A Perspective on Three Decades of Software Robustness Assessment

Nuno Laranjeiro

cnl@dei.uc.pt

1 2 9 0

UNIVERSIDADE D COIMBRA

14th International Workshop on Software Engineering for Resilient Systems (SERENE 2022), September 2022

Outline

- Context
- Robustness assessment throughout the years
- Highlights and challenges
- Lessons learned with REST

Context

Source: instagram.com/citybestviews

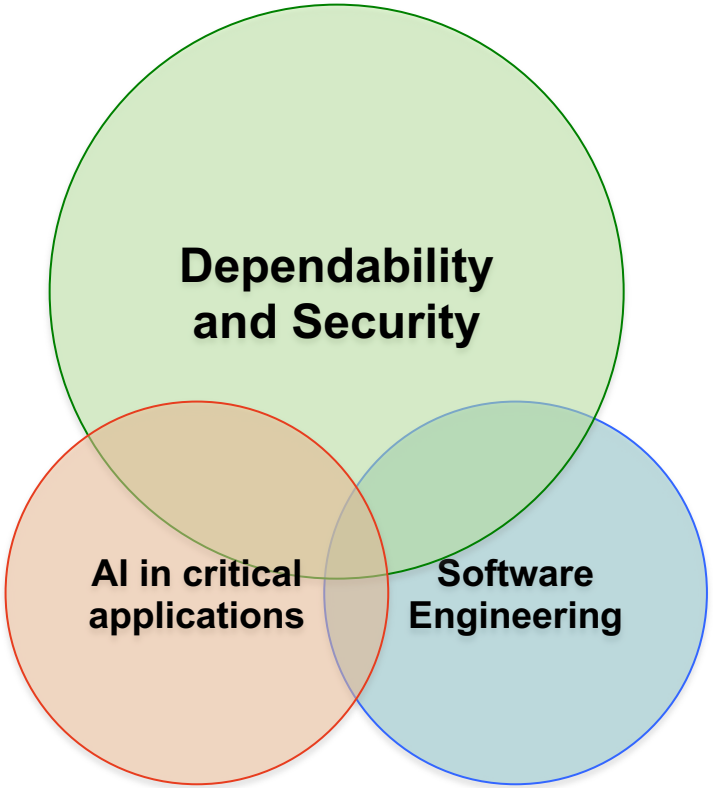
Coimbra

https://youtube.com/watch?v=n-iAyFZDgrE


1290
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA
de engenharia informática
1985 - 2020

Software and Systems Engineering Group

- 16 (+ 5) PhD members
- 34 PhD students
- ~20 MSc students
- https://www.cisuc.uc.pt/en/SSE



Nuno's background – Research

- Verification & Validation techniques
- Experimental dependability assessment
- Robustness testing
  - Web services robustness, middleware (e.g., messaging)
- Security and interoperability assessment
- Blockchain security
- Machine learning to in software engineering processes (V&V)
- https://eden.dei.uc.pt/~cnl

Software robustness assessment in the last 3 decades

Before this talk…

- Nuno Laranjeiro, João Agnelo, and Jorge Bernardino. 2021. A Systematic Review on Software Robustness Assessment. *ACM Computing Surveys* 54, 4, Article 89 (May 2022), 65 pages. https://doi.org/10.1145/3448977

- Nuno Laranjeiro, João Agnelo and Jorge Bernardino. 2021. A Black Box Tool for Robustness Testing of REST Services. *IEEE Access*, vol. 9, pp. 24738-24754, 2021, doi: 10.1109/ACCESS.2021.3056505.

Definitions

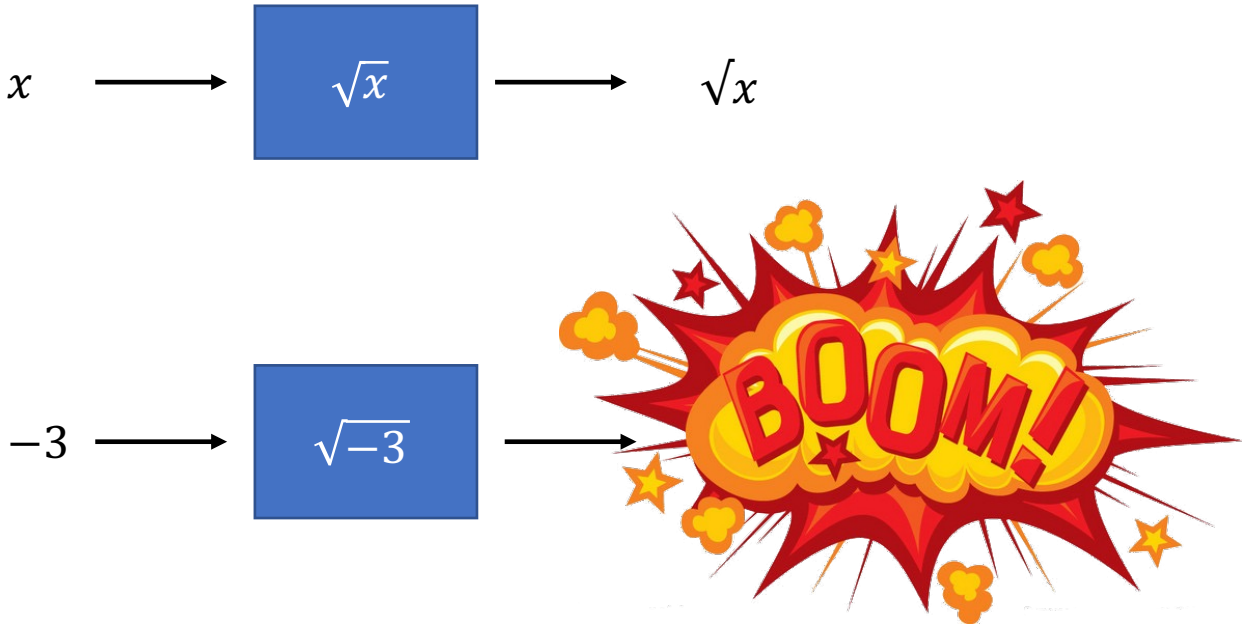
- **Robustness** is the degree to which a certain system or component can operate correctly in the presence of invalid inputs or stressful environmental conditions
- **Robustness assessment** aims at characterizing the behavior of a system in presence of a particular class of faults (i.e., external faults)

Motivation

- Software systems now support our daily lives
  - Entertainment, business, healthcare, …
- Residual faults may be activated by erroneous or malicious inputs, or stressfull conditions
- A software failure may lead to disastrous consequences
  - Financial losses, safety issues
- Robustness assessment activities are essencial
  - How does your autonomous car react in presence of a STOP sign?
  - What if the STOP sign is slightly damaged?
  - What if the camera system in your car malfunctions?
  - How will your autonomous car operate during an earthquake? or during road-side constructions?

Motivation – Example 1

- Let's have a look at this robustness assessment example



Motivation – Example 2

- Another robustness assessment example

- https://www.youtube.com/watch?v=aFuA50H9uek

Motivation

- Long period of known research on robustness evaluation
- Large number of works on robustness assessment
- Large heterogeinity of approaches and targeted systems
- No large-scale view of robustness assessment approaches

The process

- Reviewed research from 1990 – 2020
- Research strongly connected to AI/ML was not considered
- The systematic reviewing process lead to the identification of 145 works on robustness evaluation

Open questions

- Which **types of software systems** are the subject of robustness evaluation?
- Which **techniques** are used to evaluate software robustness?
- Which are the **targets** used by software robustness evaluation approaches?
- Which **types of faults** are being used in software robustness evaluation?
- Which are the **methods** used to characterize robustness?

Which types of software systems are the subject of robustness evaluation?

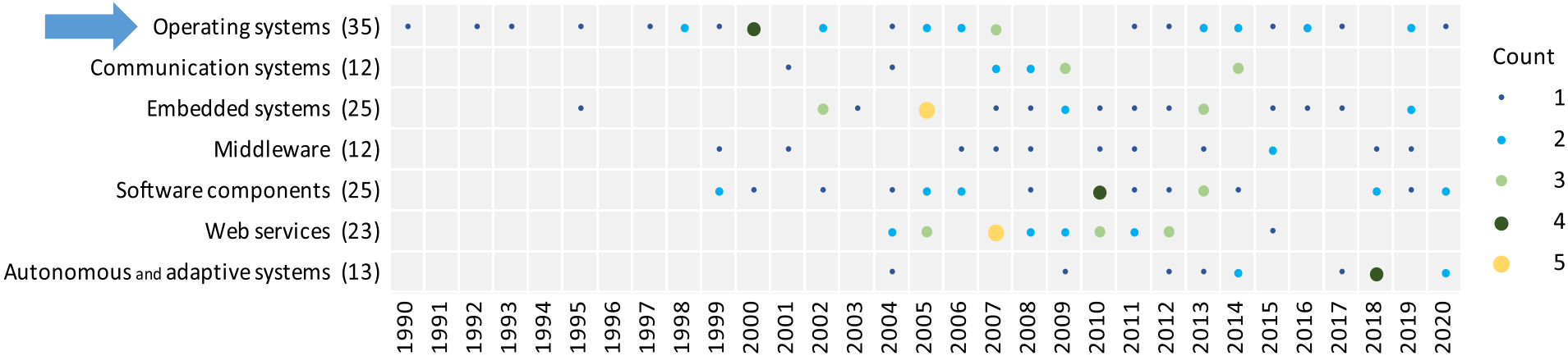
Types of systems

- Operating systems
  - General-purpose, including mobile
- Communication systems
  - Network-centric systems, including protocol implementations
- Embedded systems
  - Designed to handle a certain single specific task
  - Often used in mission or safety-critical environments
  - Time as an important property

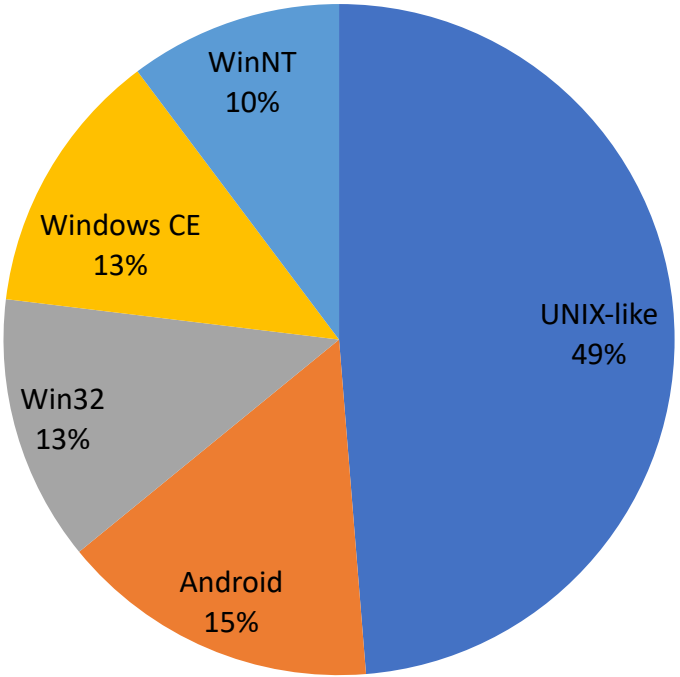
Types of systems

- Middleware
- Software components
  - Commercial Off-the-shelf software and applications (COTS), that do not overlap with other groups (namely, general purpose OS)
  - Other reusable software (libraries)
- Web services
  - HTTP-based, including SOAP services and web applications
- Autonomous and adaptive systems
  - Systems that are able to adapt to environment changes
  - Usually involve a feedback loop

