

Testing Challenges for Cyber Physical Systems



Jan Tretmans

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

CPS : Cyber Physical Systems

Cyber-Physical Systems



Semiconductor manufacturing equipment



Traffic management



Combat management systems



Industrial printers



Automotive



Agricultural robots



Robotized warehousing



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**

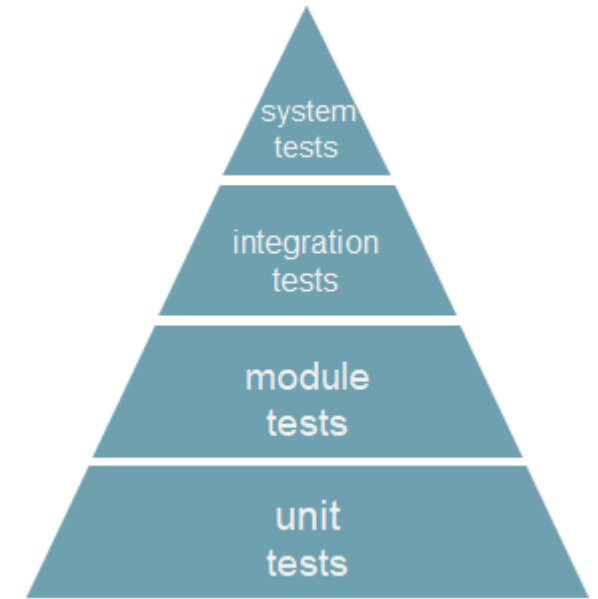
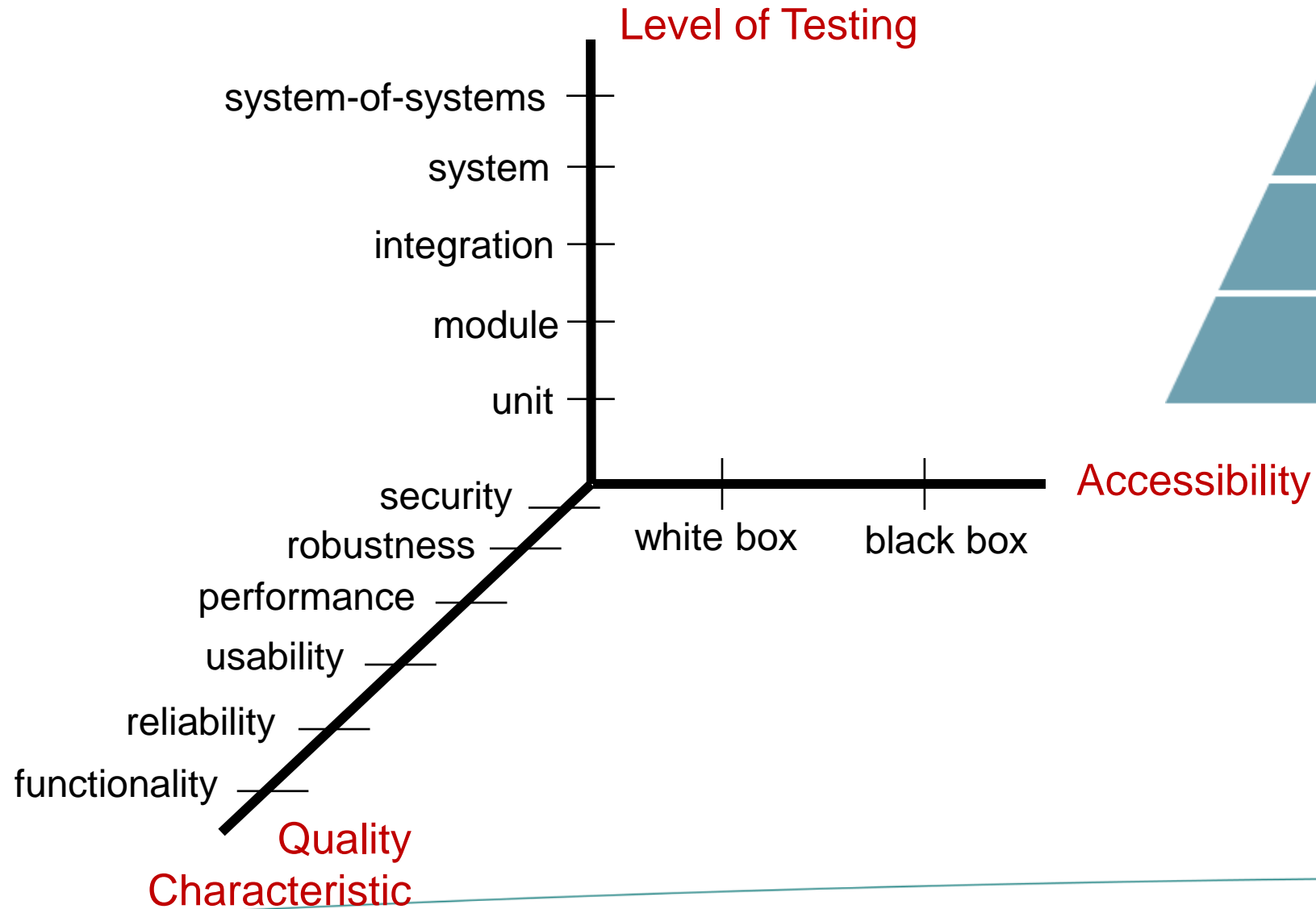
Software Testing

Testing: A Definition

Software testing is:

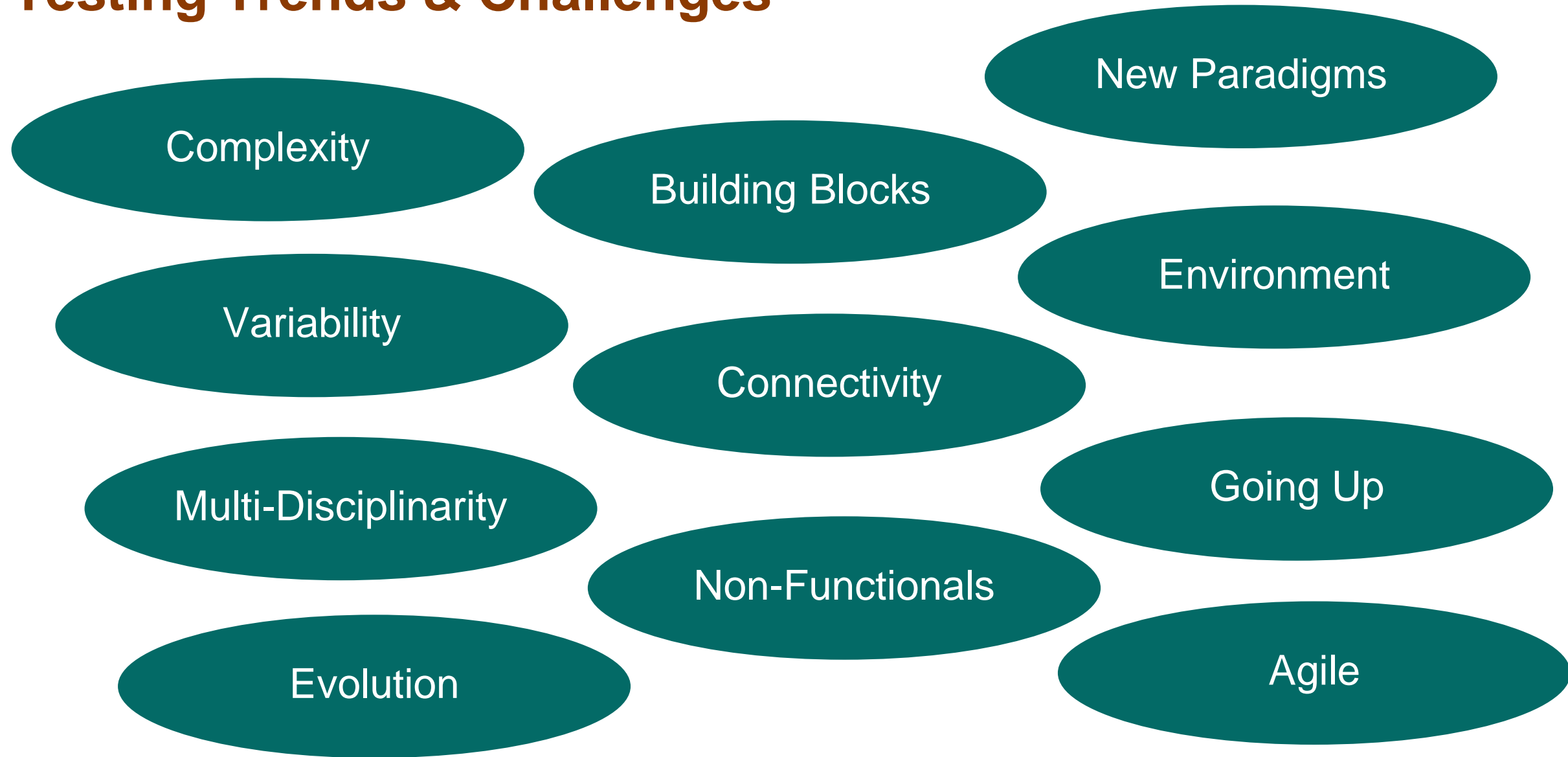
- a technical process,
- performed by executing / experimenting with a product,
- in a controlled environment, following a specified procedure,
- with the intent of measuring one or more characteristics / quality of the software product
- by demonstrating the deviation of the actual status of the product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

