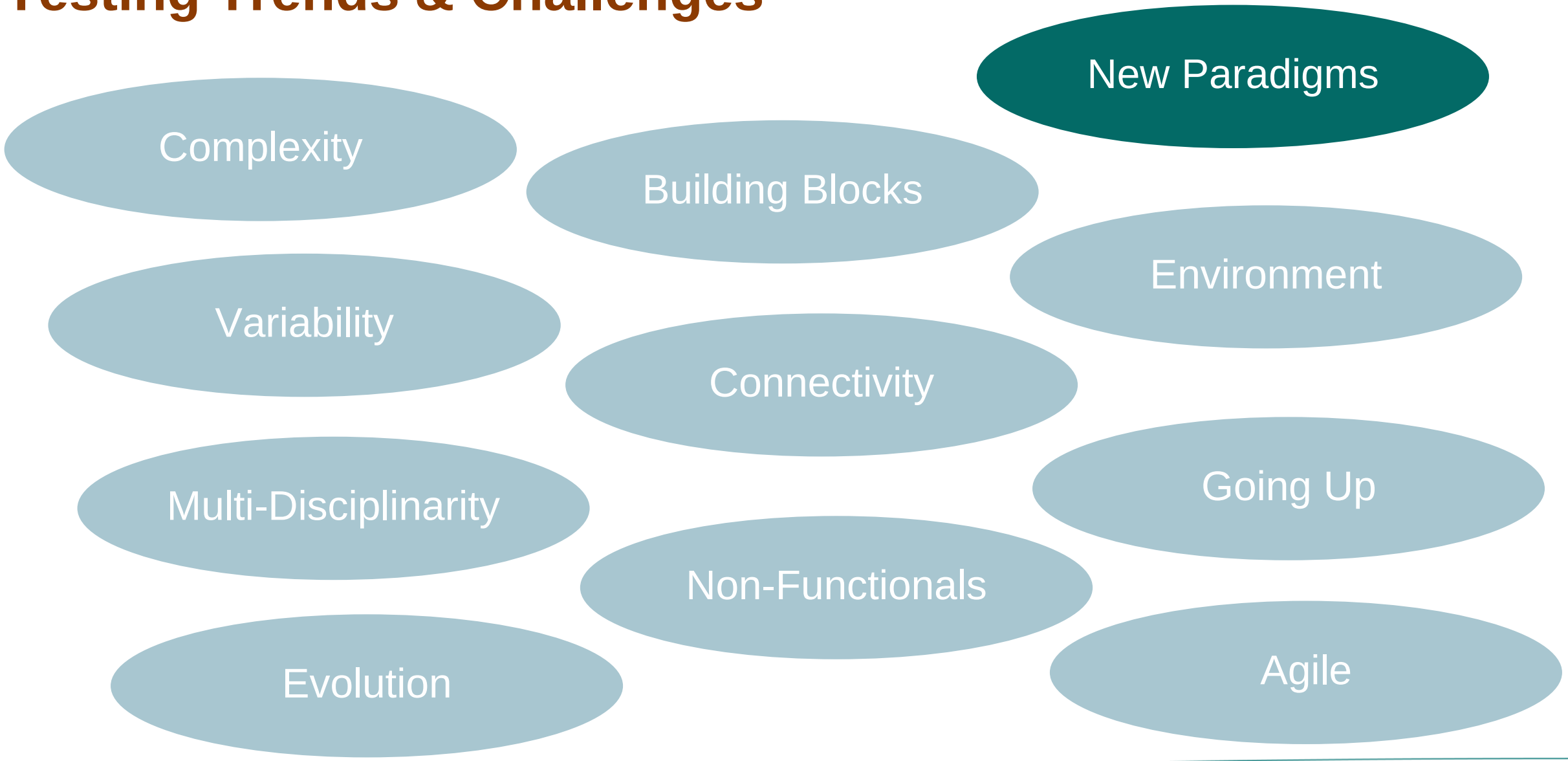
→ Large repositories of scripted tests

  - the night is too short
  - traceability to requirements ?
  - maintainability ?
  - pesticide paradox : *how to increase variation in tests ?*

Agile - fallacy of complete specification:

We finally have the guts to admit that we don't know precisely what the system should do when we start coding.

Testing Trends & Challenges



New Paradigms and Technologies

- Cloud
- Mircroservices
- Autonomous, self-adaptive systems
- AI, Machine Learning
- Quantum Computing
- Ethics, sustainability, …
- . . . . .

Microservices Testing

Quality Characteric

- **functionality testing**
- interoperability, reliability, efficiency, security, safety, portability, availability, ...