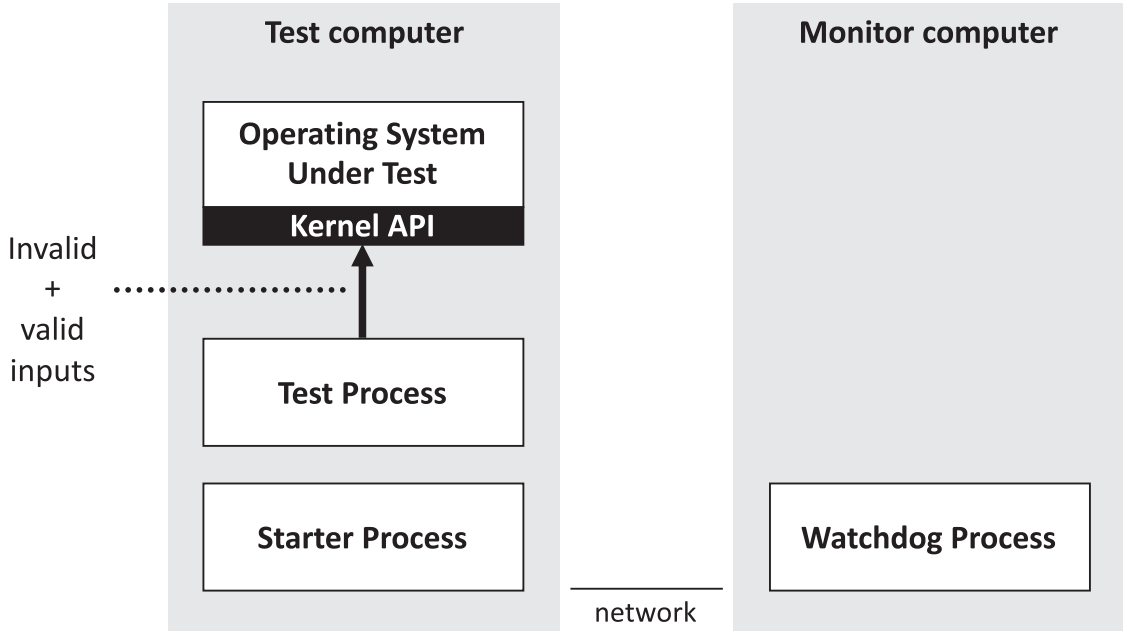
Distribution across time



Operating systems



A classic example (Ballista project)



Approach

- Set of system calls in the operating system API
- Definition of valid inputs used along with invalid inputs for each data type in the call parameters
- **Invalid inputs** – values holding particular characteristics, which tend to be the source of robustness problems (e.g., NULL, 0, 1, −1, string overflow, special characters).
- **Mature systems, but…** results showed significant failures in the ability to gracefully or correctly handle exceptional conditions.

Failure classification

- **C**atastrophic (operating system crashes or multiple tasks affected)
- **R**estart (process hangs and requires restart)
- **A**bort (process aborts)
- **S**ilent (exception was not signaled but should have been)
- **H**indering (incorrect exception signaled)

Failure classification

- **C**atastrophic (operating system crashes or multiple tasks affected
- **R**estart (process hangs and requires restart)
- **A**bort (process aborts)
- **S**ilent (exception was not signaled but should have been)
- **H**indering (incorrect exception signaled)

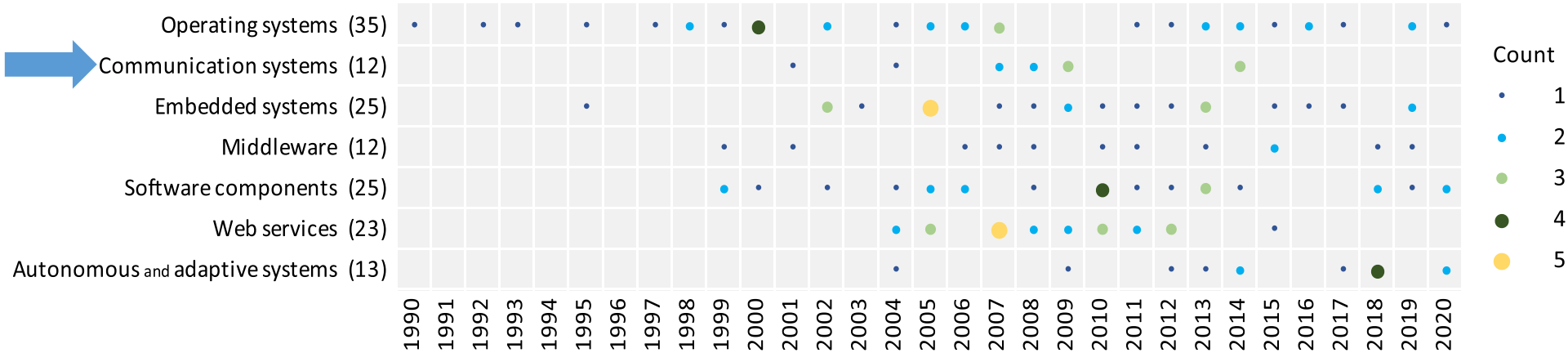
Operating Systems (highlights) (1)

- Testing is the main approach among OS
- Combination of valid and invalid inputs
- Kernel as starting point
- Challenges
  - Good quality workloads are important
    - Code/functionality coverage
  - Difficult to identify certain types of failures
    - Observation points and oracles
- Application to multi-version software
  - Check the conformance to standards

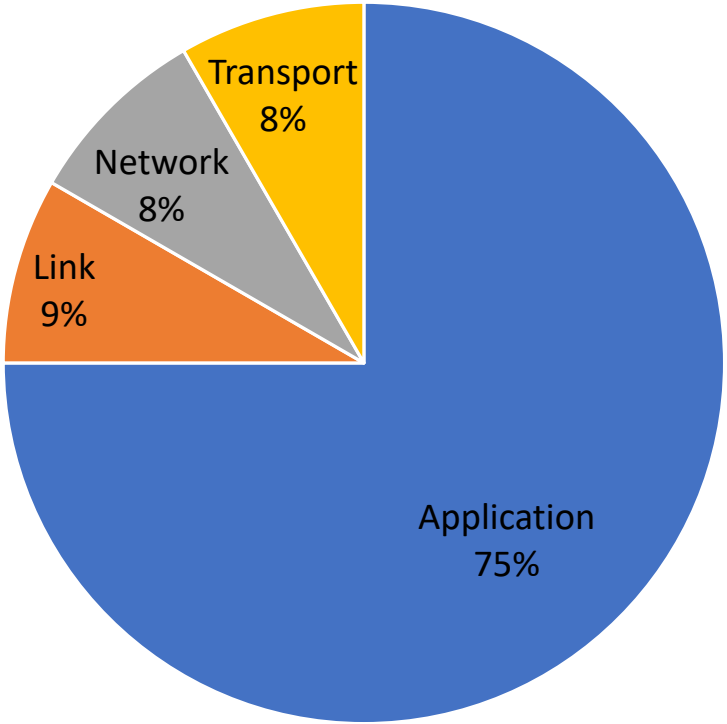
Operating Systems (highlights) (2)

- Focus on better workloads
- From kernel to libraries, utilities, drivers
- Same programming mistakes repeatedly observed over time
- Move from traditional to mobile operating systems

Distribution across time

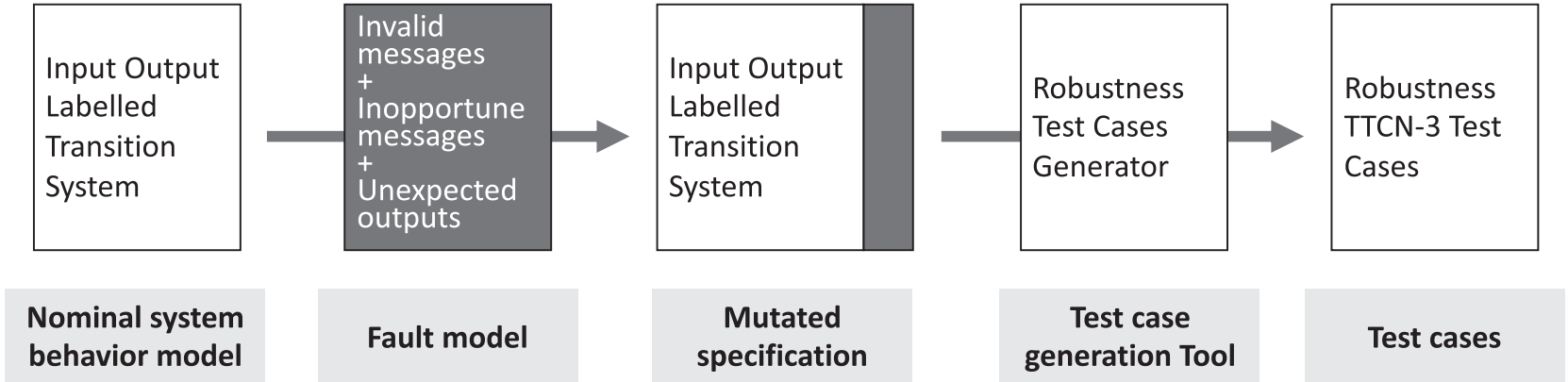


Communication systems



Common approach

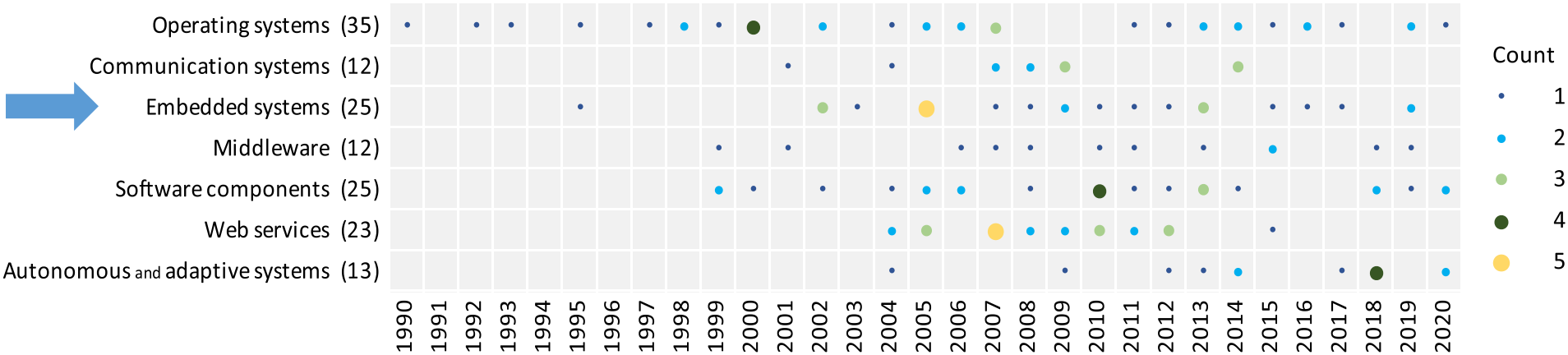
- Testing and Test Control Notation Version 3 (TTCN-3)
- A strongly typed testing language used in conformance testing of communicating systems