Sorts of Testing

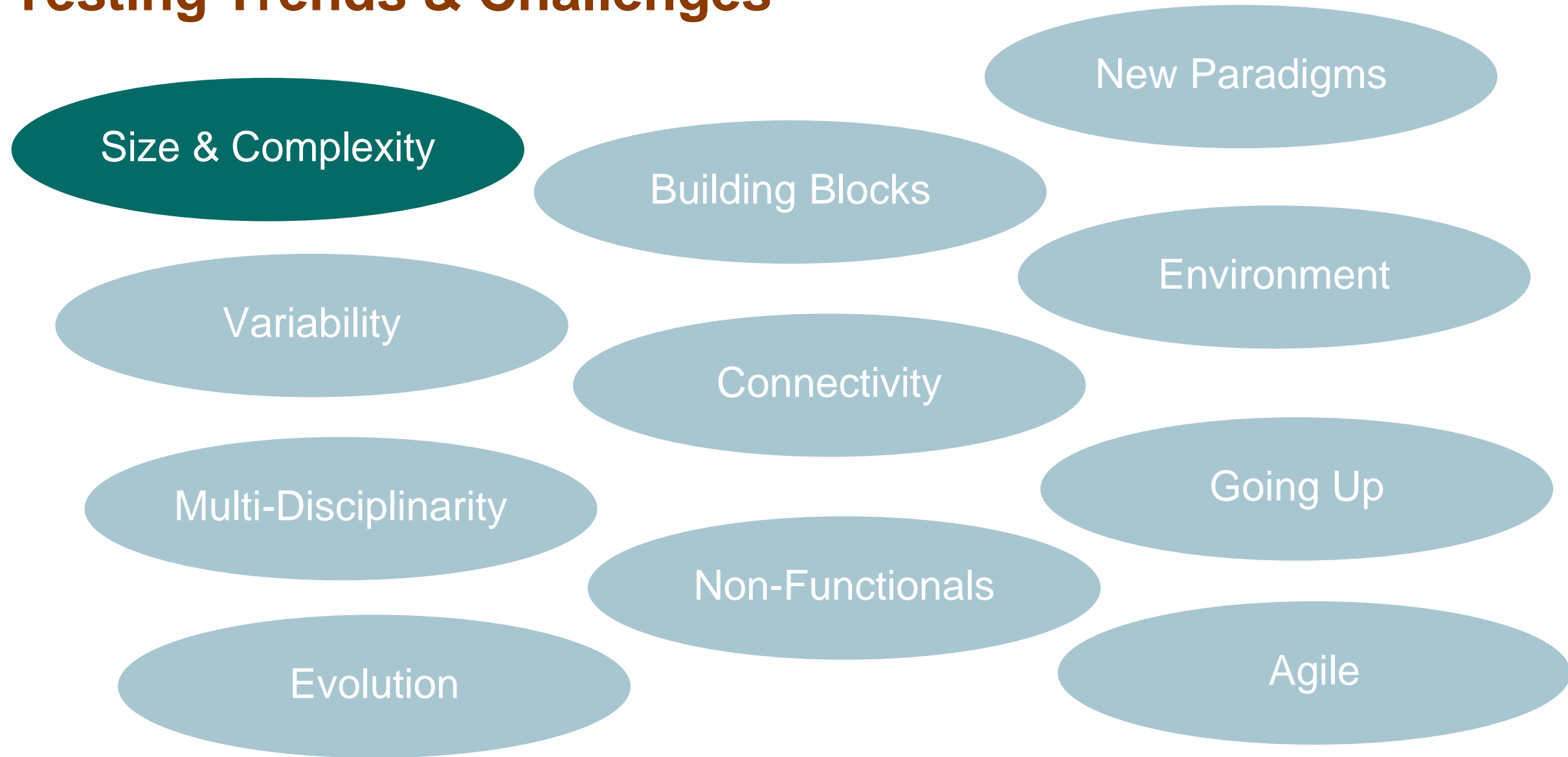


Testing Challenges for Cyber-Physical Systems

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

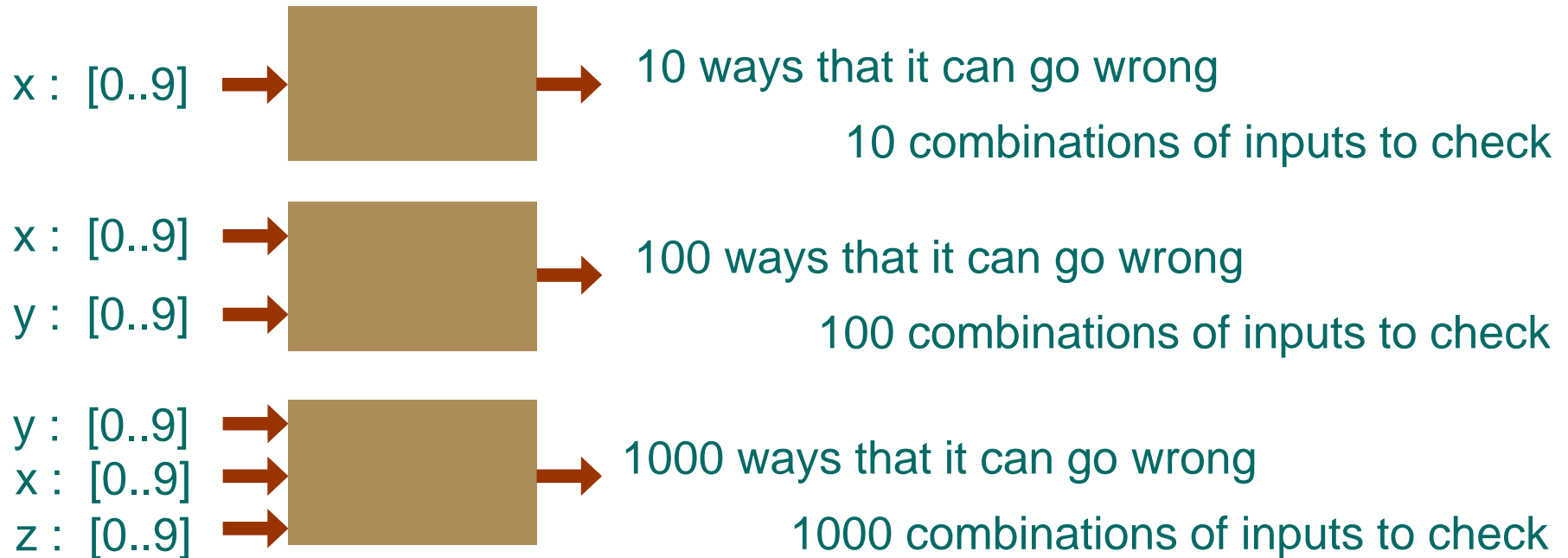
Machine with 300 parameters

- $2^{300} = 10^{90}$ 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



Size & Complexity

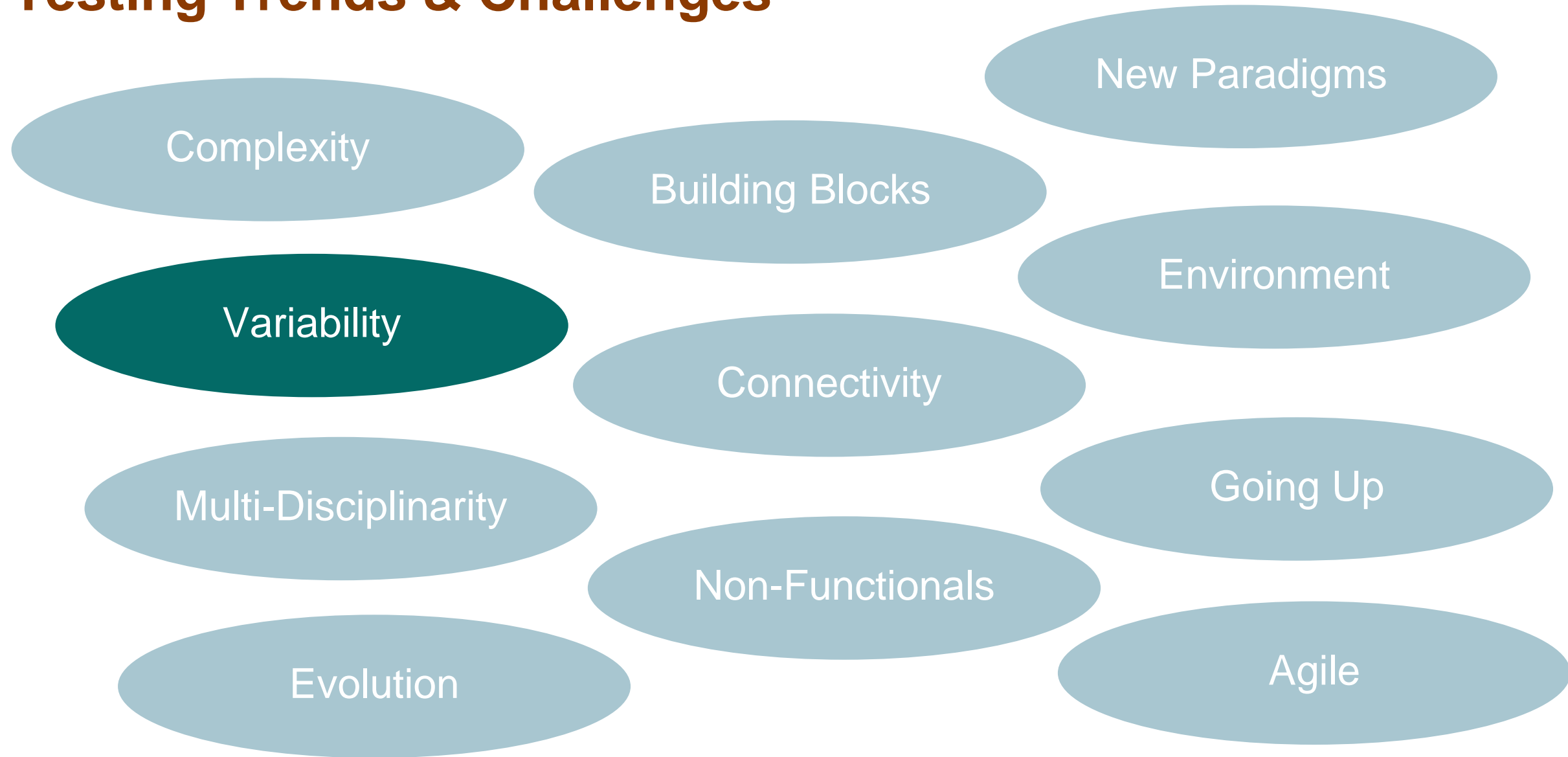
Testing effort grows exponentially with system size

Testing cannot keep pace with development



→ combinatorial explosion of required testing effort

Testing Trends & Challenges



Variability & Product Lines

or: How to Select your Sandwich



VEGETARIAN

WHICH WICH WOULD YOU LIKE?

☐ TRIPLE CHEESE MELT
☐ ELVIS WICH (Pb, Honey & Bananas)
☐ TOMATO & AVOCADO
☐ BLACK BEAN PATTY
☒ HUMMUS & BELL PEPPERS

CHOOSE YOUR BREAD

☐ WHITE ☒ WHEAT

CHOOSE YOUR CHEESE (Optional)

☐ AMERICAN ☐ SWISS ☐ PROVOLONE
☐ CHEDDAR ☒ PEPPER JACK ☐ MOZZARELLA

How Would You Like Your WICH Worked?

MUSTARDS
☐ Yellow ☐ Dijon ☐ Honey ☒ Deli

MAYOS
☐ Regular ☐ Lite ☐ Horseradish ☒ Spicy

SPREADS & SAUCES
☐ BBQ ☐ Buffalo ☐ Marinara
☐ 1000 Island ☐ Ranch

ONIONS
☒ Red ☐ Grilled ☐ Crispy Strings

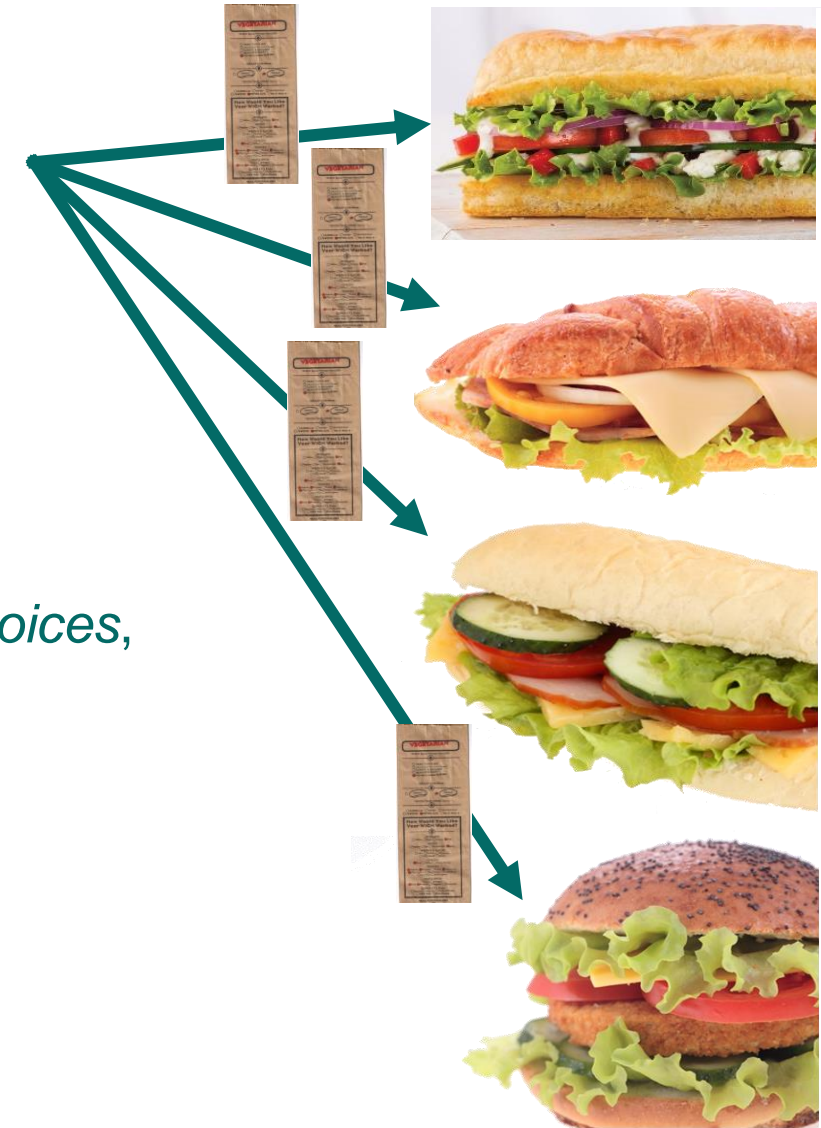
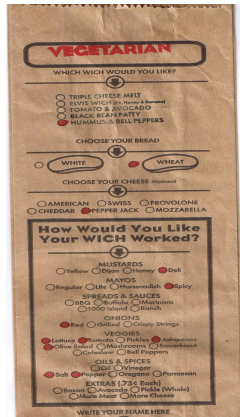
VEGGIES
☒ Lettuce ☒ Tomato ☐ Pickles ☒ Jalapenos
☒ Olive Salad ☐ Mushrooms ☐ Sauerkraut
☐ Coleslaw ☐ Bell Peppers

OILS & SPICES
☐ Oil ☐ Vinegar
☒ Salt ☒ Pepper ☐ Oregano ☐ Parmesan

EXTRAS (.75¢ Each)
☐ Bacon ☐ Avocado ☐ Pickle (Whole)
☐ More Meat ☐ More Cheese

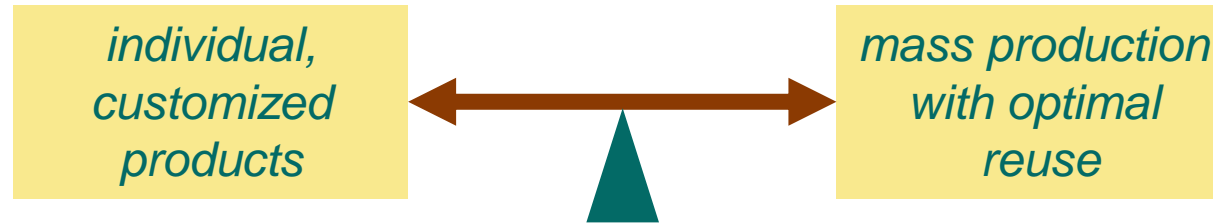
WRITE YOUR NAME HERE

Variability



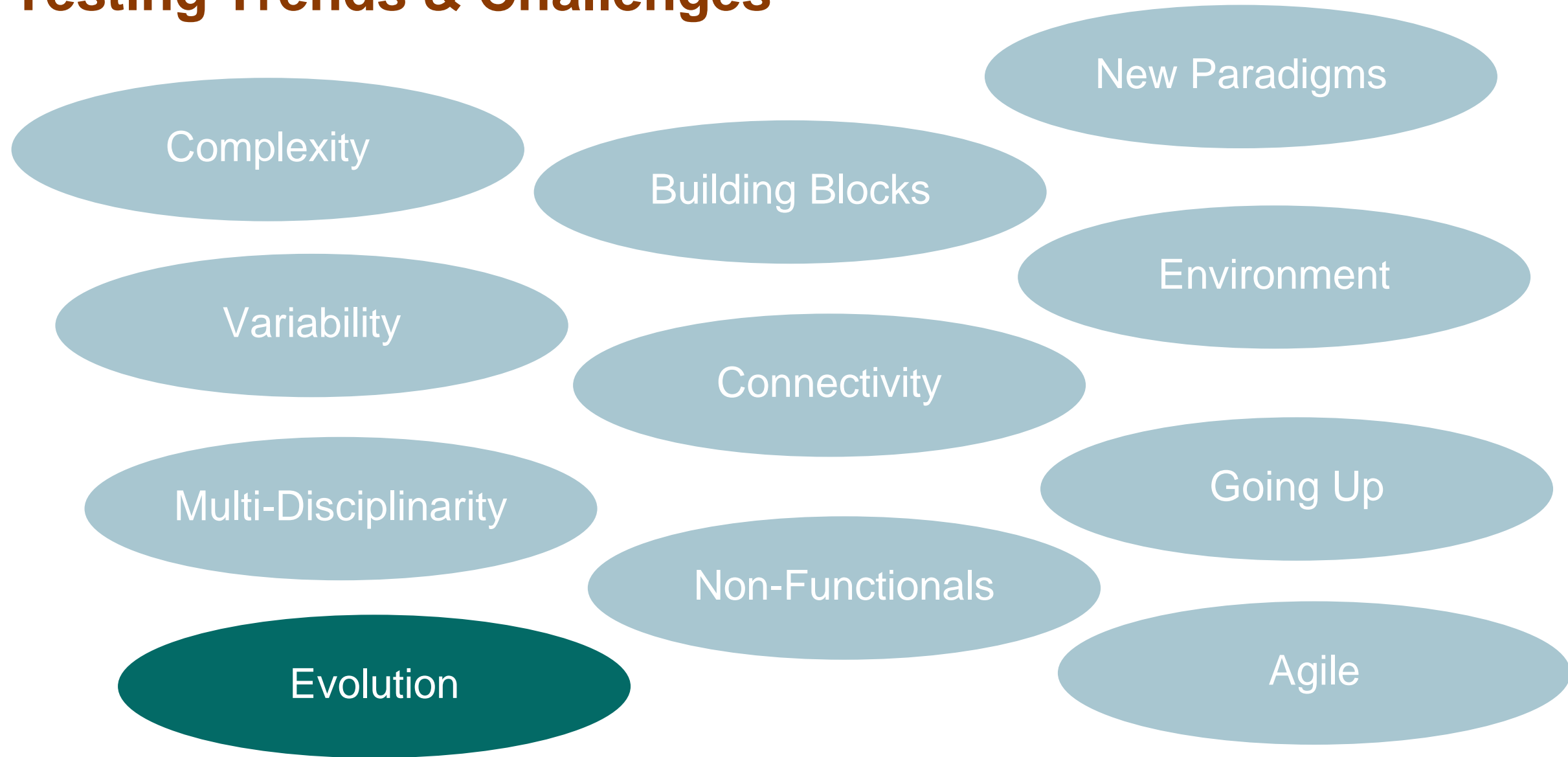
- Highly configurable sandwich: *exponential number of choices*, **a different sandwich for everybody on the planet!**
- Not all combinations make sense: *dependencies*
- **How to taste / test all of them?**
- Sandwich product line = family of sandwiches
- Also for high-tech systems
Linux, cars, . .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