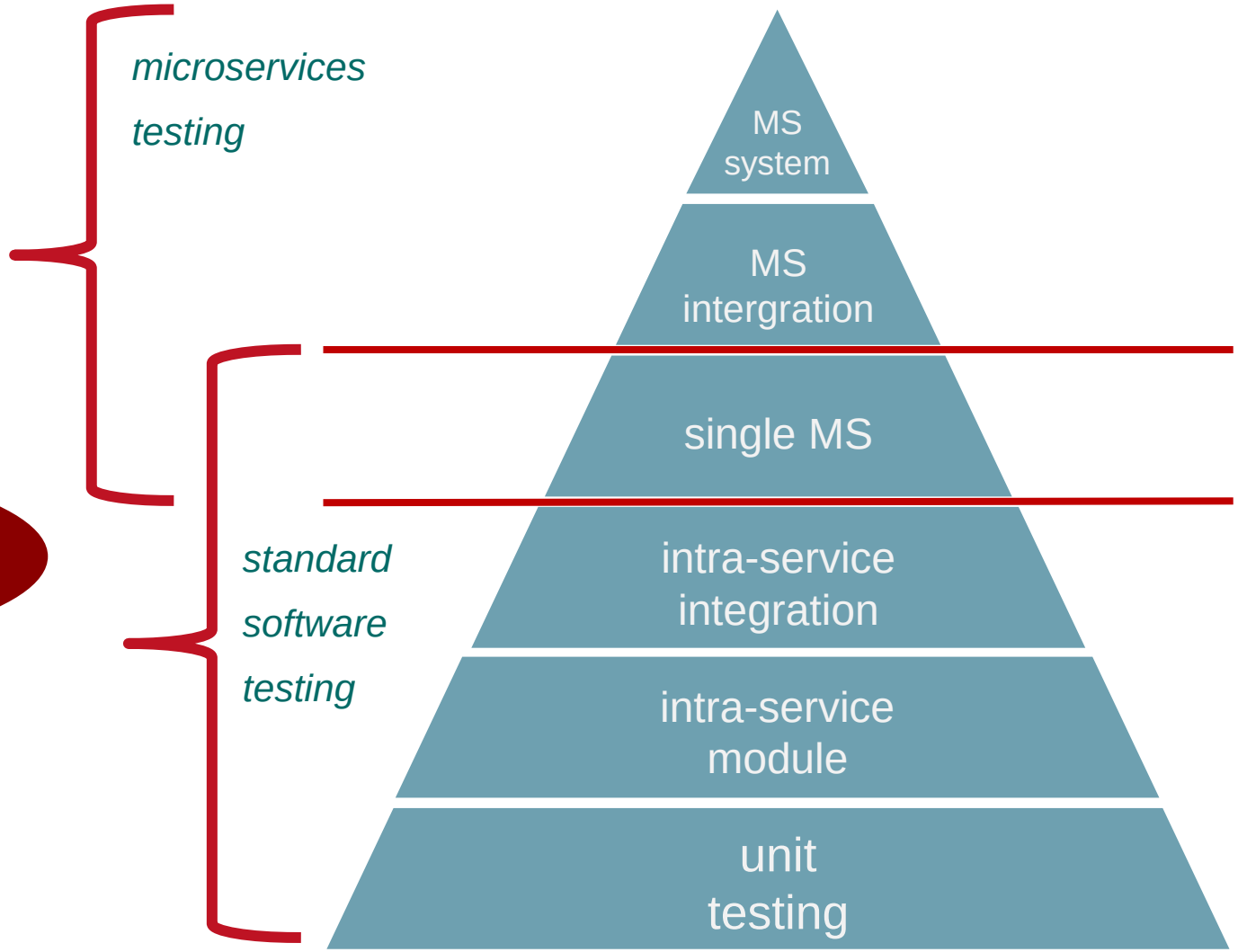
Actors

- **developer/independent testing**
- user, cloud

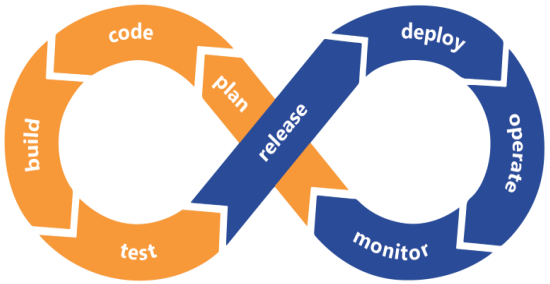
Going Up

Level of Testing

- **service system, end-to-end**
- **service integration**
- **black-box single service**



Microservices Testing Challenges



'Every advantage has its disadvantage':