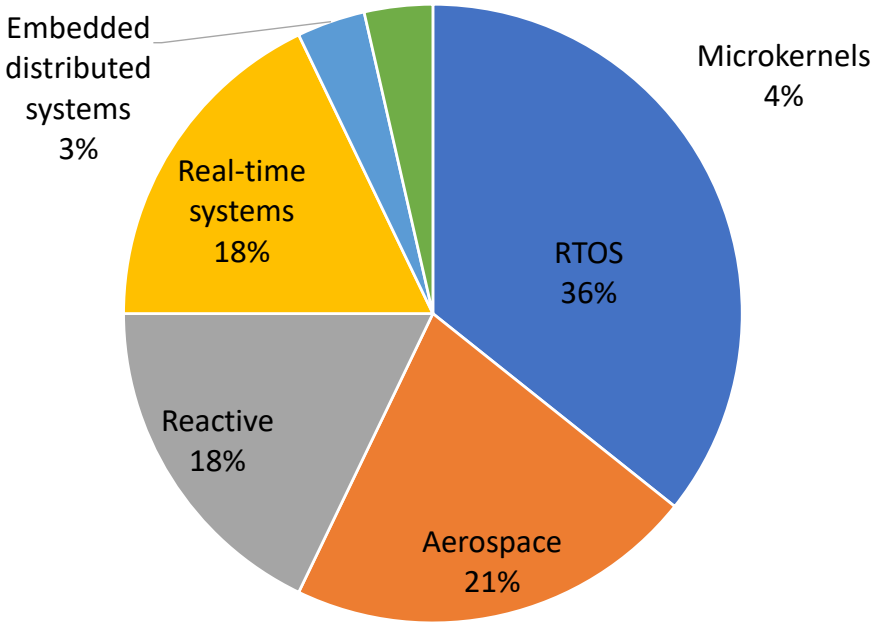
Communication systems (highlights)

- One of the least explored groups
- Session Initiation Protocol as the frequent case study
- Frequent use of TTCN-3
- **Prevalence of model-based approaches**

Distribution across time

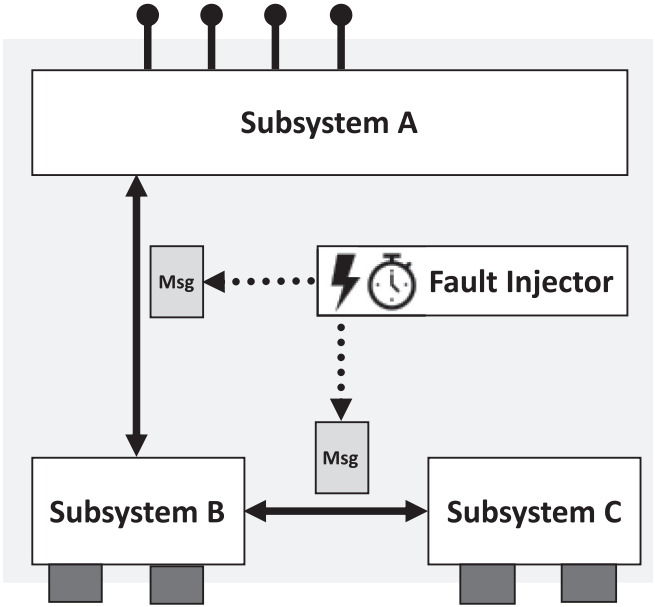


Embedded systems



- Strong focus on real-time
- Traditionally important in aerospace
- Increasingly important in autonomous systems

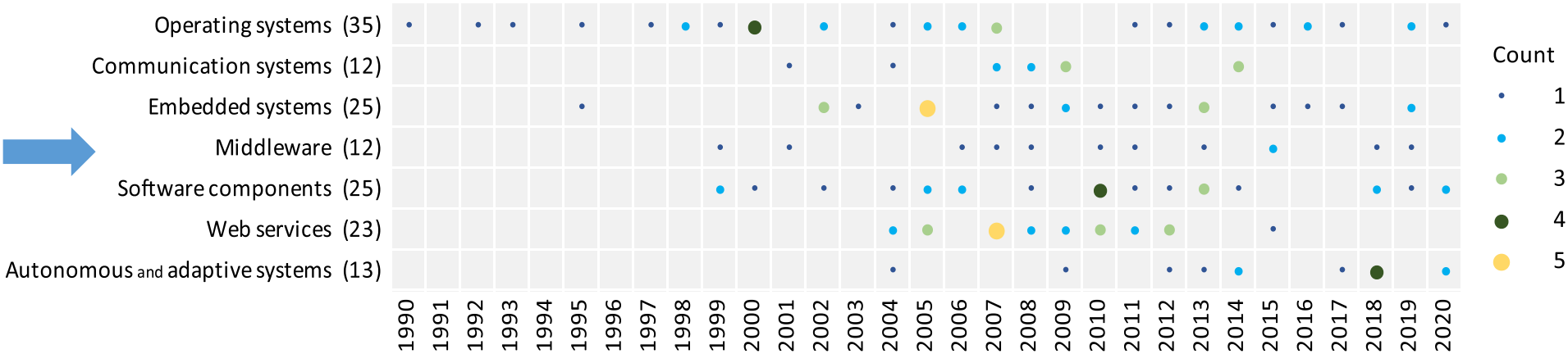
Evaluation of an embedded system



Embedded systems (highlights)

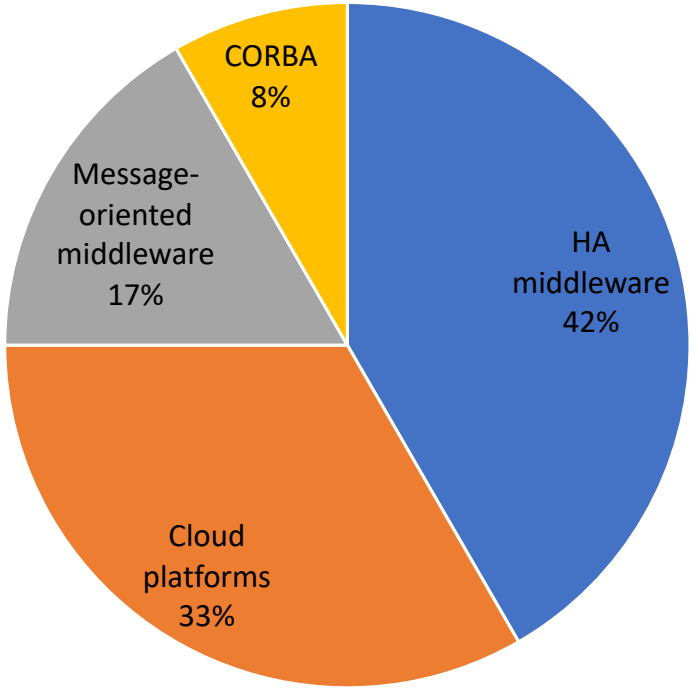
- Techniques used at very diverse abstraction levels
  - From system interfaces to processor registers
- Faults used are also quite diverse
  - Interface parameter mutations (e.g., invalid or boundary values)
  - bit-flips on processor registers
  - message-level faults (e.g., reordering messages)
  - timing faults.

Distribution across time

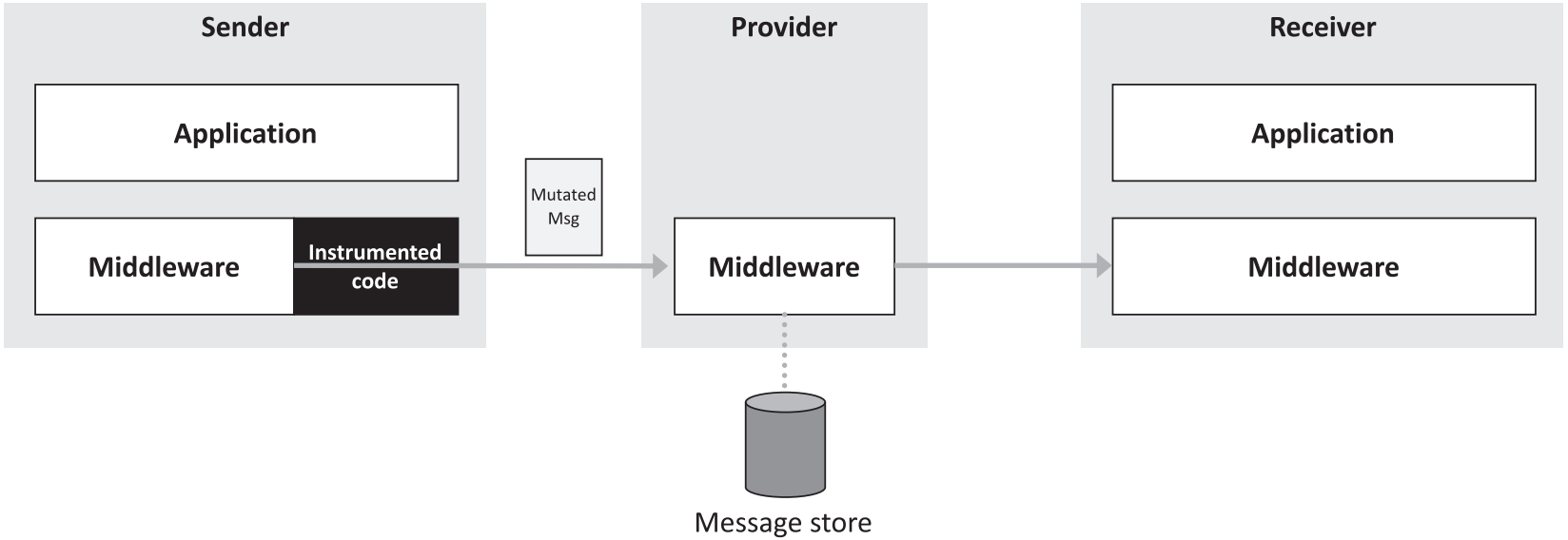


Middleware

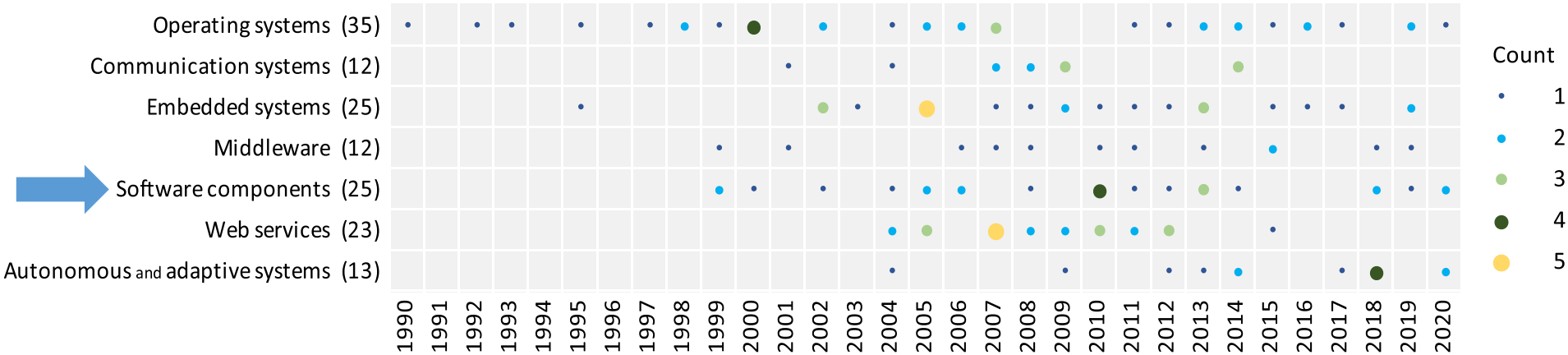
- Heterogeneous category
- Mainstream (CORBA, JMS, DDS)
- High availability middleware/architecture
- Management platforms (cloud)