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 ?**

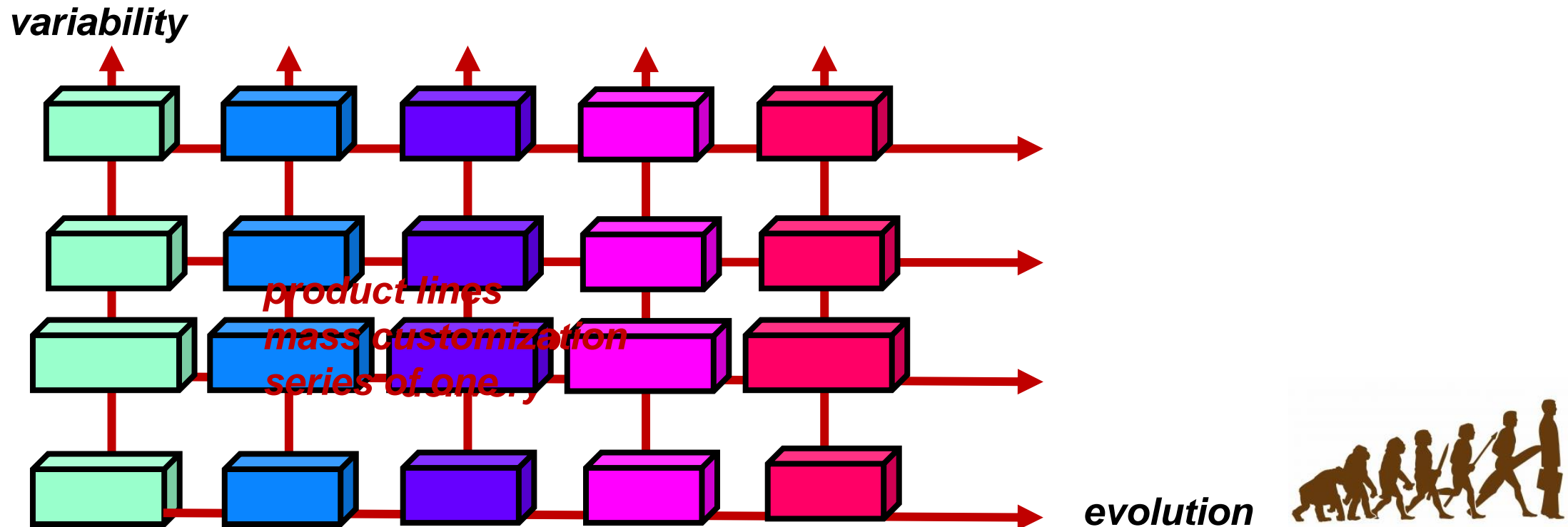
Testing Trends & Challenges



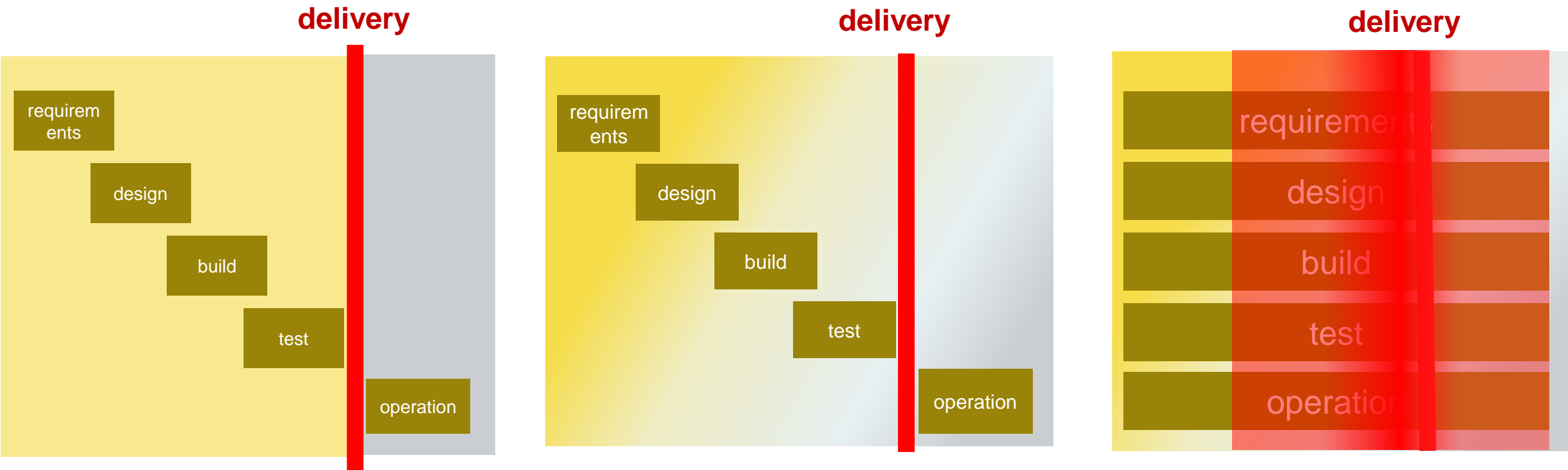
Evolution : Change over Time

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

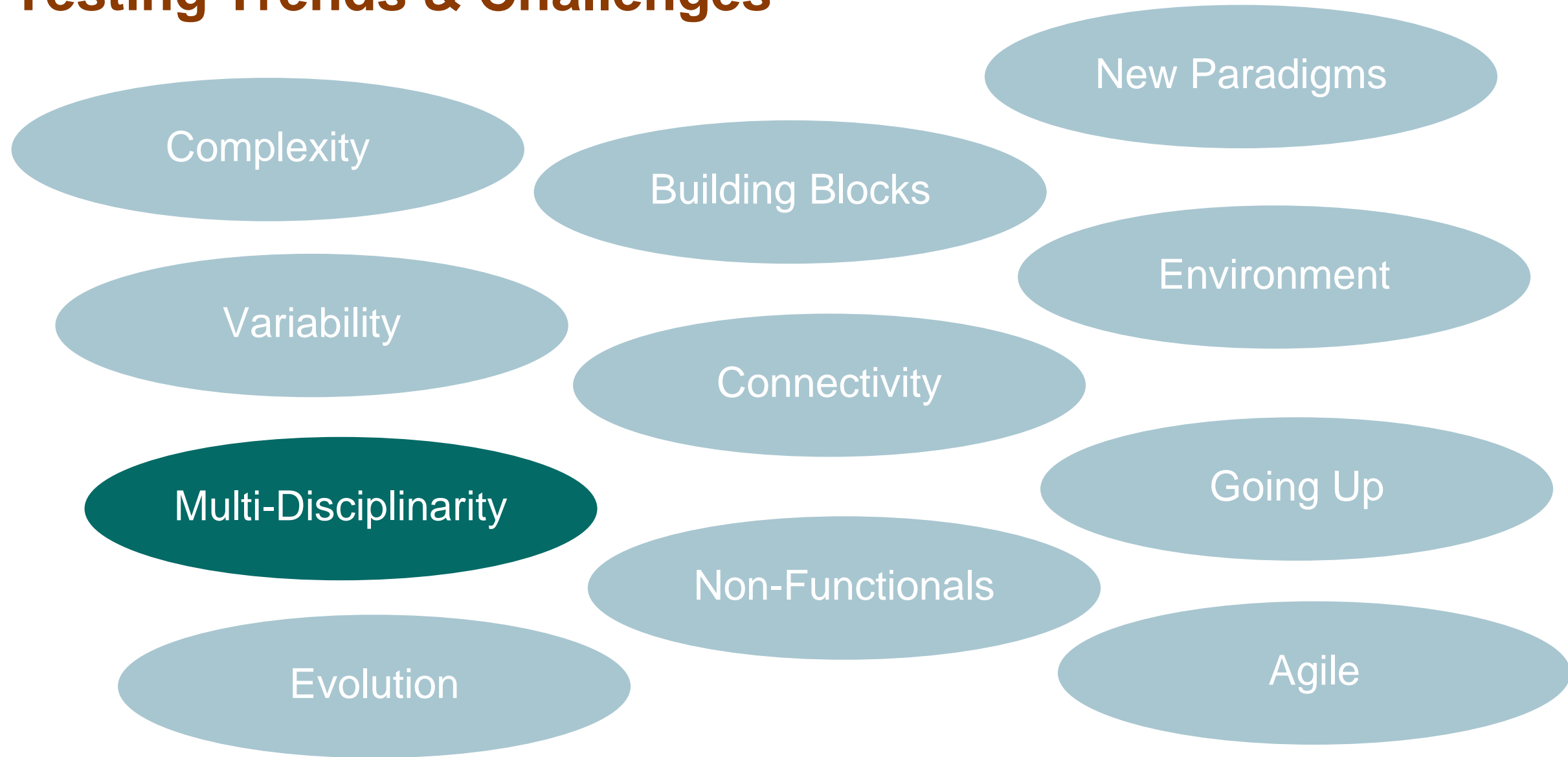
yet another source of
Test Explosion



Evolution, Change : Fading Boundaries



Testing Trends & Challenges



Cyber-Physical Systems



Semiconductor manufacturing equipment



Medical systems



Food processing



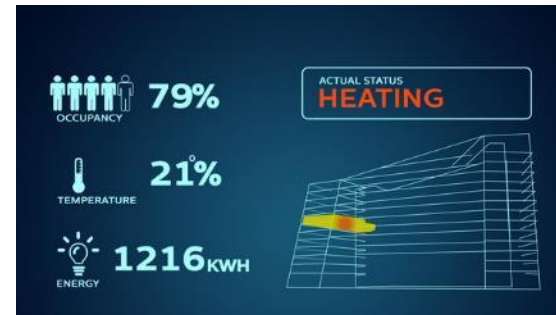
Agricultural robots



Traffic management



Electron
microscopes



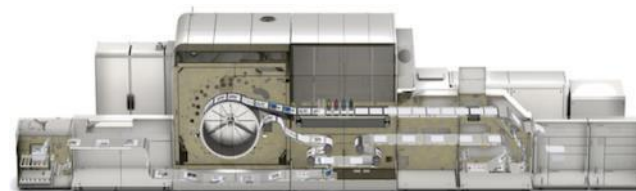
Building control



Robotized warehousing



Combat management systems



Industrial printers



Automotive

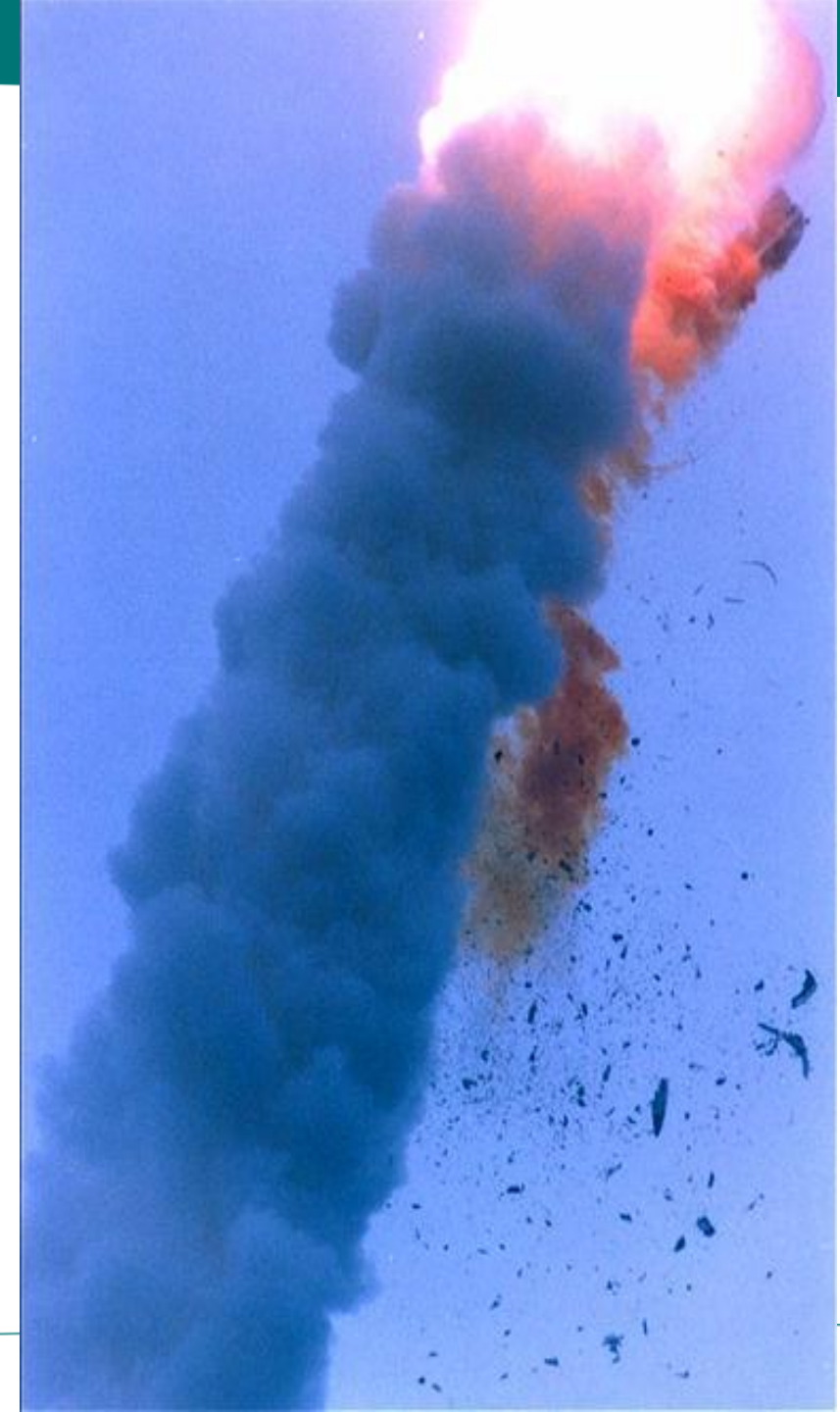
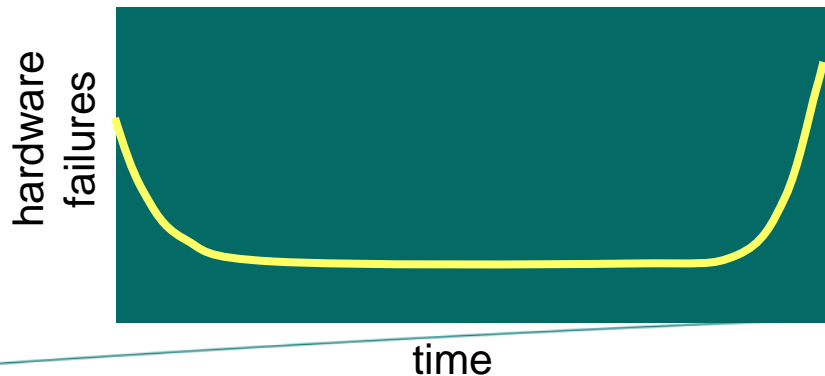