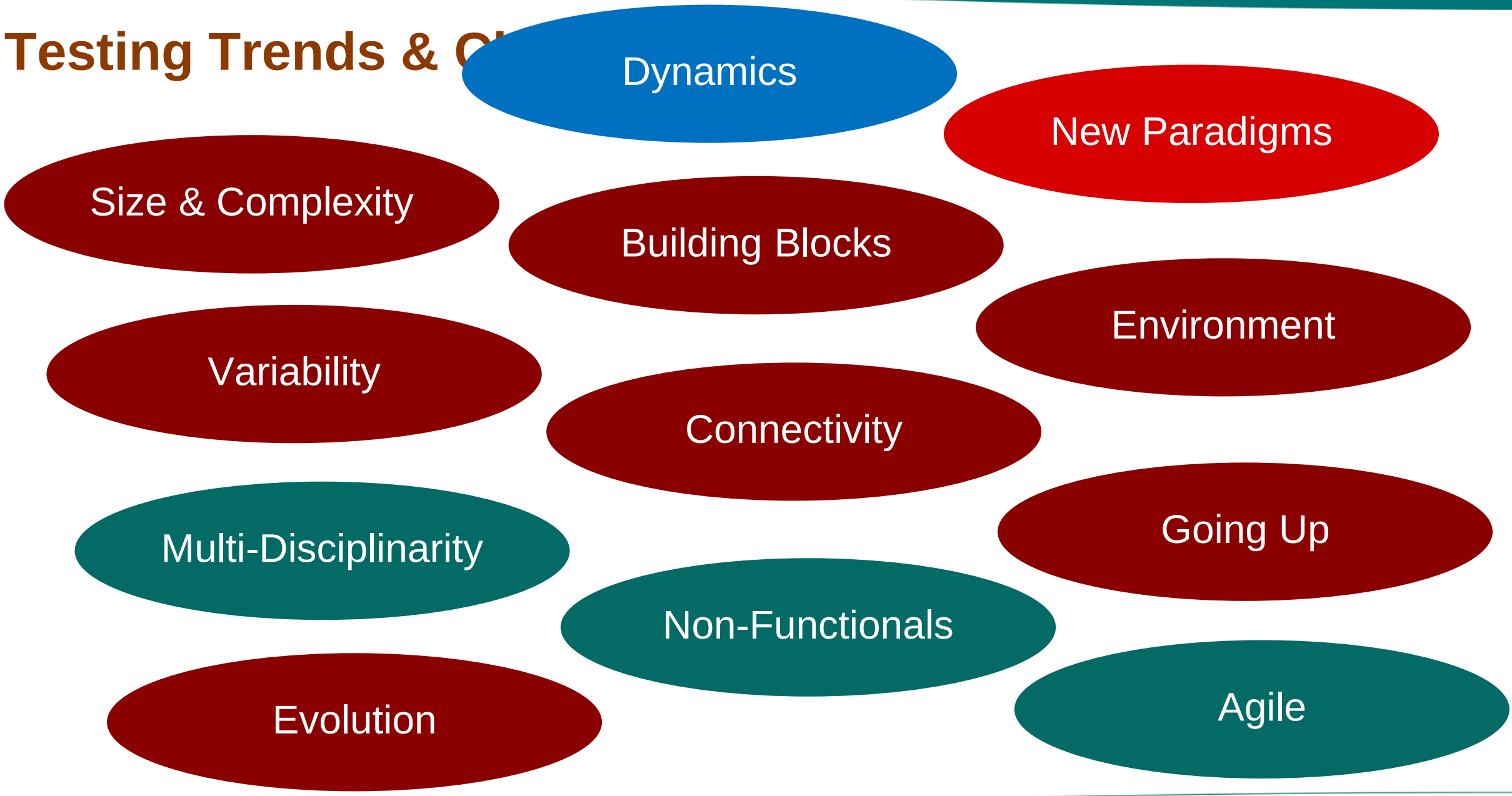
Testing MS more complex than testing monoliths

1. *SUT*
2. *complete system of MS (SoS) is big*
3. *MS come with a lot of additional/context/helper/platform systems*
4. *distribution*
5. *configurability*
6. *test-data generation*
7. *dynamics*
8. *bug analysis*

State of Practice

- Many tools that log, track, monitor, or measure
  - *passive testing*
- API testing (based on standards – OpenAPI, Swagger, . . .)
- contract testing
- not that many tools for active testing
- *unit testing* and *end-to-end testing* often done; less often in between

Testing Trends & Ch



Testing Trends & Challenges

New Paradigms

Complexity

Building Blocks

Environment

Variability

Connectivity

Multi-Disciplinarity

Non-Functionals

Evolution