MOM example

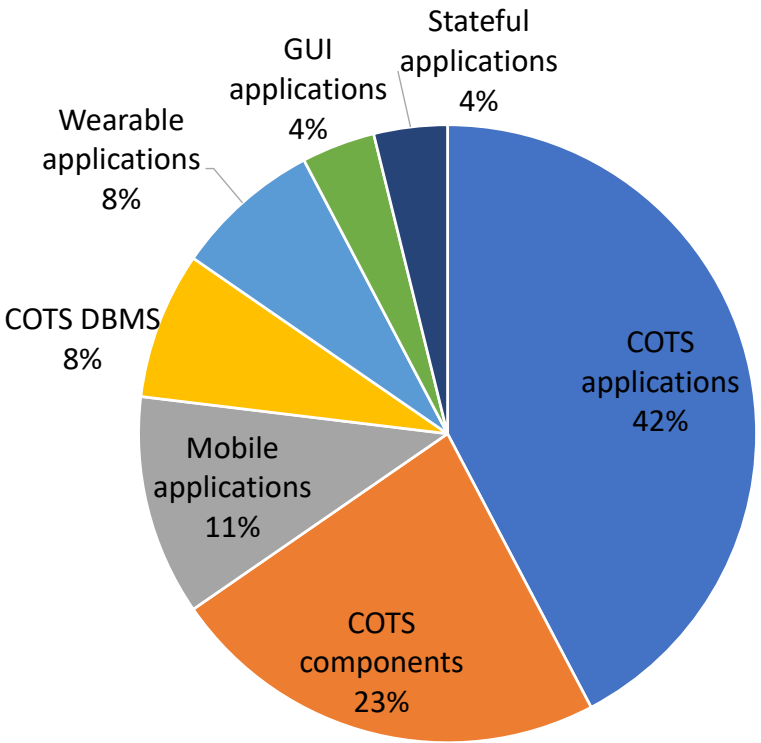

Sender
Application
Middleware
Instrumented code
Mutated Msg
Provider
Middleware
Receiver
Application
Middleware
Message store

Distribution across time



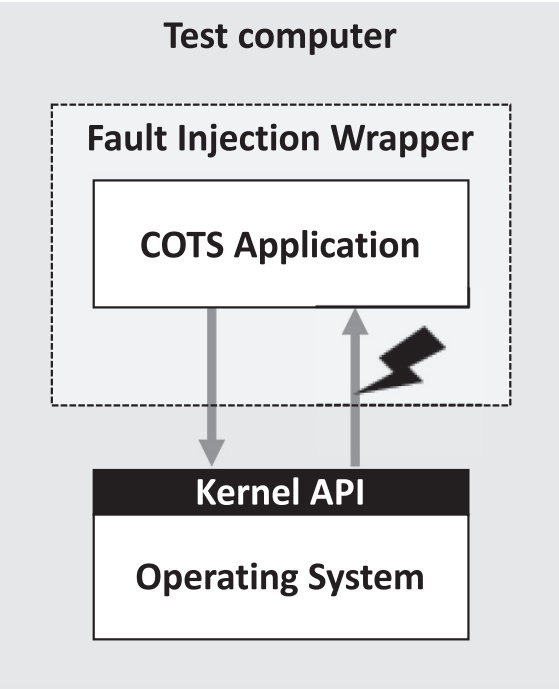
Software components

- COTS applications and components
- Mobile and wearables

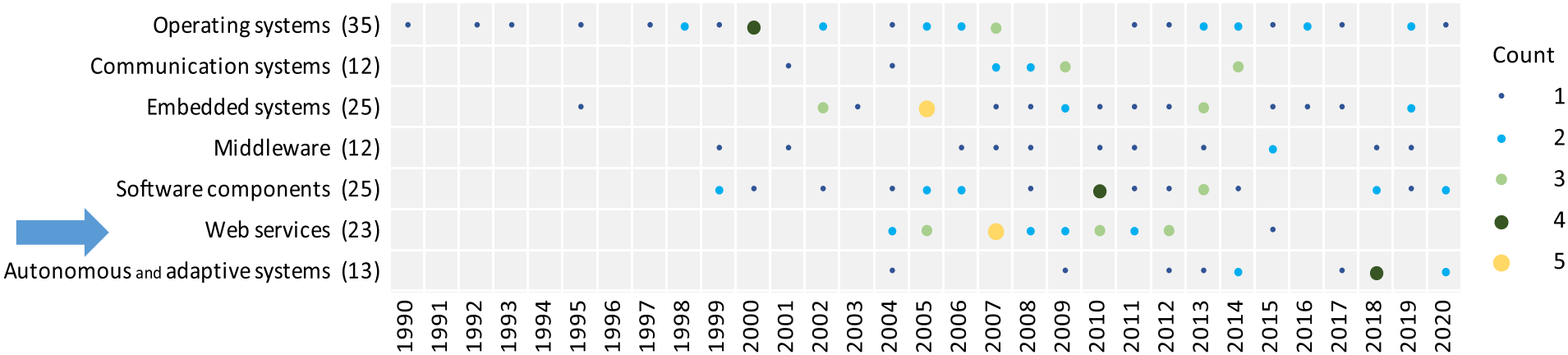


Software components example

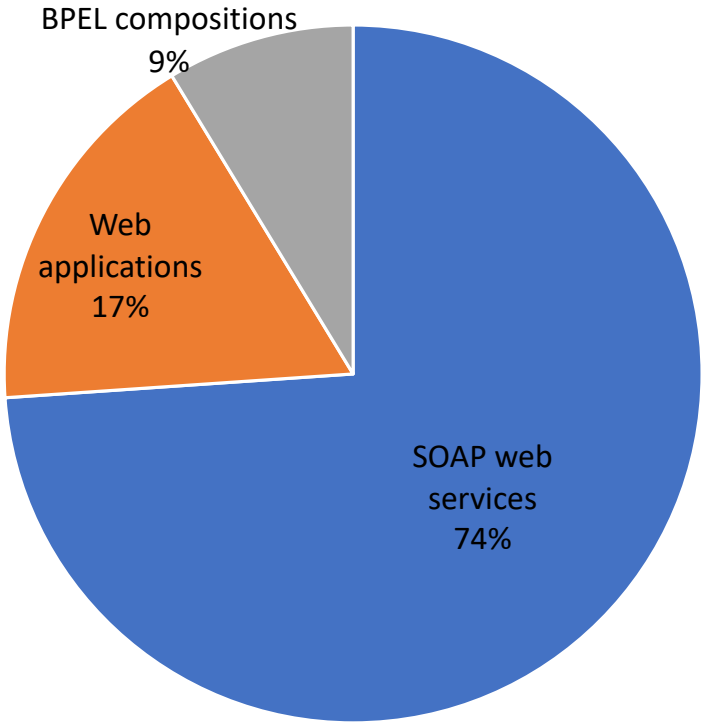
- Fault injection as common technique
- Code changes injection second main technique
- API calls and also machine code as targets
- Invalid and random inputs



Distribution across time

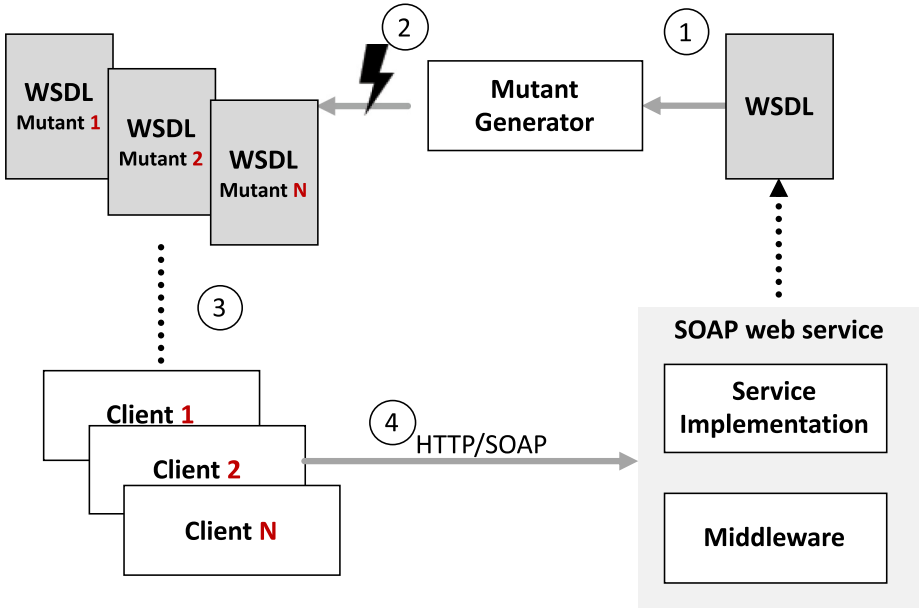


Web services

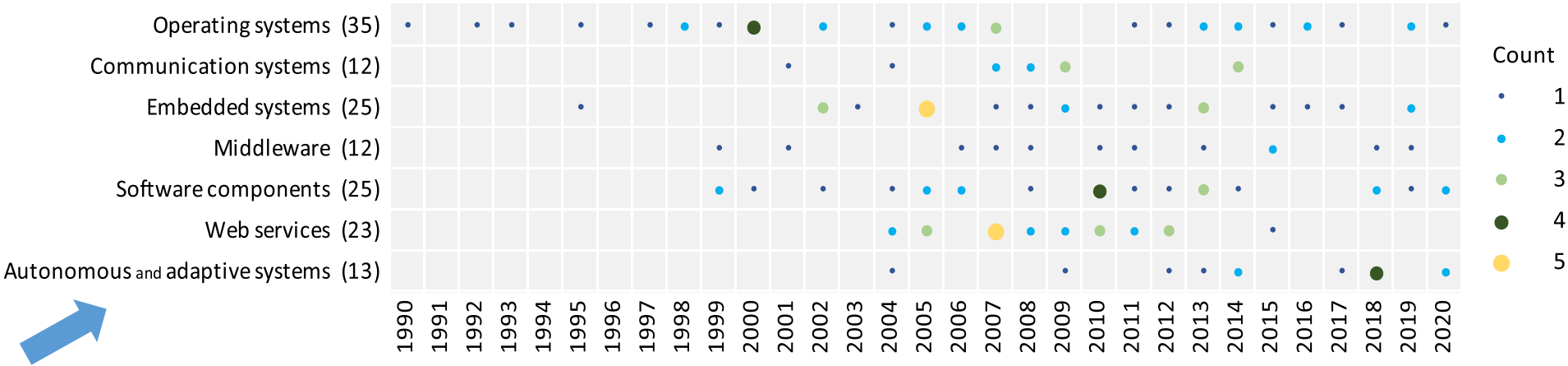


- Strong empahsis on SOAP web services
- Targets of the techniques now also set on the interface description
- Several cases also focused on delivering tools
- Fault injection with invalid inputs over message fields
- 2015 as the last year of work on SOAP
- Research on REST is rising