Dike

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**

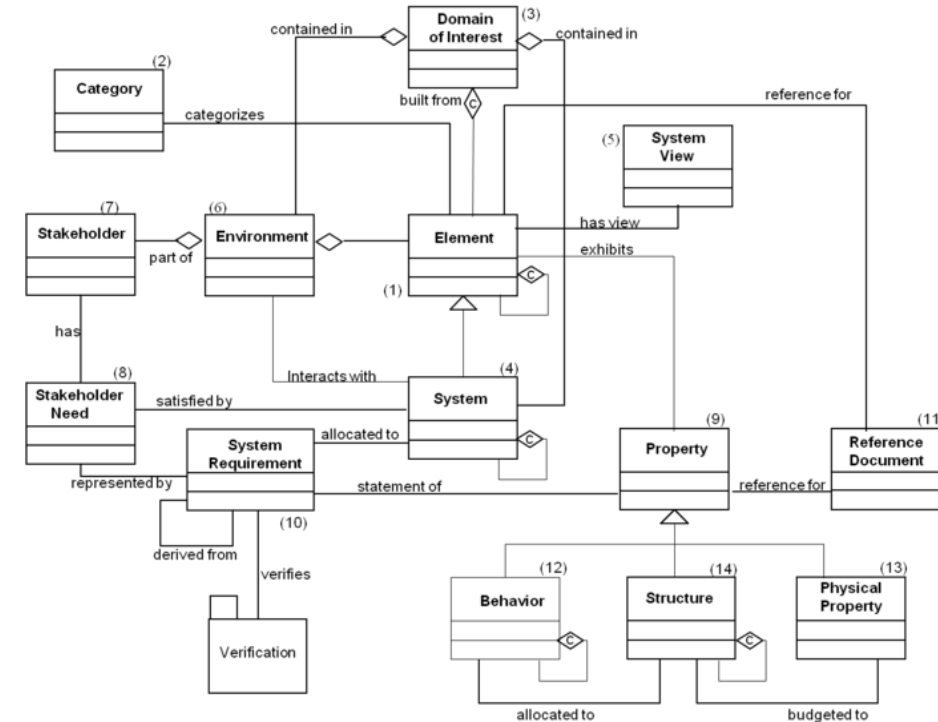
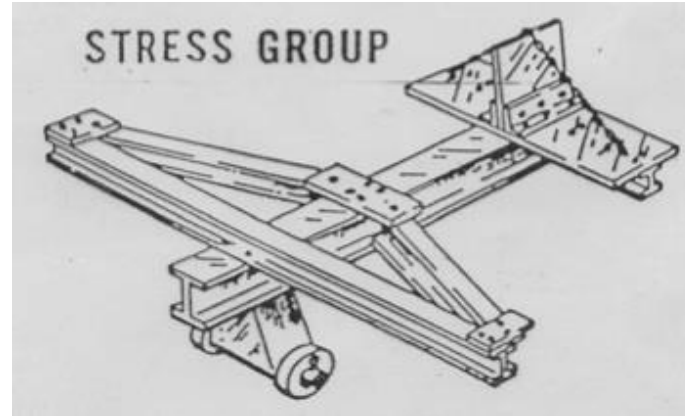
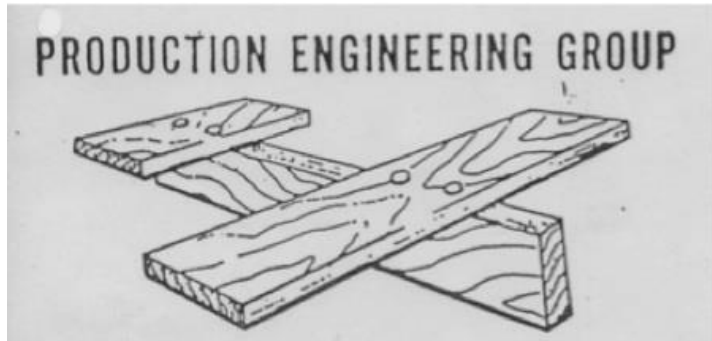
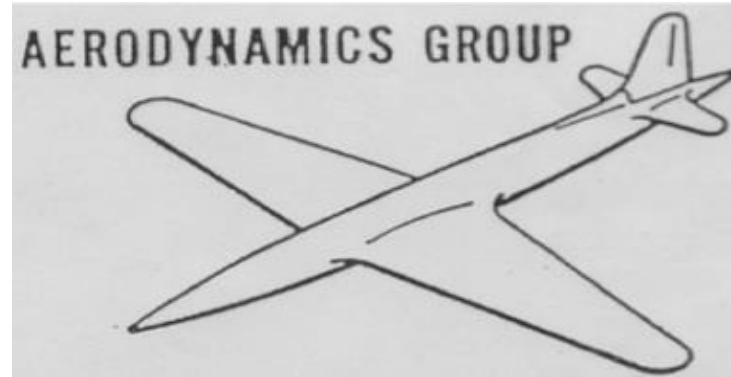
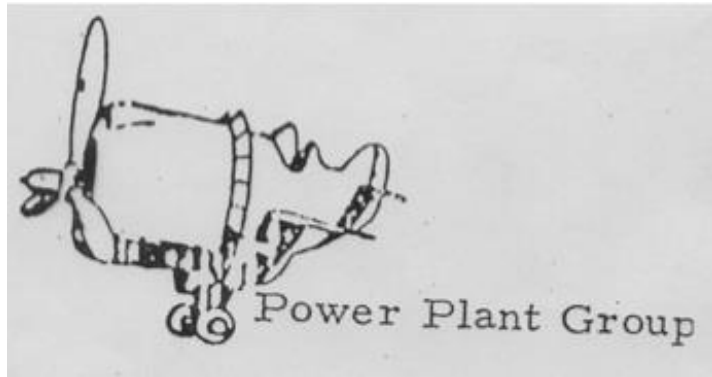
- models to simulate/emulate physical and environment in **1**
- intelligent stub, in-the-loop testing
- because real system is: expensive, infeasible, dangerous, too slow, too fast, cannot produce error scenarios, ...

- **Modeling**

- system \leftrightarrow physical part \leftrightarrow software \leftrightarrow environment
- models for virtualization \leftrightarrow models for testing

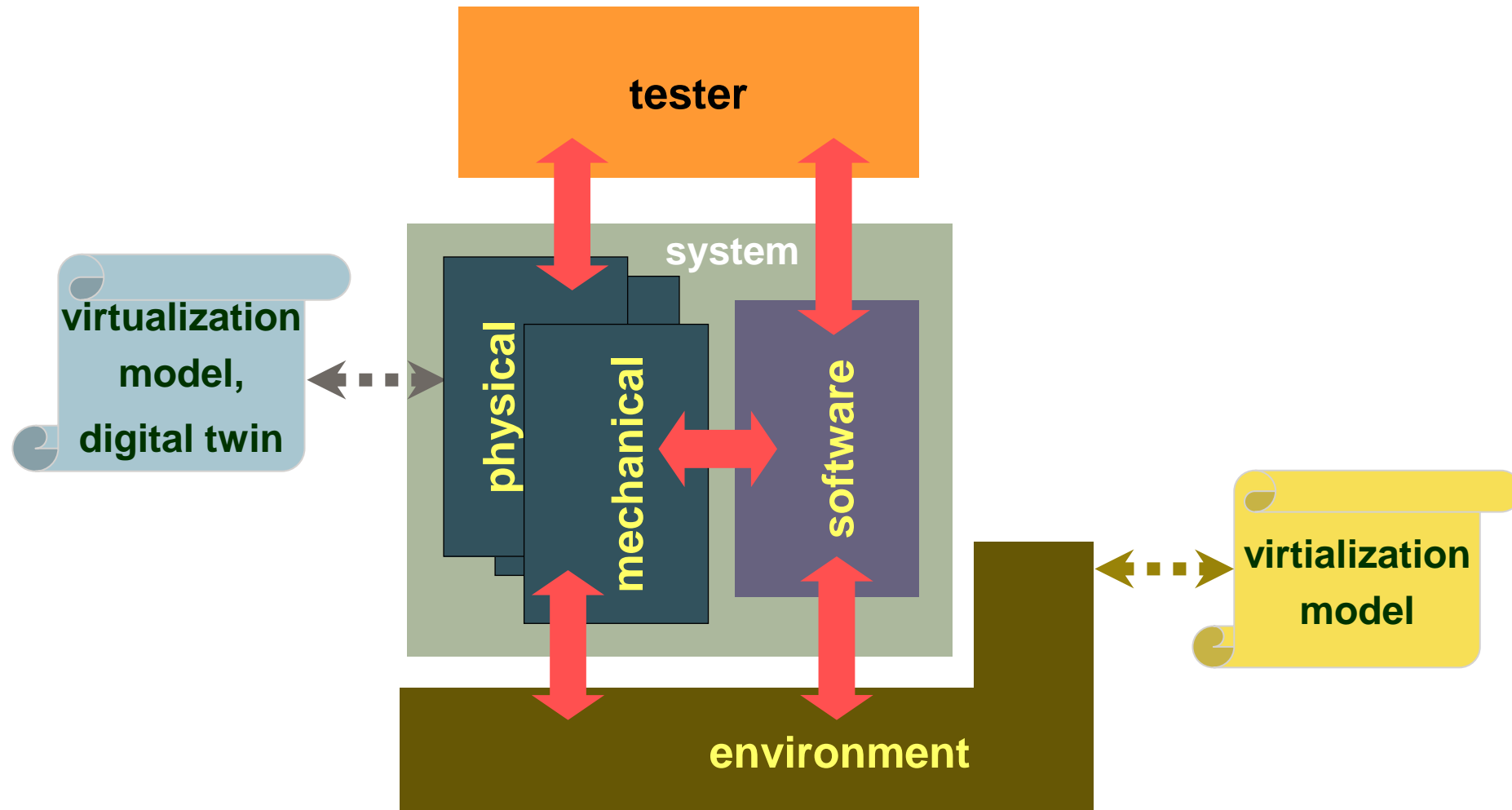


Multi-disciplinarity : Different Views on Systems

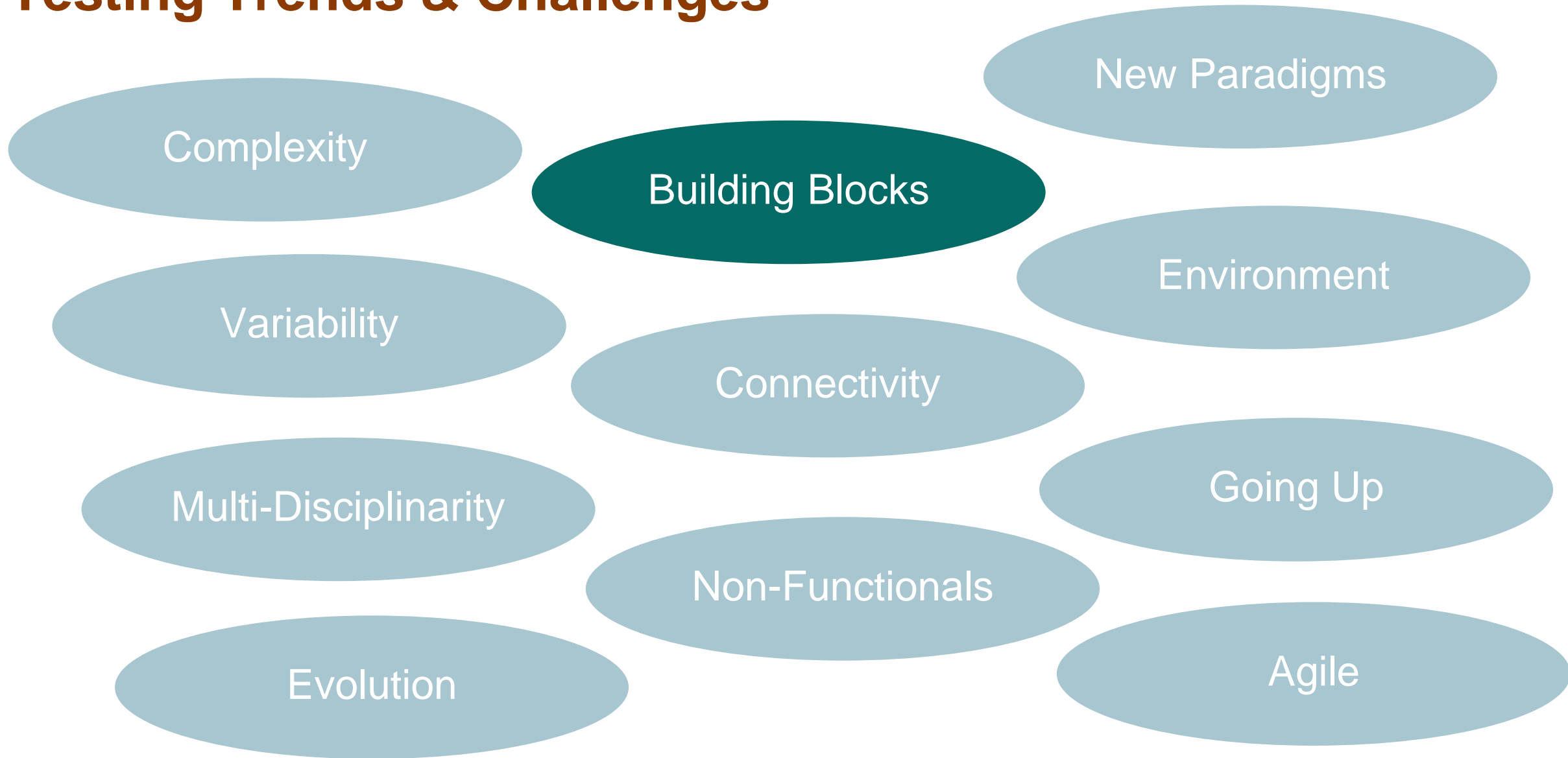


software group

Models for Multi-disciplinary Testing



Testing Trends & Challenges



Building Blocks : Components

Component

TO REUSE, OR NOT TO REUSE



open source

IN PARTICULAR,
IN TIMES OF CONTINUOUS CHANGES

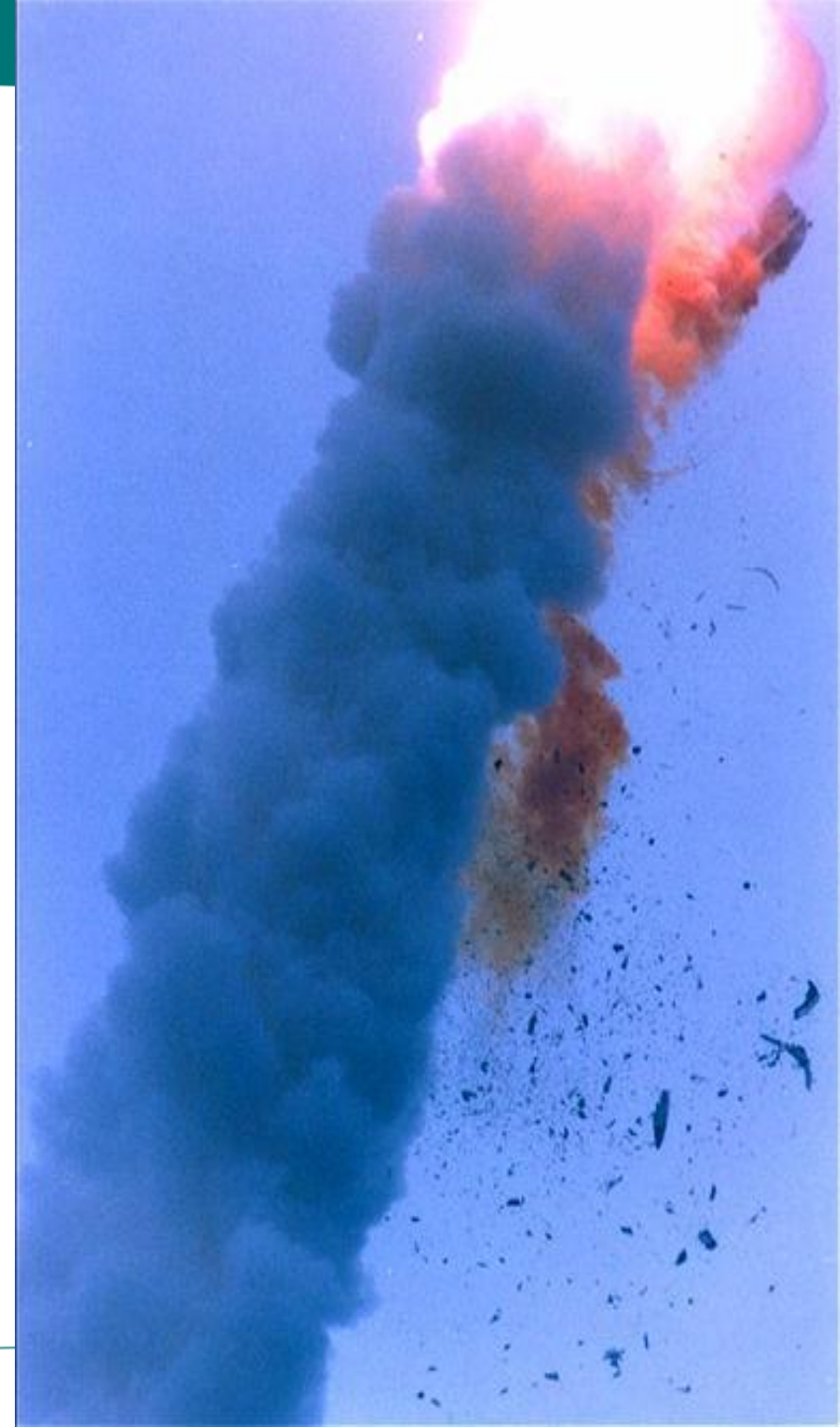


- reuse
- platform
- integration challenges
- dependencies
- when to test
- where to diagnose, repair