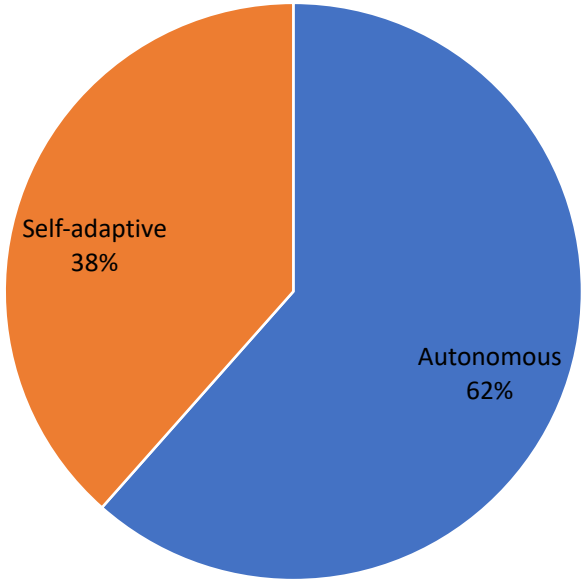
Web services - example



Distribution across time

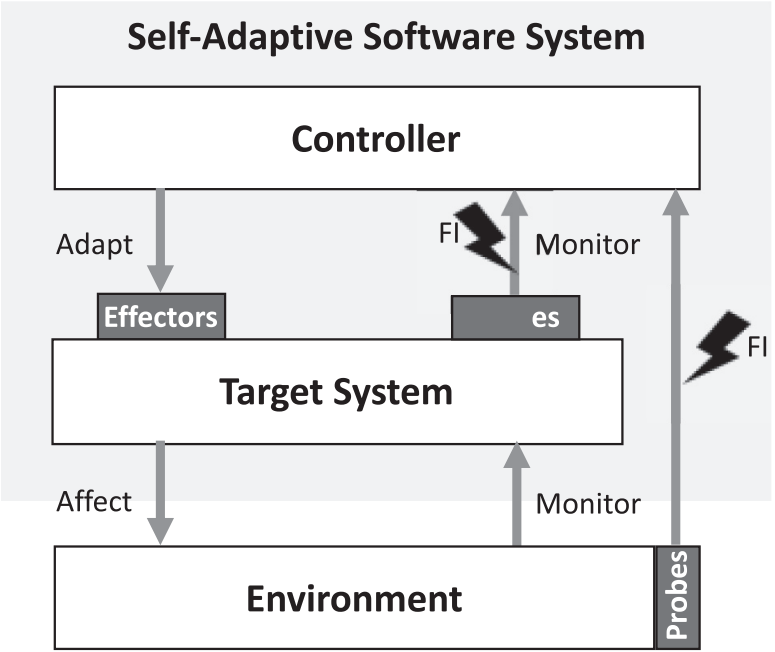


Autonomous and adaptive systems



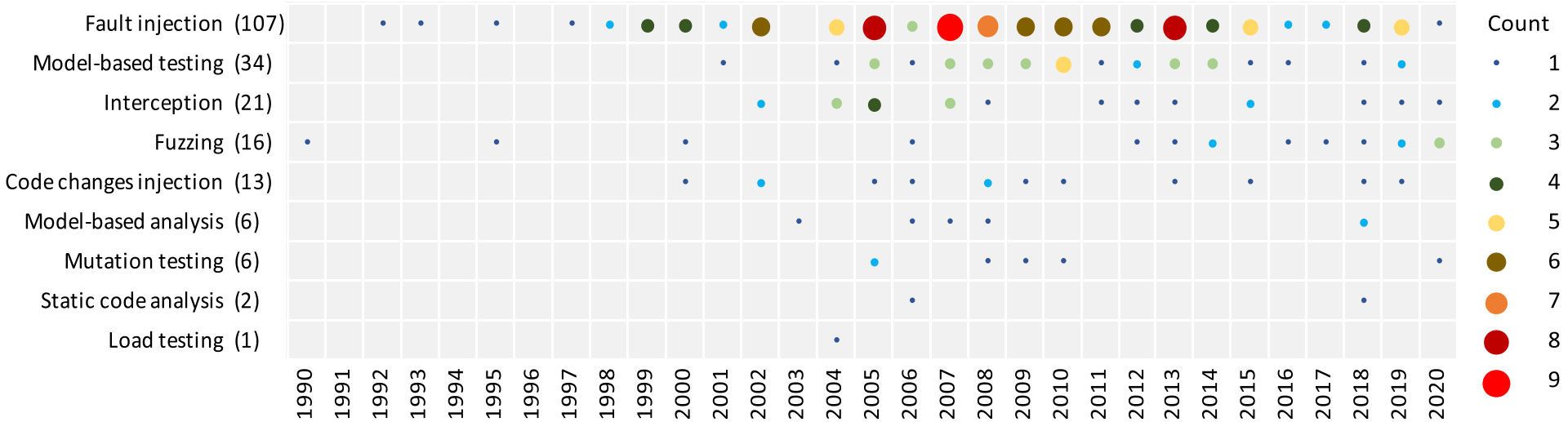
Autonomous and adaptive - example

- Stateless / stateful
- Typical faults, but timing and MACD also relevant
- Increasing presence of research in autonomous systems

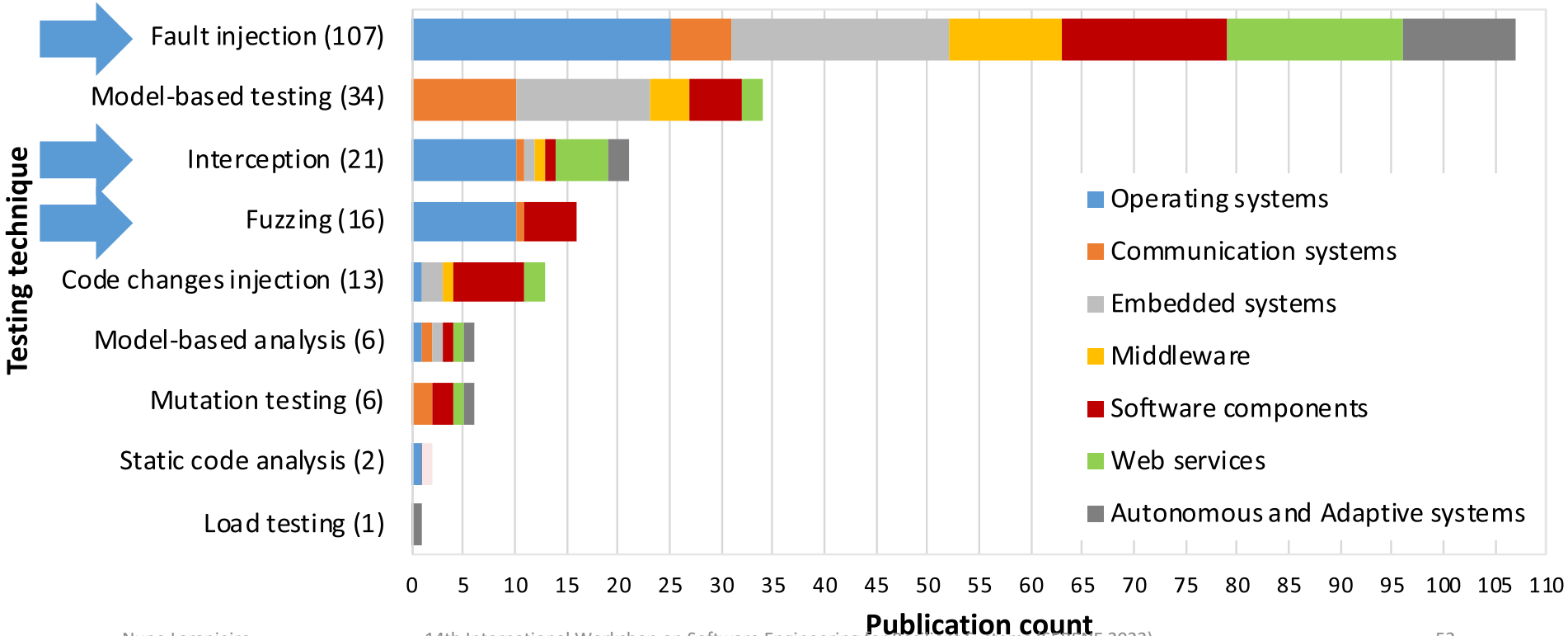


Which techniques are being used to assess robustness?