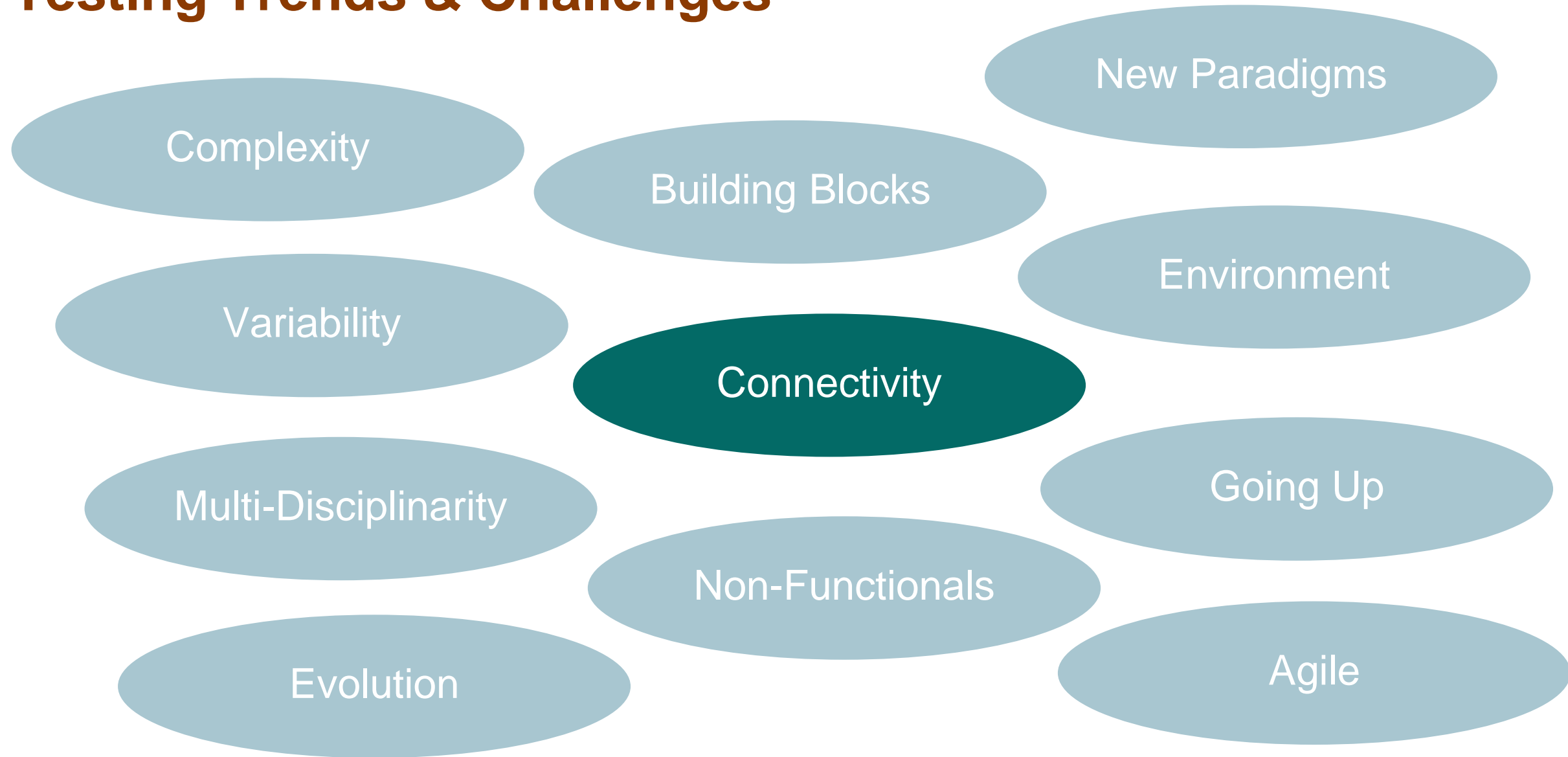
Components and Failures

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

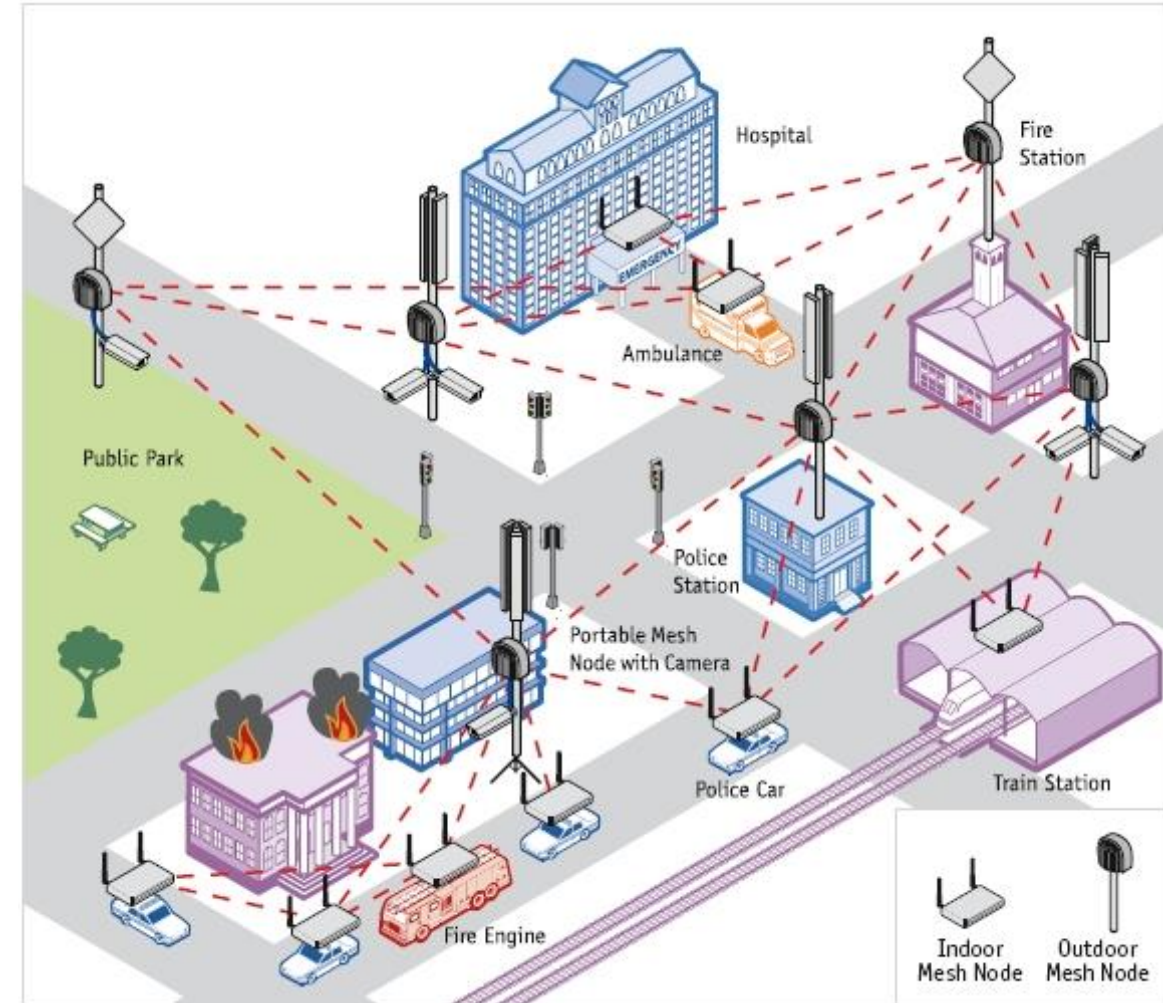


Testing Trends & Challenges

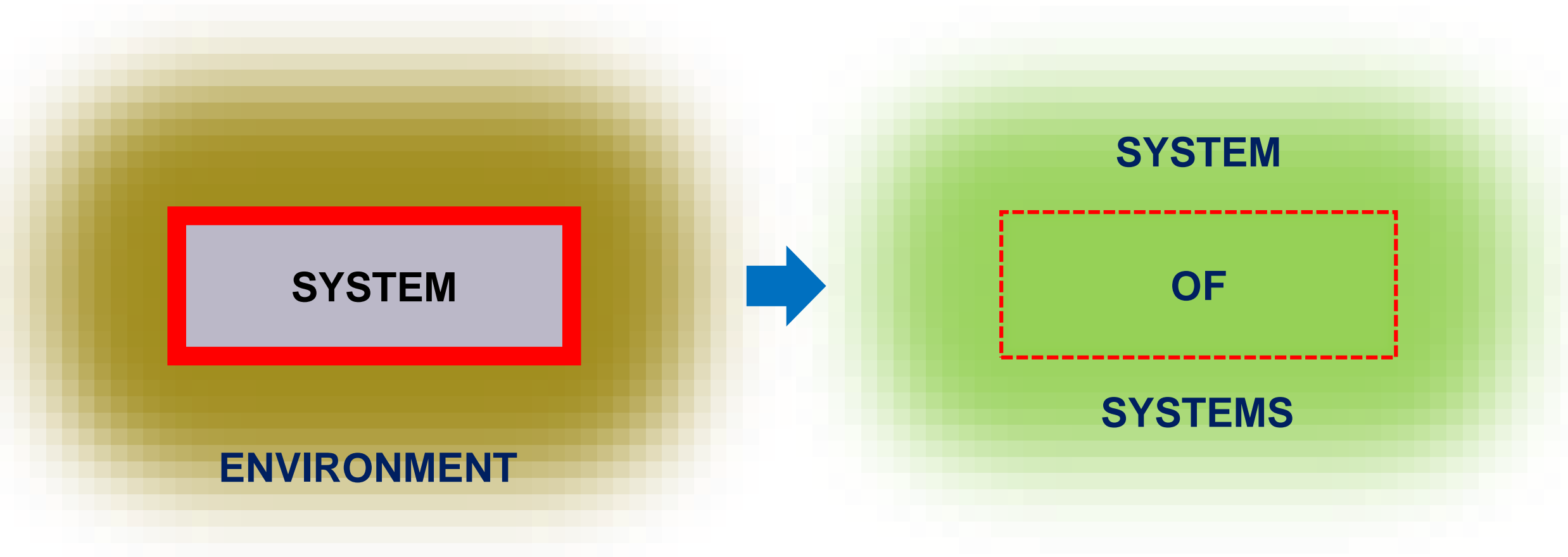


Connectivity

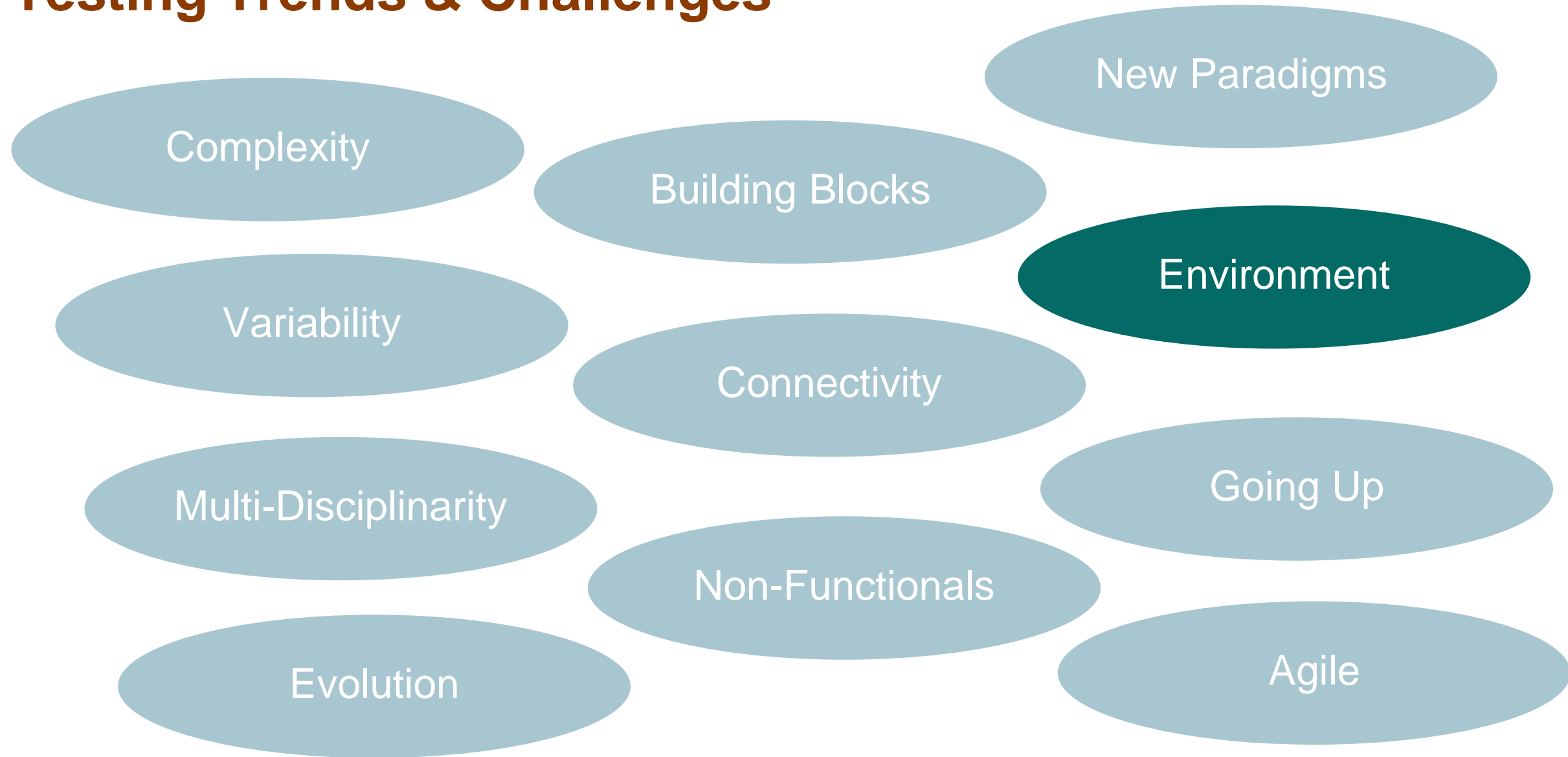
- Blurring boundaries of systems
→ everything connected
- Systems-Of-Systems
 - Dynamically connected systems
 - Not under own control
- Software is glue
 - with internal and external world
- Testing:
 - what is SUT ?
- Virtualization
 - which systems are available for testing ?
 - which systems must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