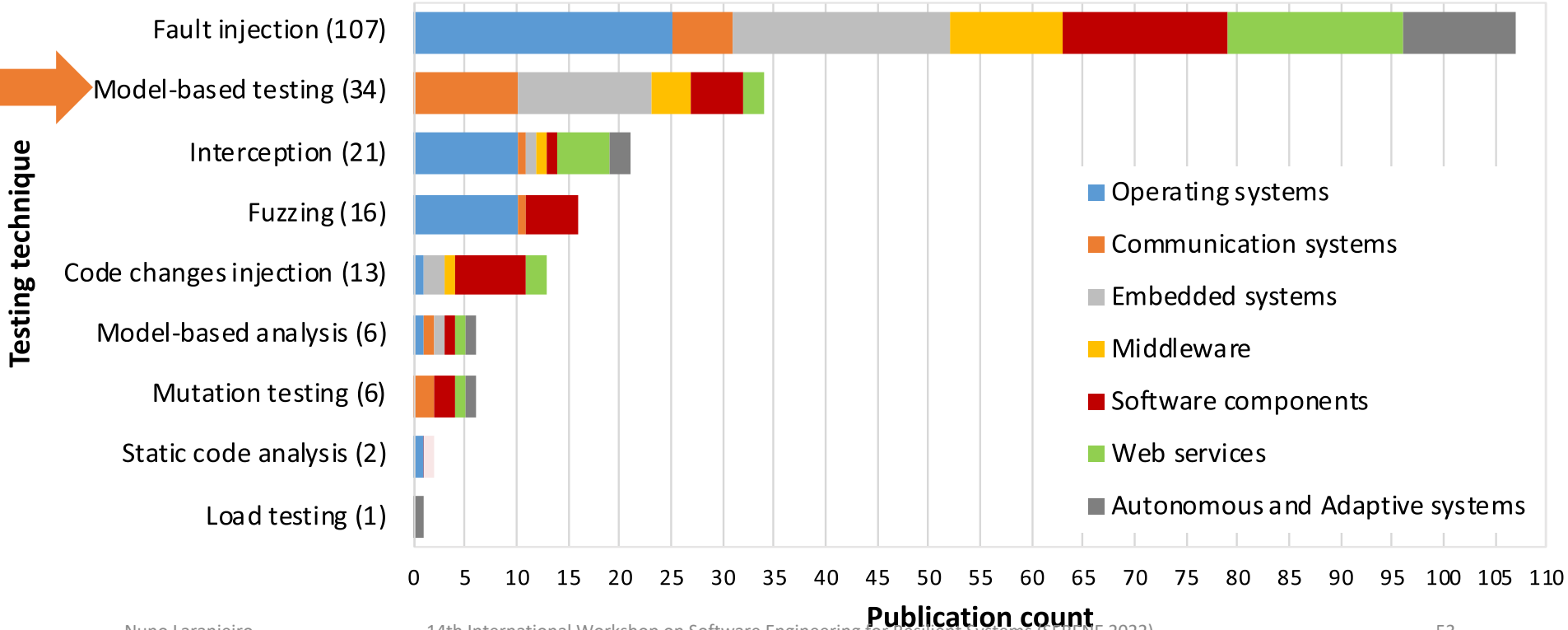
Techniques distribution



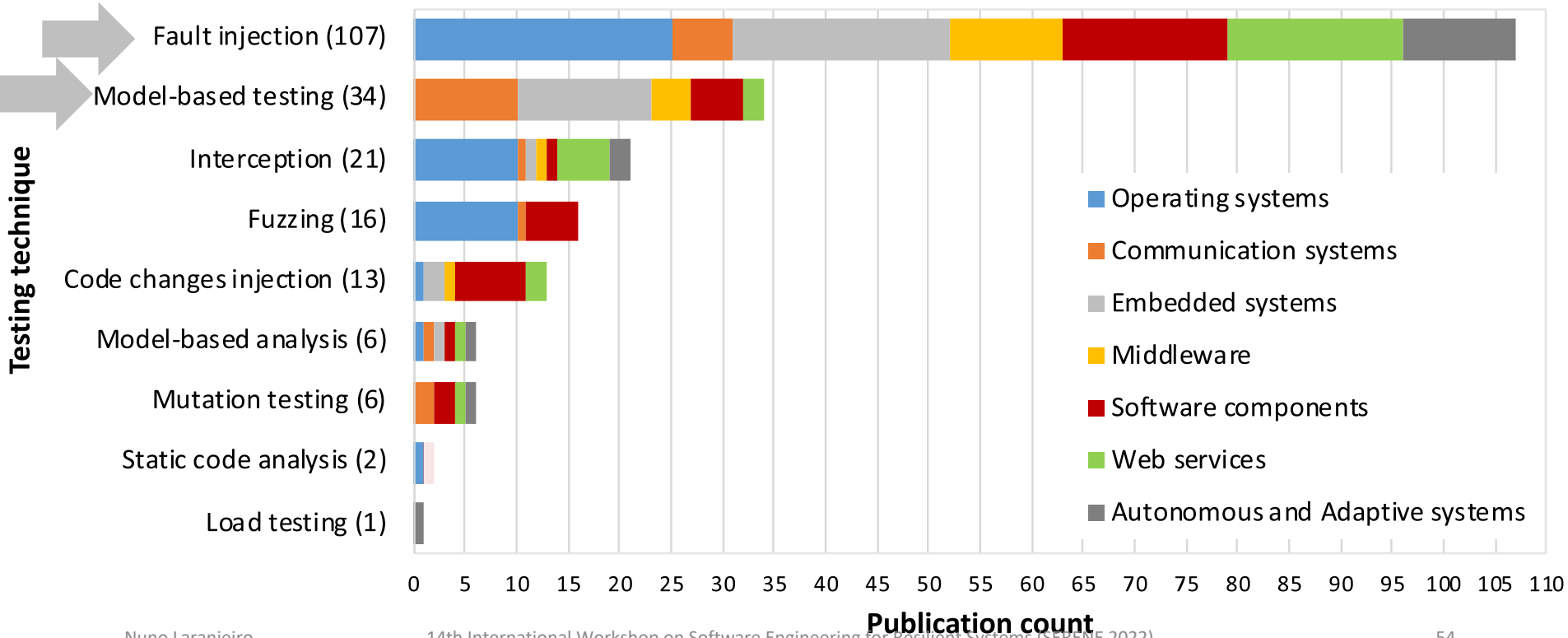
Techniques – Operating systems



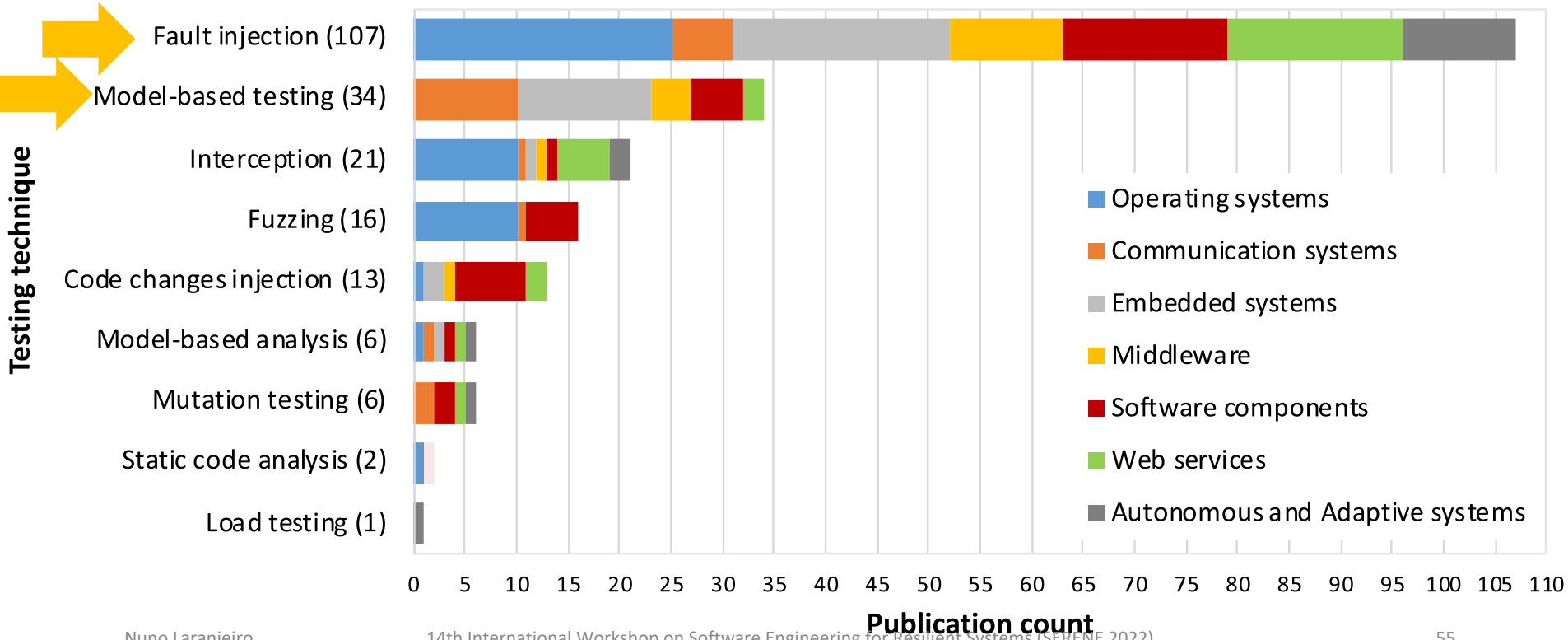
Techniques – Communication systems



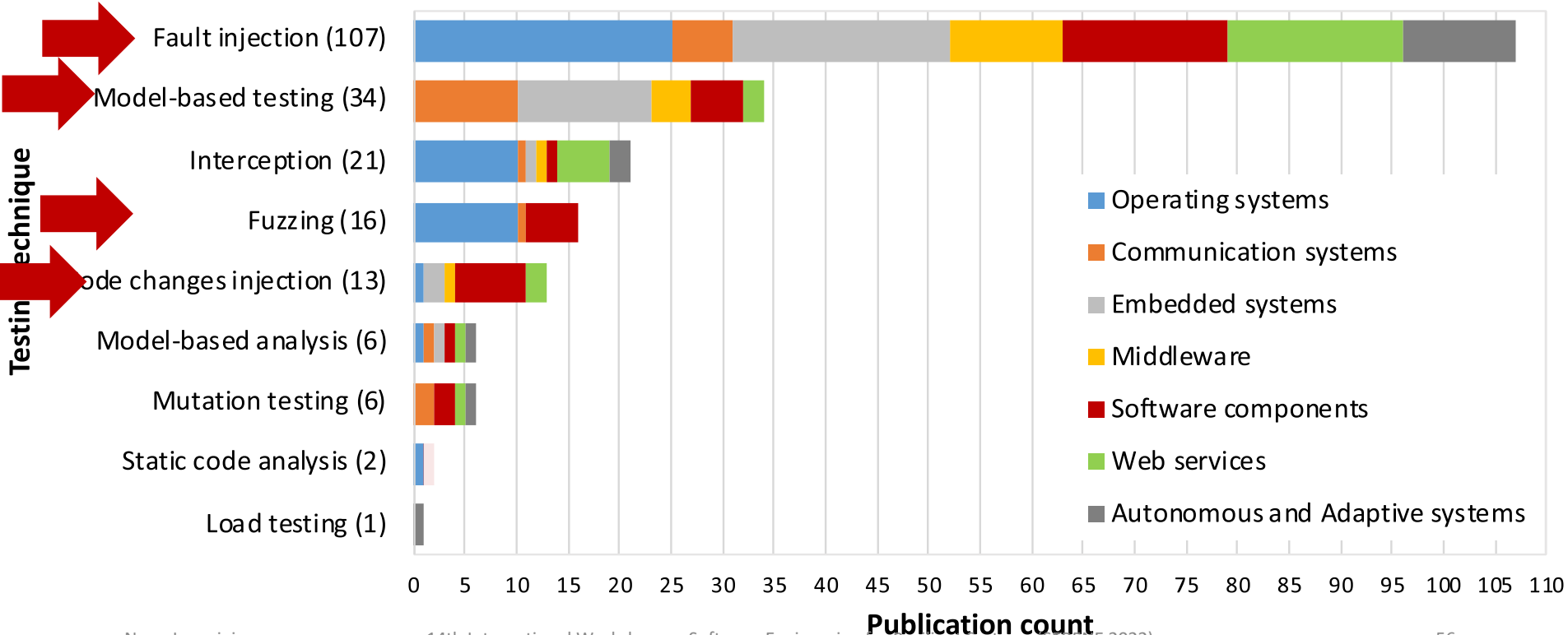
Techniques – Embedded systems



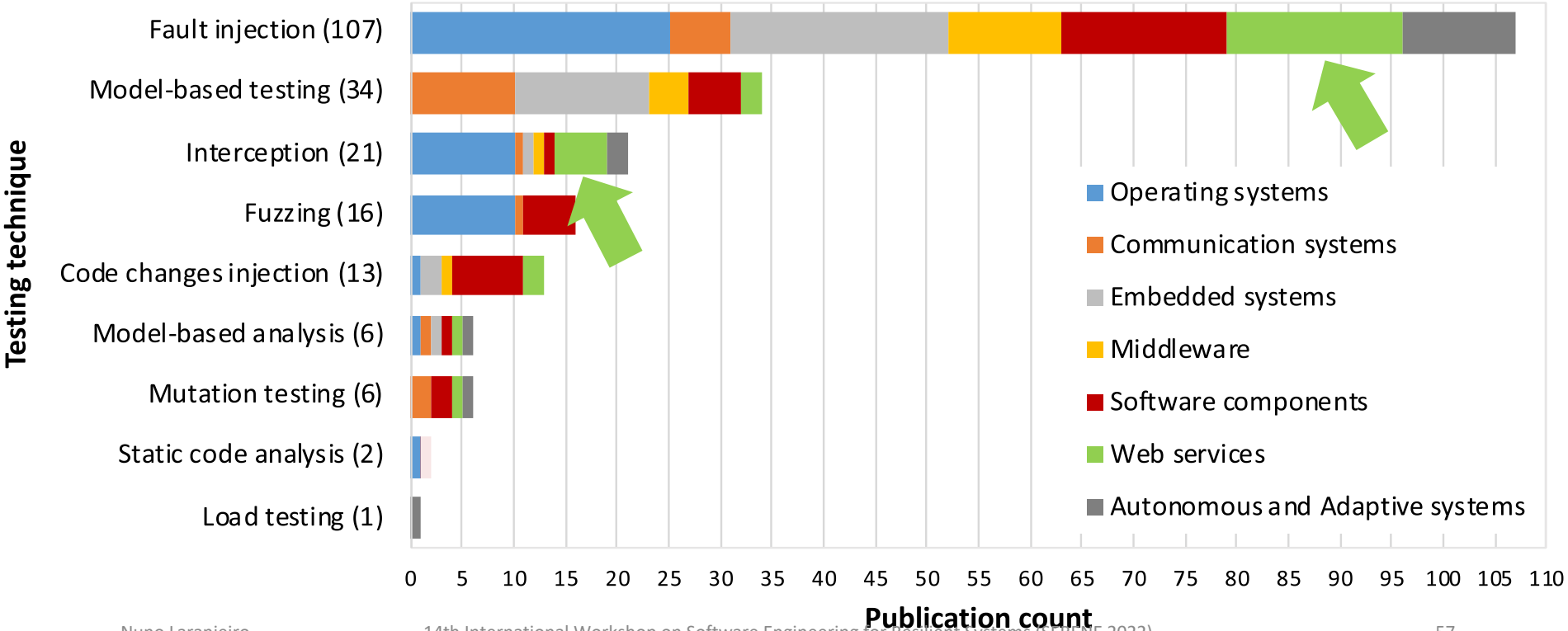
Techniques – Middleware



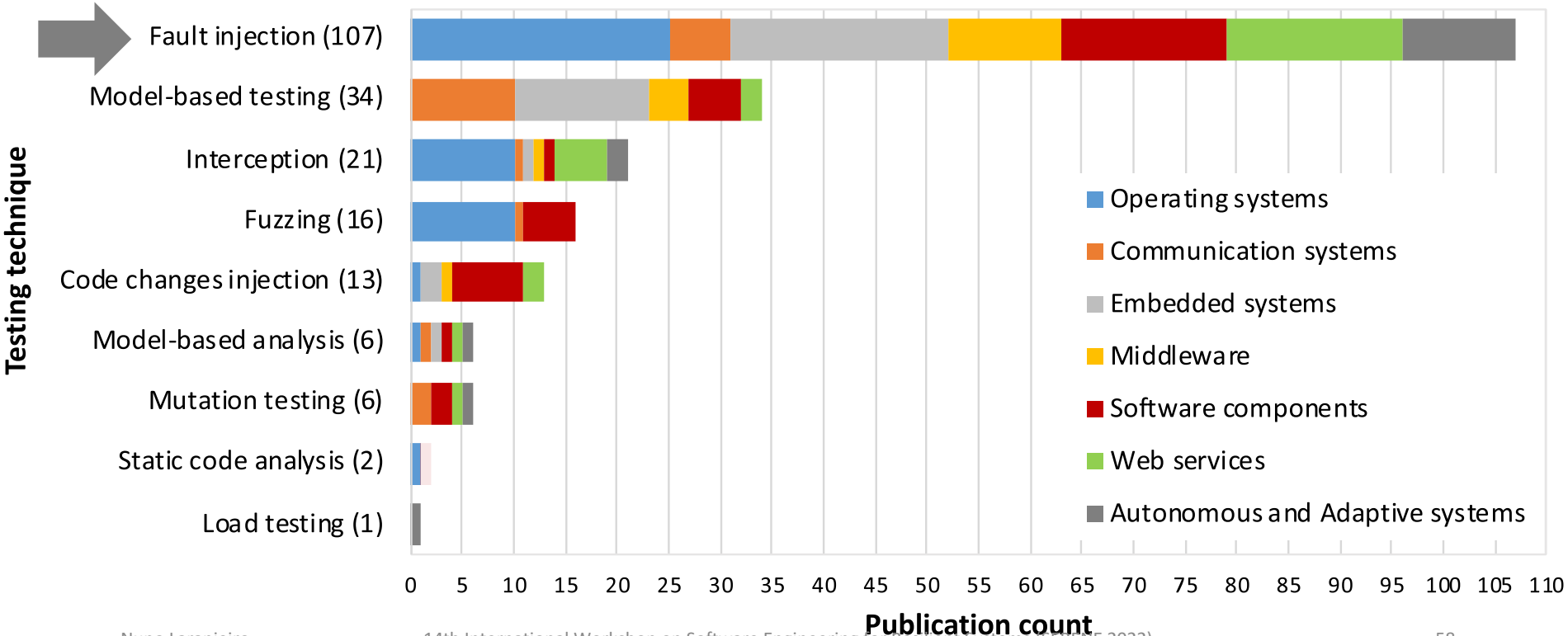
Techniques – Software components



Techniques – Web services



Techniques – Autonomous and adaptive



About the techniques…