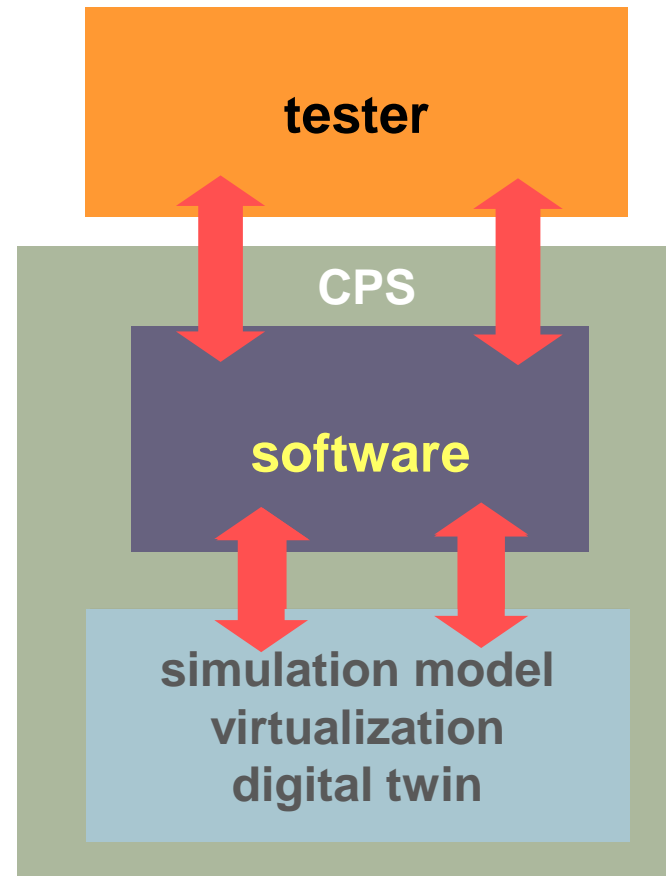
Fading Boundaries



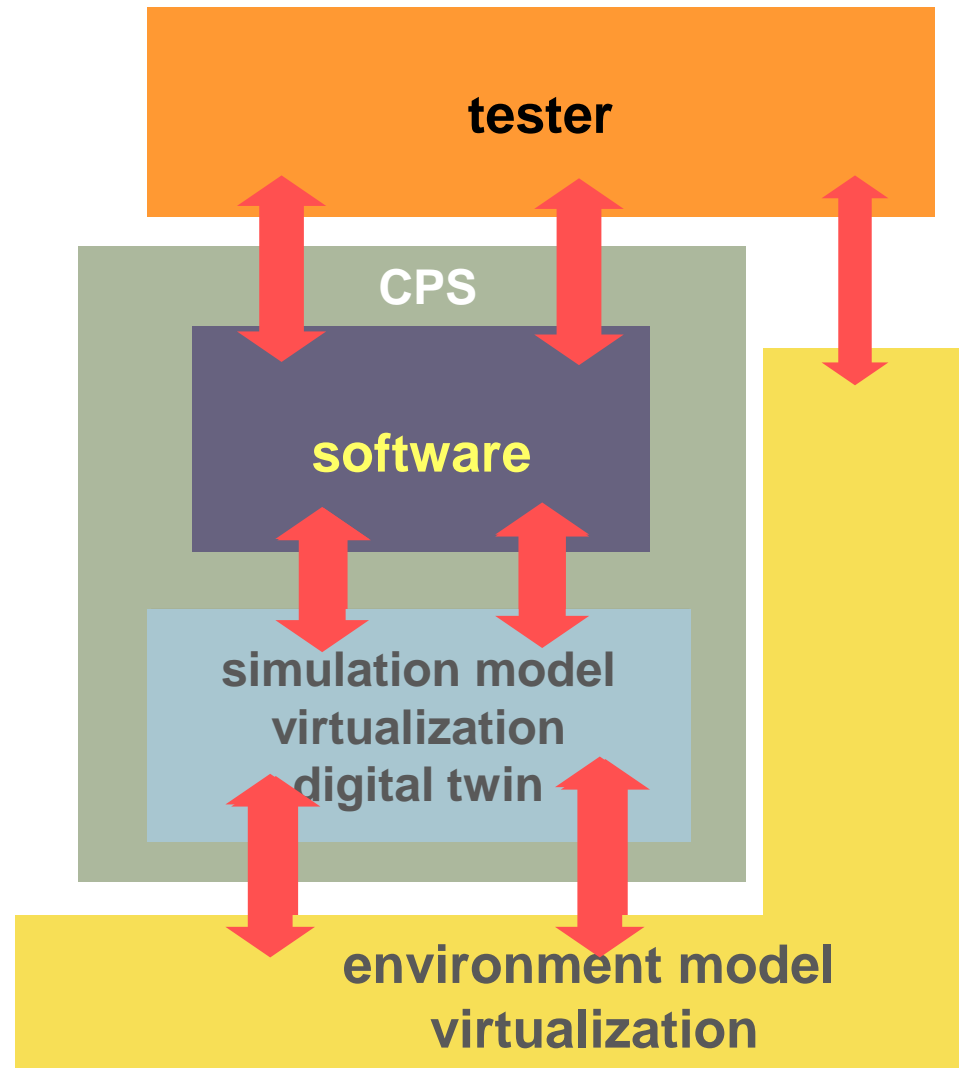
Testing Trends & Challenges



Environment



Environment



Environment

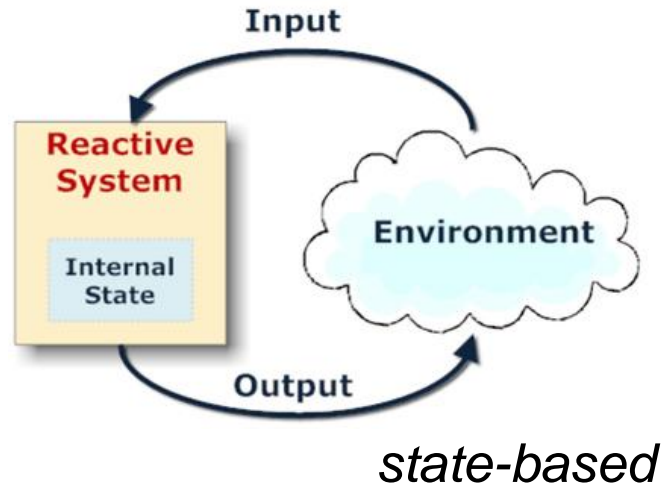


functional

computation : $I \rightarrow O$

tests over I

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



reactive : $I, S \rightarrow O, S'$

tests over I, S

autonomous



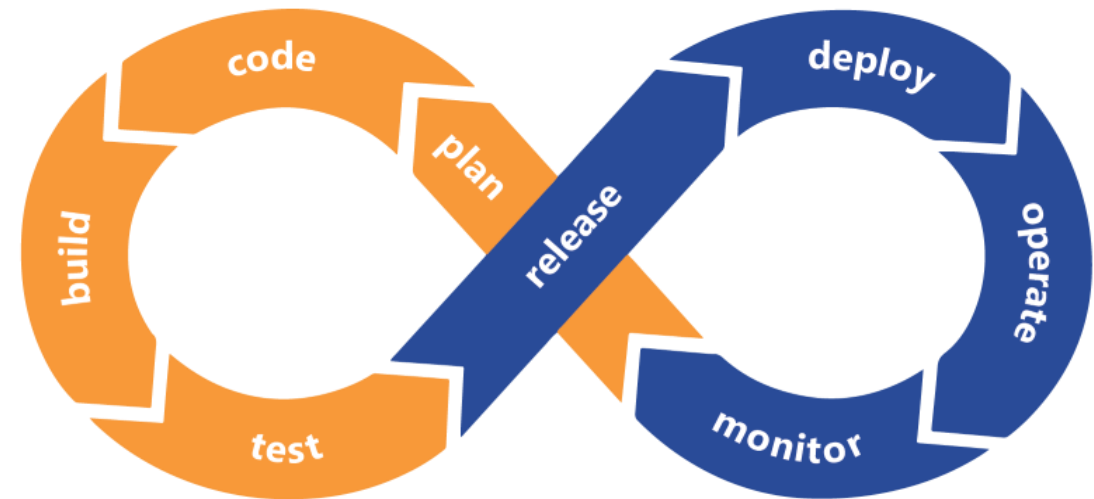
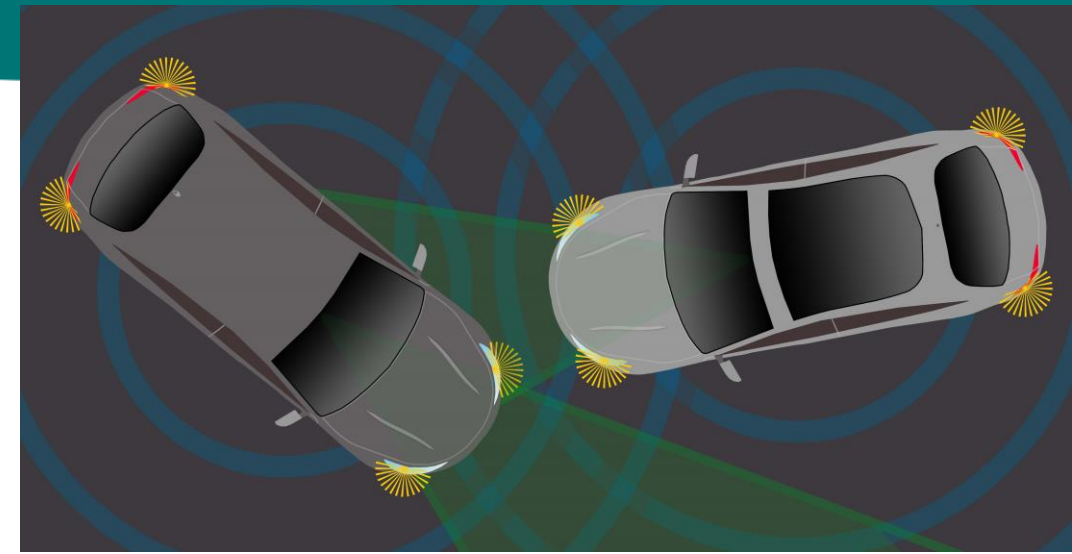
autonomous : $I, S, E \rightarrow O, S', E'$

tests over I, S, E

who determines I ?

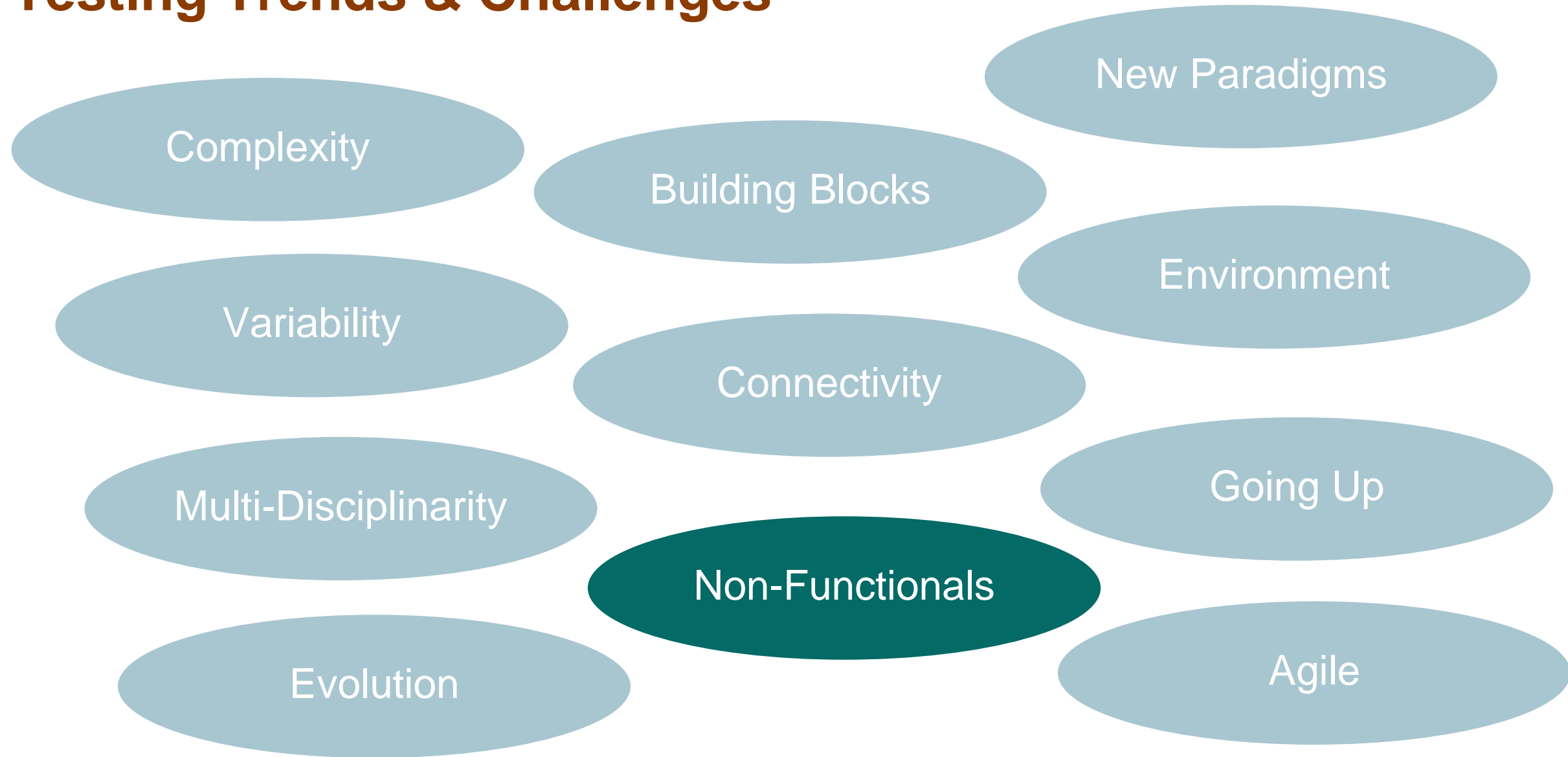
Environment

- **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

```
graph TD; Root[Quality Characteristics] --- Reliability[Reliability<br/>maturity<br/>fault tolerance<br/>recoverability<br/>availability<br/>degradability]; Root --- Efficiency[Efficiency<br/>time behaviour<br/>resource behaviour]; Root --- Portability[Portability<br/>adaptability<br/>installability<br/>conformance<br/>replaceability]; Root --- Functionality[Functionality<br/>suitability<br/>accuracy<br/>interoperability<br/>compliance<br/>security<br/>traceability]; Root --- Usability[Usability<br/>understandability<br/>learnability<br/>operability<br/>explicitness<br/>customisability<br/>attractivity<br/>clarity<br/>helpfulness<br/>user-friendliness]; Root --- Maintainability[Maintainability<br/>analysability<br/>changeability<br/>stability<br/>testability<br/>manageability<br/>reusability];
```