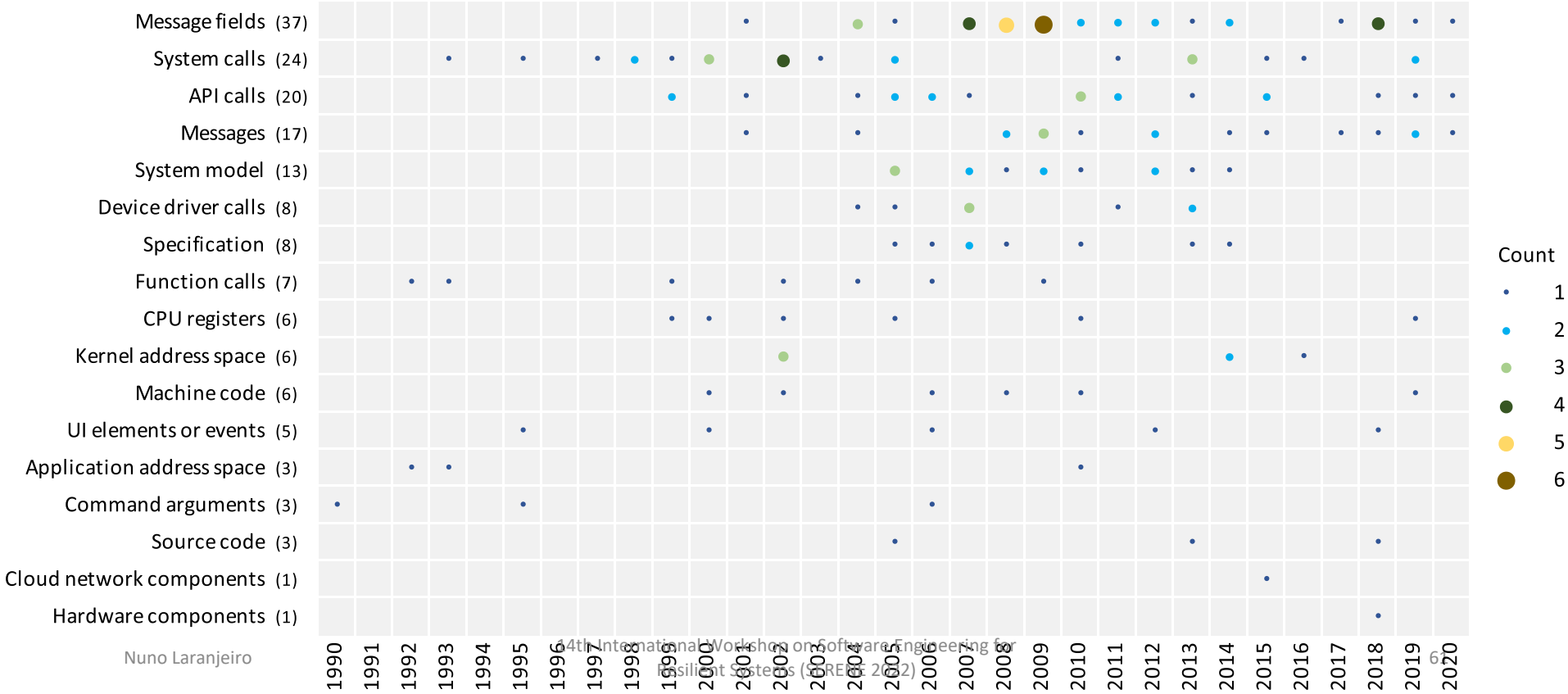
- Mostly experimental (3/4 uses fault injection)

- There is some coupling of techniques to certain types of systems:
  - Formal techniques → embedded and communication systems
  - Code changes injection → Software components
  - Fuzzing → operating systems

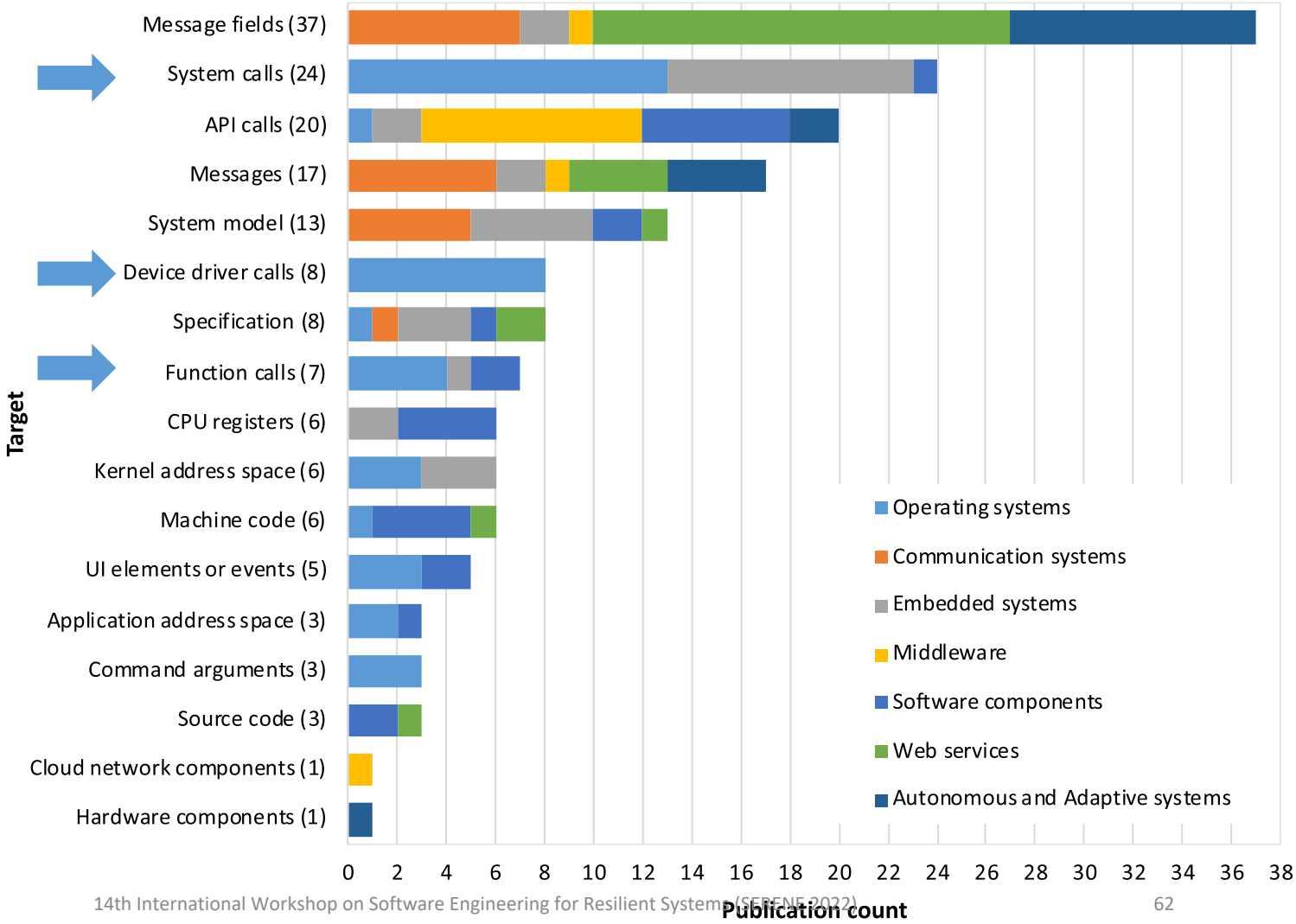
- Certain combinations were not observed
  - e.g., code changes injection and autonomous and adaptive systems

Which are the targets of robustness evaluation techniques?