Reliability

maturity
fault tolerance
recoverability
availability
degradability

Efficiency

time behaviour
resource behaviour

Portability

adaptability
installability
conformance
replaceability

Functionality

suitability
accuracy
interoperability
compliance
security
traceability

Usability

understandability
learnability
operability
explicitness
customisability
attractivity
clarity
helpfulness
user-friendliness

Maintainability

analysability
changeability
stability
testability
manageability
reusability

Quality Characteristics

```
graph TD; QC[Quality Characteristics] --> FS[Functional Suitability]; QC --> PE[Performance Efficiency]; QC --> P[Portability]; QC --> I[Interoperability]; QC --> S[Security]; FS --> FS_Attr[suitability, accuracy, interoperability, compliance, security, traceability]; PE --> PE_Attr[operationality, explicitness, customisability, attractivity, clarity, helpfulness, user-friendliness]; P --> P_Attr[portability, adaptability, installability, conformance, replaceability]; I --> I_Attr[interoperability, compatibility, compatibility, compatibility]; S --> S_Attr[security, security, security, security];
```

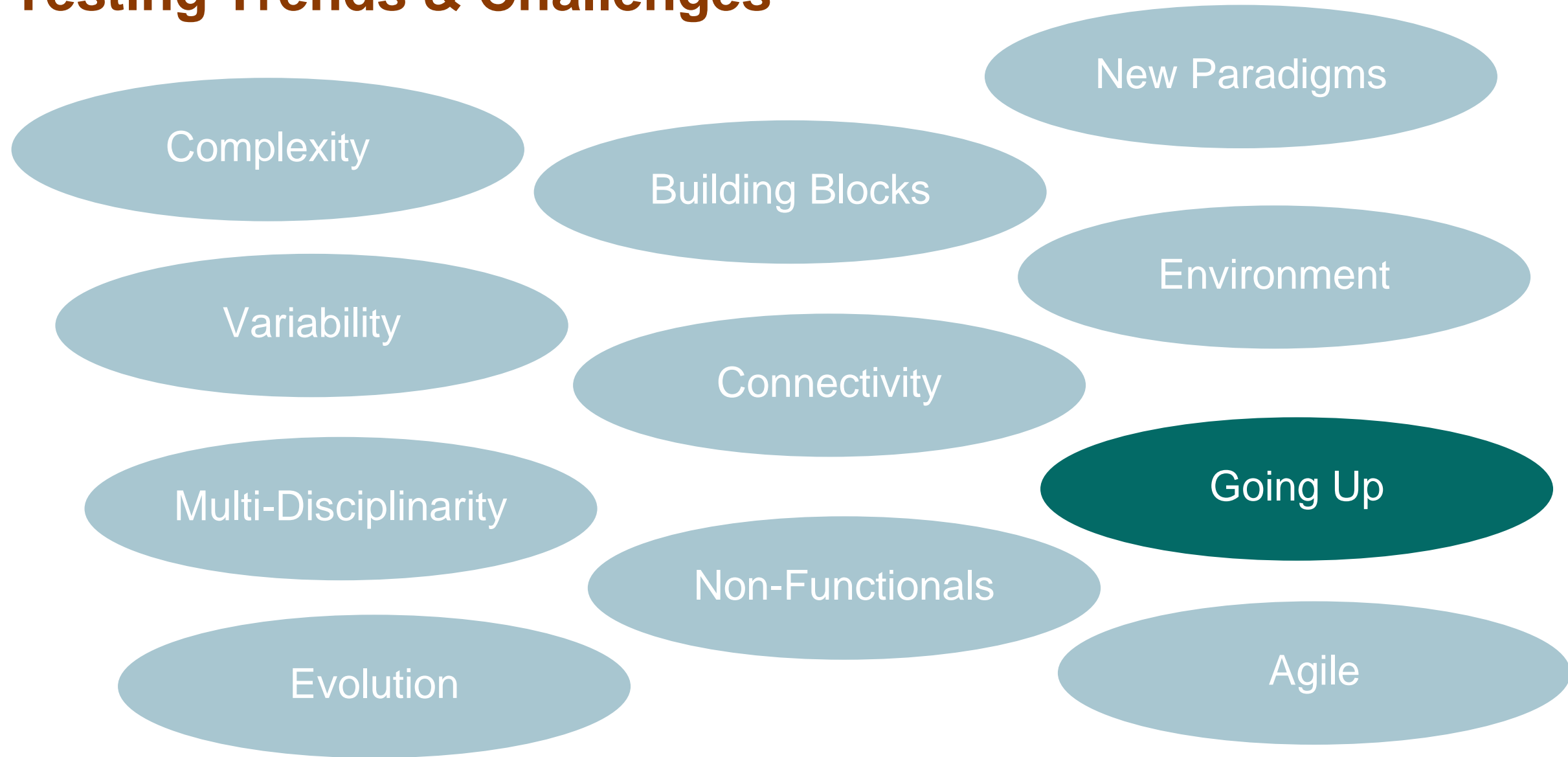
Trend: More Emphasis on
Non/Extra Functional
Quality Attributes
(“-ilities”)

Quality Characteristics

```
graph TD; QC[Quality Characteristics] --> FQC[Functional Quality Characteristics]; QC --> PQC[Performance Quality Characteristics]; QC --> RQC[Reliability Quality Characteristics]; QC --> UQC[Usability Quality Characteristics]; FQC --> F1[suitability]; FQC --> F2[accuracy]; FQC --> F3[interoperability]; FQC --> F4[compliance]; FQC --> F5[security]; FQC --> F6[traceability]; PQC --> P1[performance efficiency]; PQC --> P2[performance effectiveness]; PQC --> P3[performance predictability]; PQC --> P4[performance controllability]; PQC --> P5[performance flexibility]; PQC --> P6[performance robustness]; RQC --> R1[maturity]; RQC --> R2[fault tolerance]; RQC --> R3[recoverability]; RQC --> R4[availability]; RQC --> R5[degradability]; UQC --> U1[learnability]; UQC --> U2[interactivity]; UQC --> U3[changeability]; UQC --> U4[stability]; UQC --> U5[testability]; UQC --> U6[manageability]; UQC --> U7[reusability]; UQC --> U8[clarity]; UQC --> U9[helpfulness]; UQC --> U10[user-friendliness];
```

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

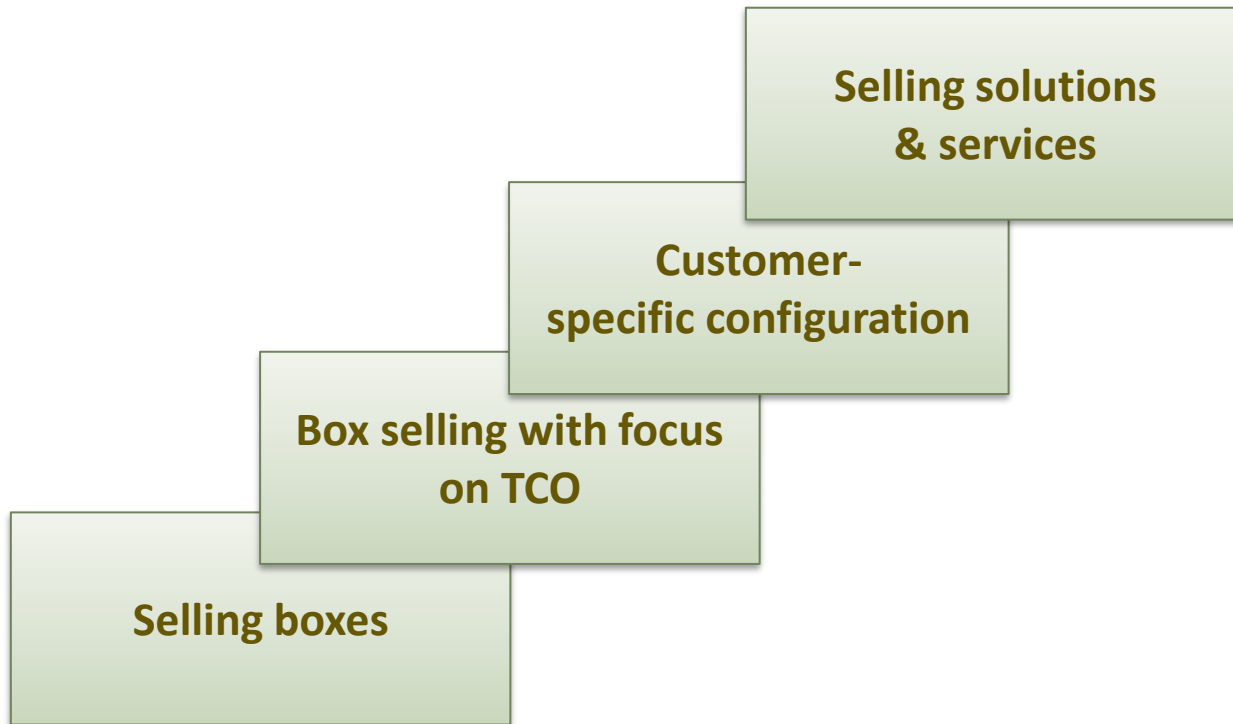
Testing Trends & Challenges



Going Up

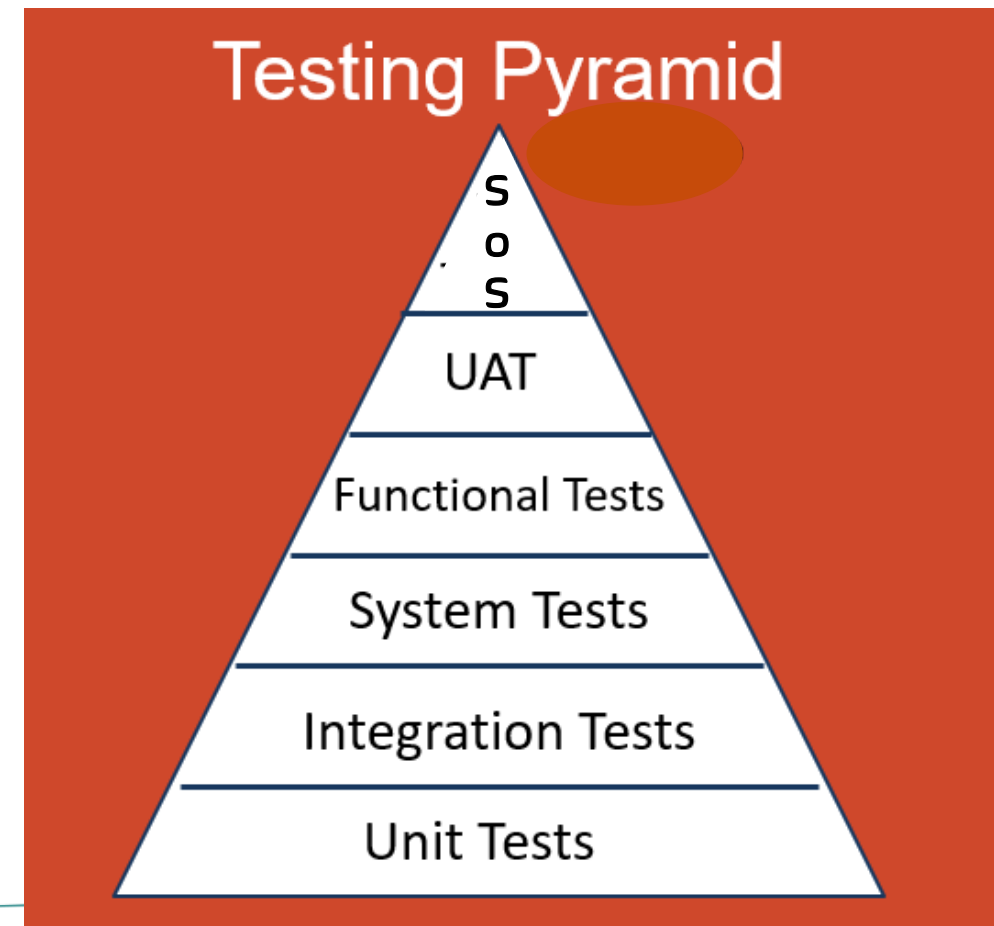
- **In the Value Chain**

- new business models
- testing quality-of-service



- **In the Test Pyramid**

- everybody does unit tests
- bugs are on the higher levels



Going Up

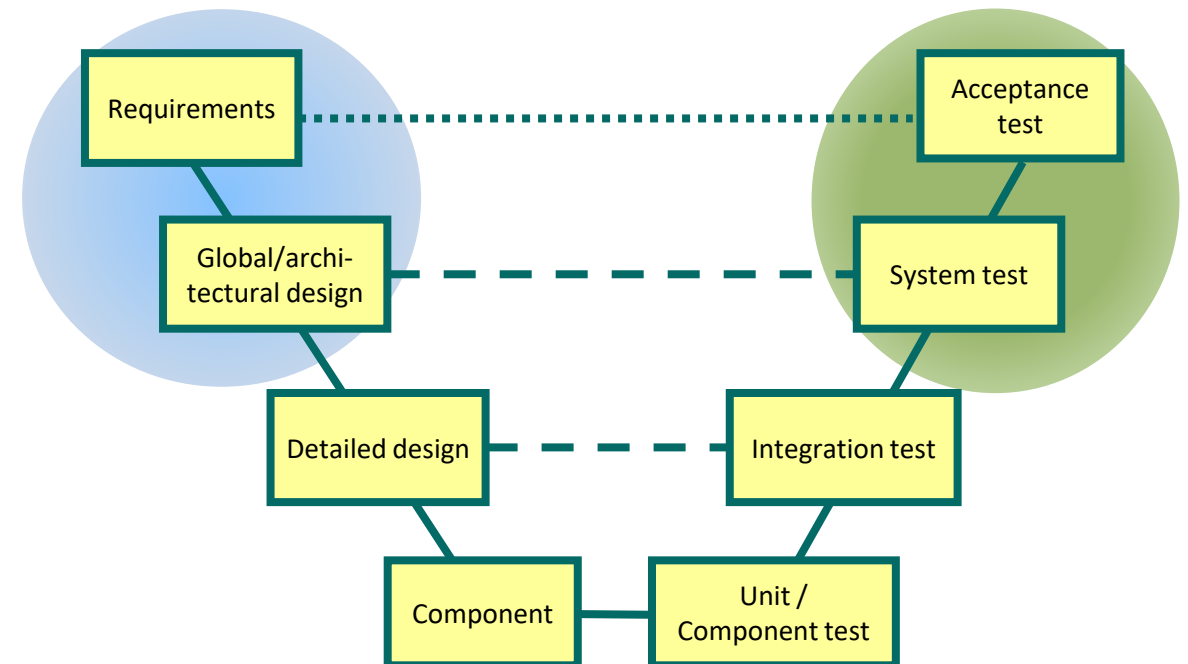
- In Coding

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

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
2.   <modelVersion>4.0.0</modelVersion>
3.   <groupId>com.mycompany.app</groupId>
4.   <artifactId>my-app</artifactId>
5.   <version>1.0-SNAPSHOT</version>
6.   <properties>
7.     <maven.compiler.source>1.7</maven.compiler.source>
8.     <maven.compiler.target>1.7</maven.compiler.target>
9.   </properties>
10.  <dependencies>
11.    <dependency>
12.      <groupId>junit</groupId>
13.      <artifactId>junit</artifactId>
14.      <version>4.12</version>
15.      <scope>test</scope>
16.    </dependency>
17.  </dependencies>
18. </project>
```

- In the V-Model

- requirements, design, system test
- detailed design, coding, unit tests outsourced

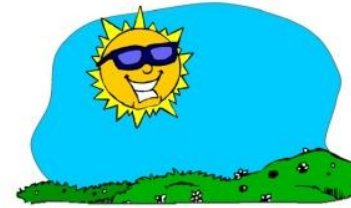


Going Up Consequence : Uncertainty & Non-Determinism

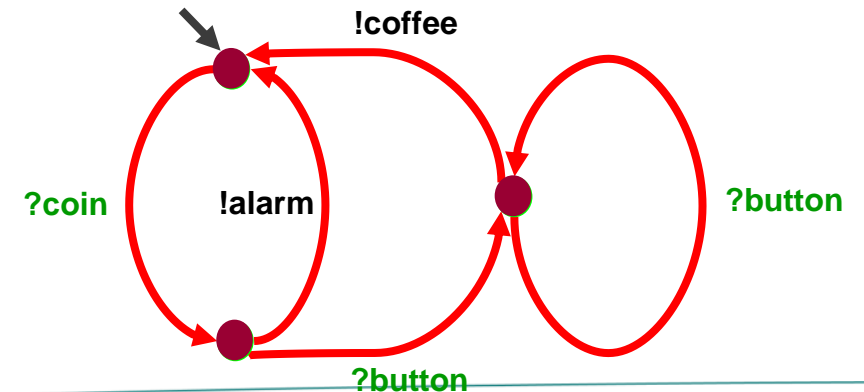
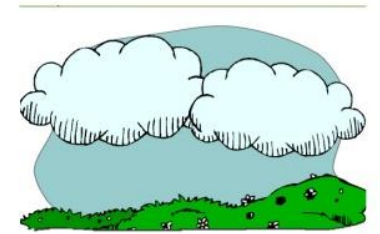
- Sometimes you don't know
 - testing a search engine, weather forecast, ...
 - systems-of-systems, big data, ...
- Sometimes you don't want to know
 - no details
 - abstraction
 - particular view

Uncertainty of test outcomes & oracles

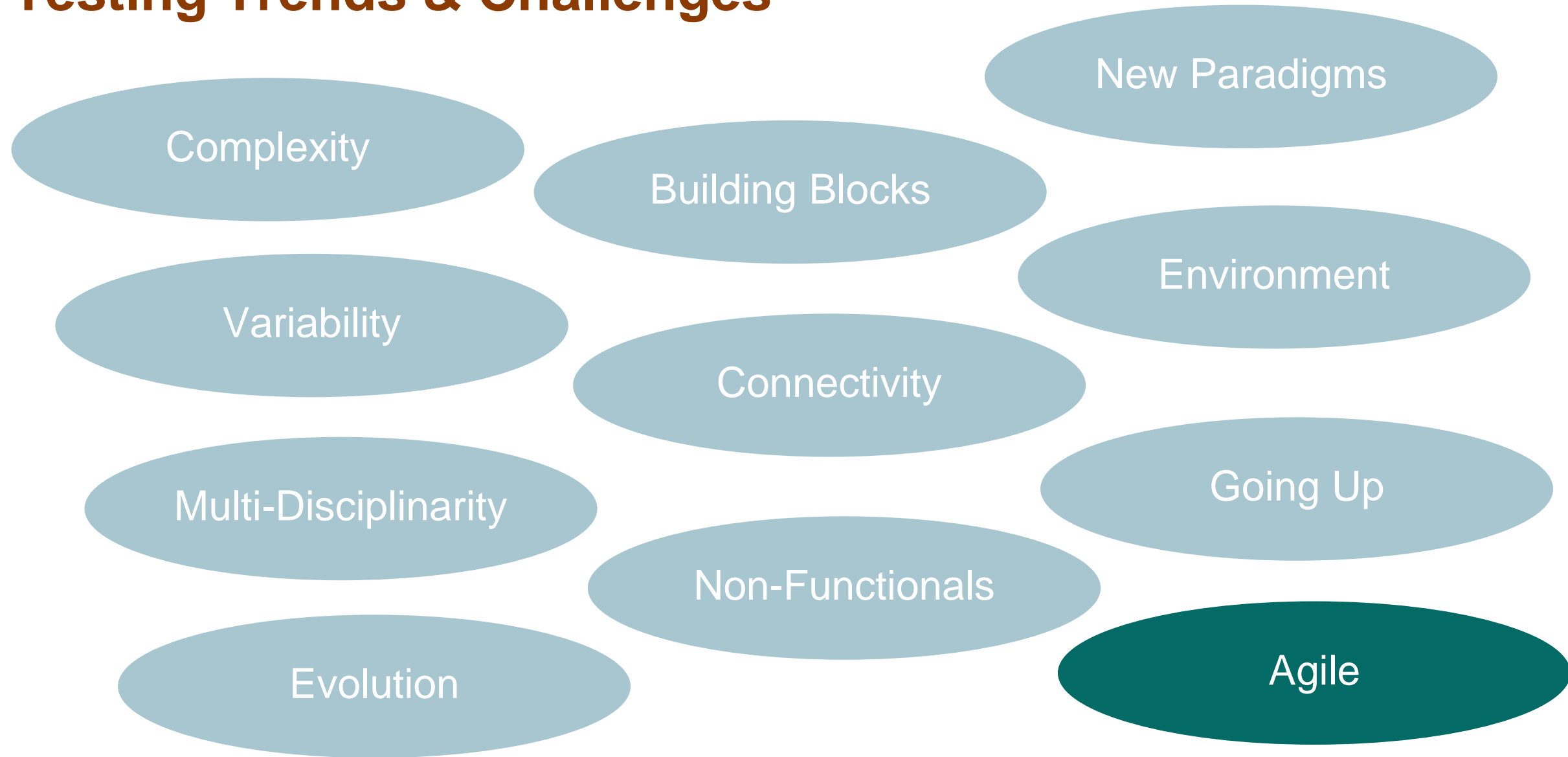
- non-determinism
- probabilities



What is the weather like ?



Testing Trends & Challenges



Agile ?



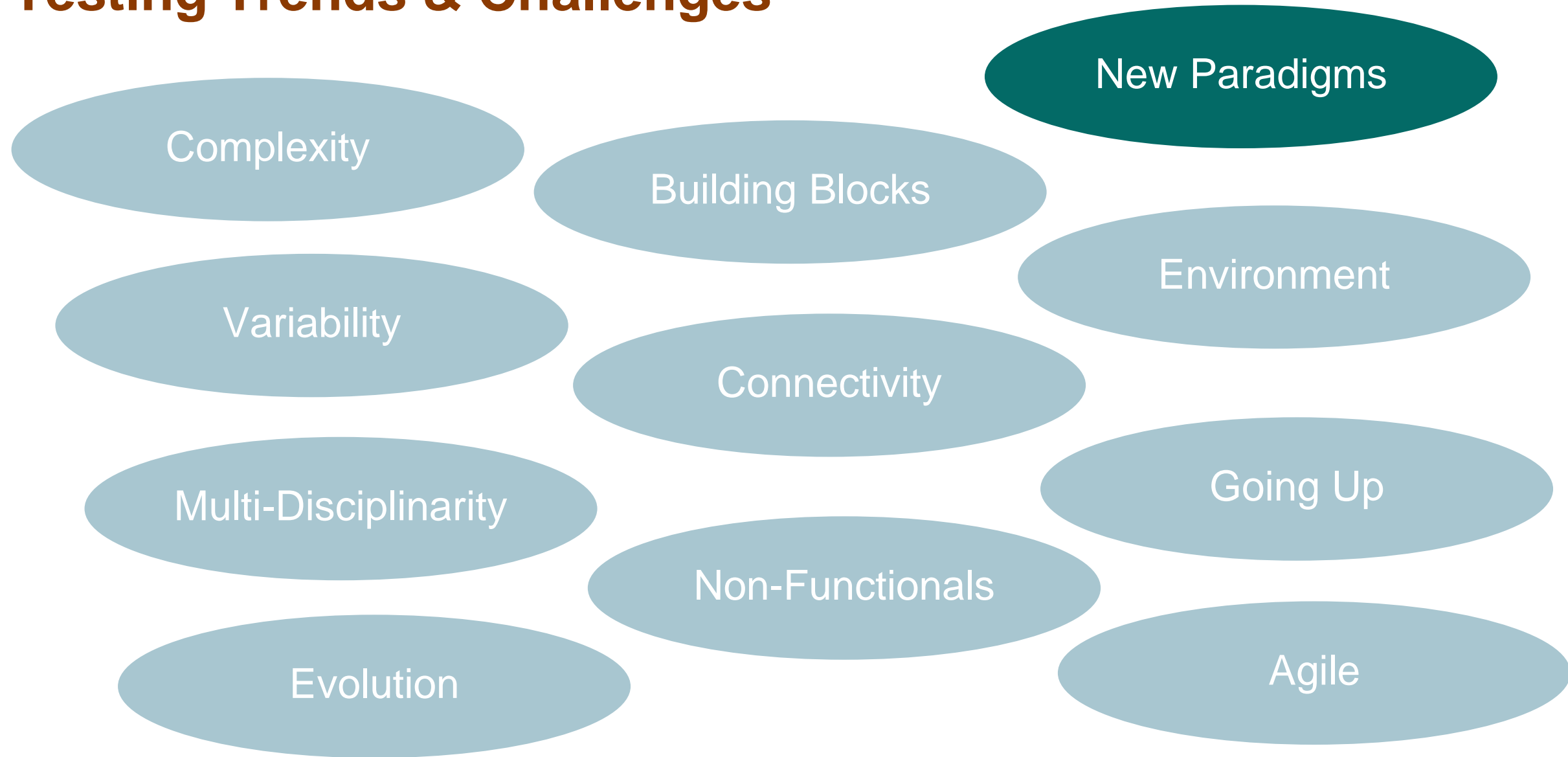
Agile

- 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
- Microservices
- 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 provider, ...

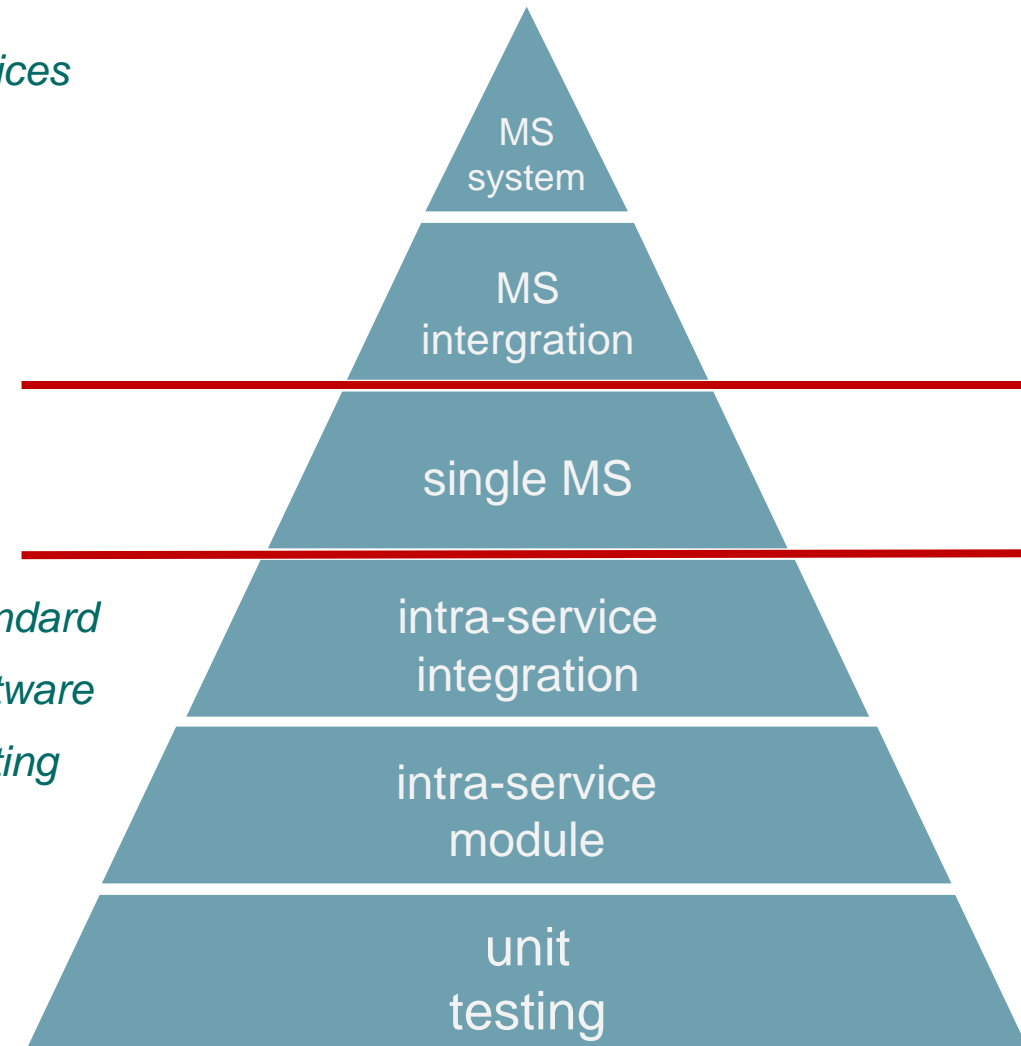
Level of Testing

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

Going Up

*microservices
testing*

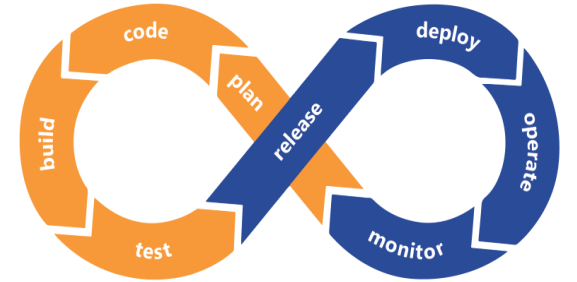
*standard
software
testing*



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 & Outlook

Dynamics

New Paradigms

Size & Complexity

Building Blocks

Variability

Environment

Connectivity

Multi-Disciplinarity

Going Up

Non-Functionals

Evolution

Agile

Testing Trends & Challenges

Complexity

Building Blocks

New Paradigms

Variability

Environment

Connectivity

Multi-Disciplinarity

Non-Functionals

Evolution