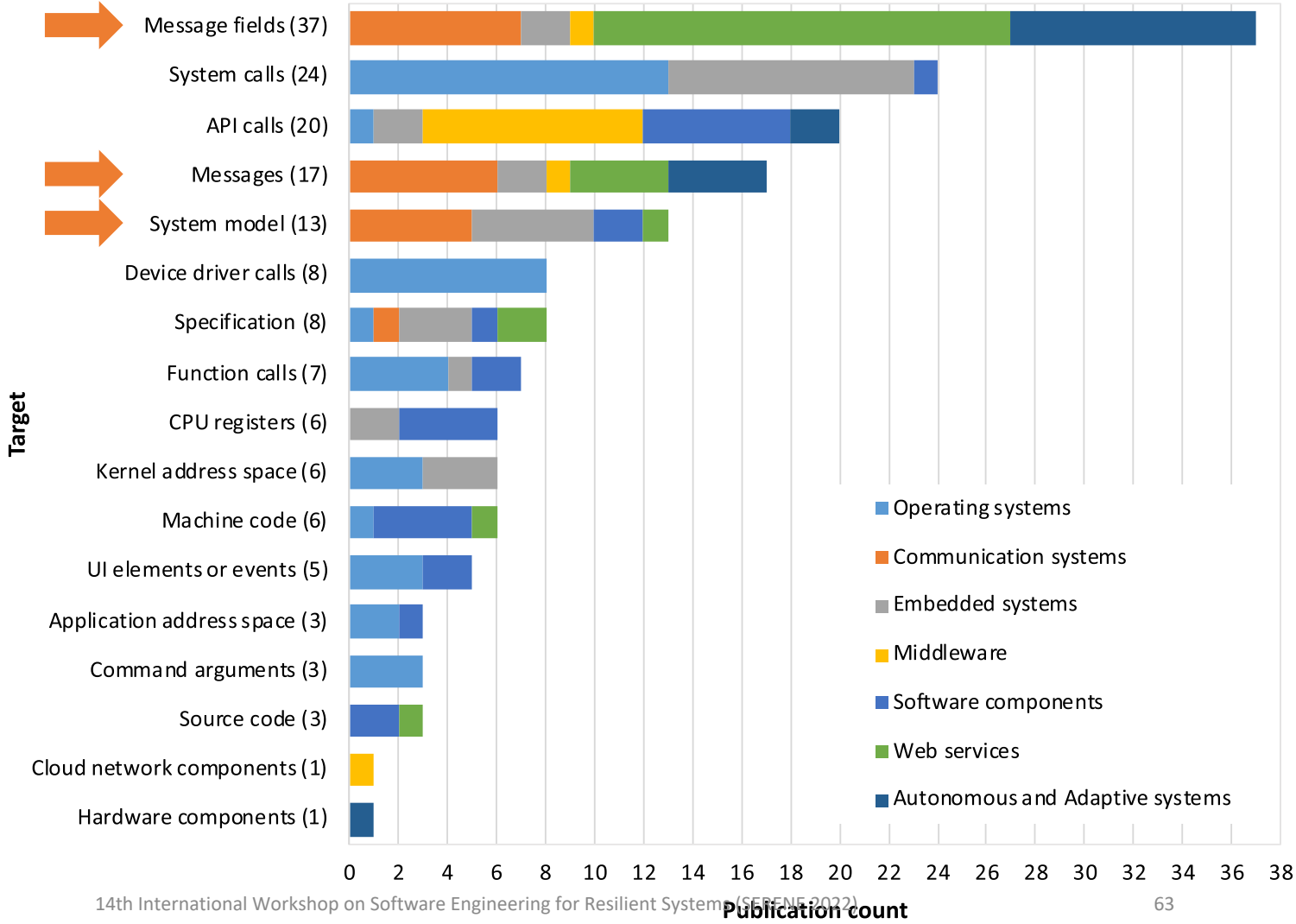
Technique target - distribution



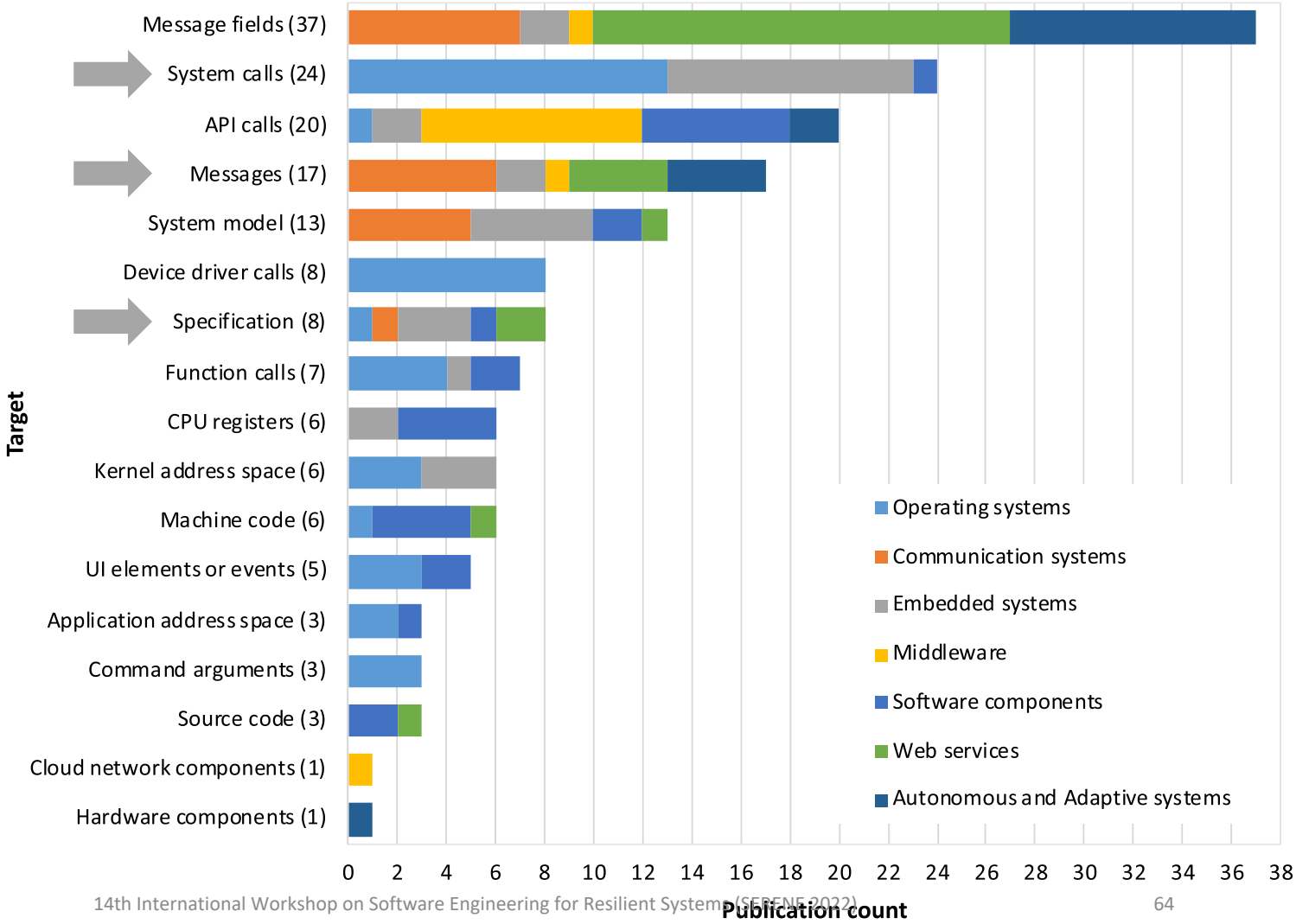
Technique target
Operating systems



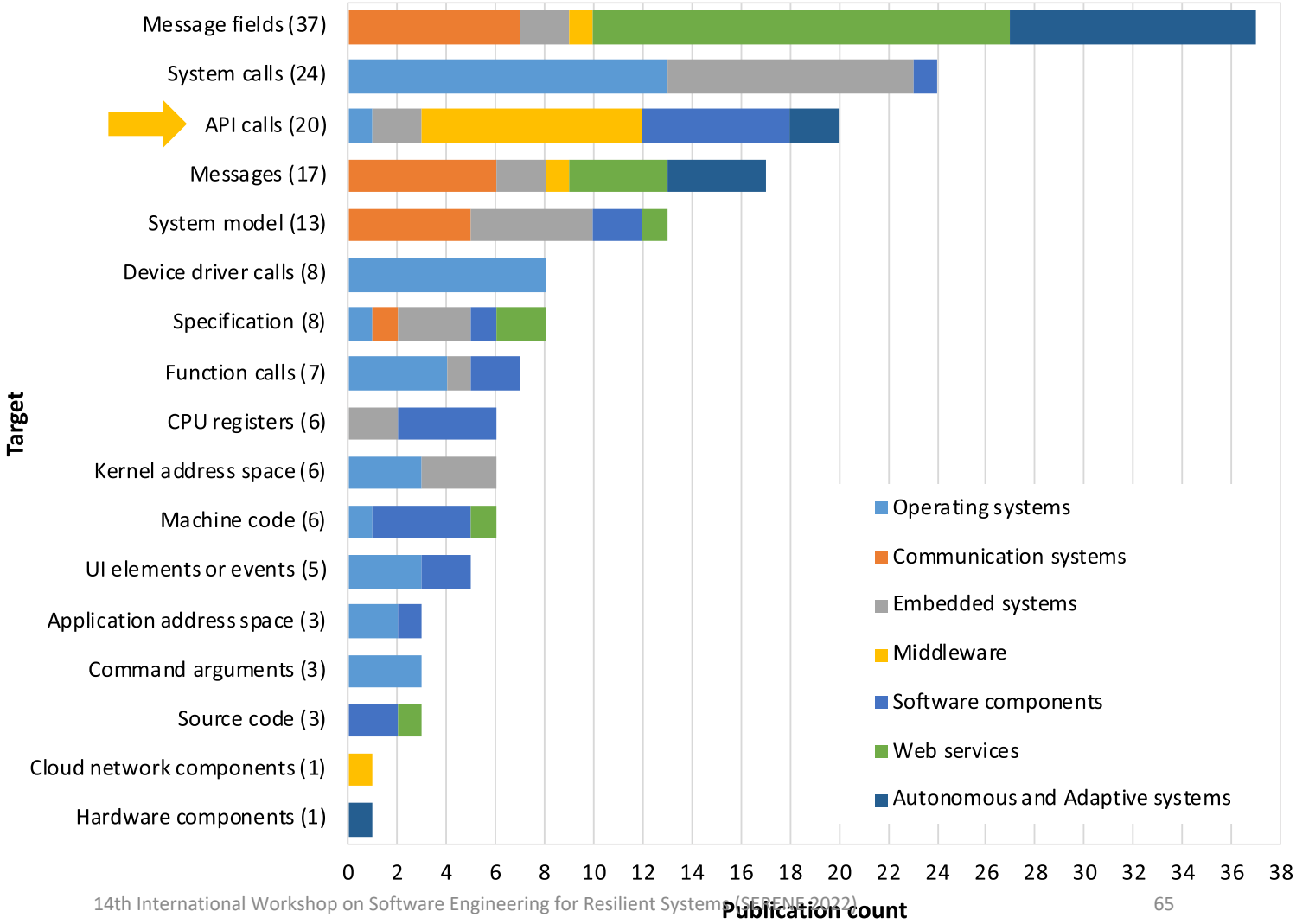
Technique target **Communication systems**



Technique target
Embedded systems

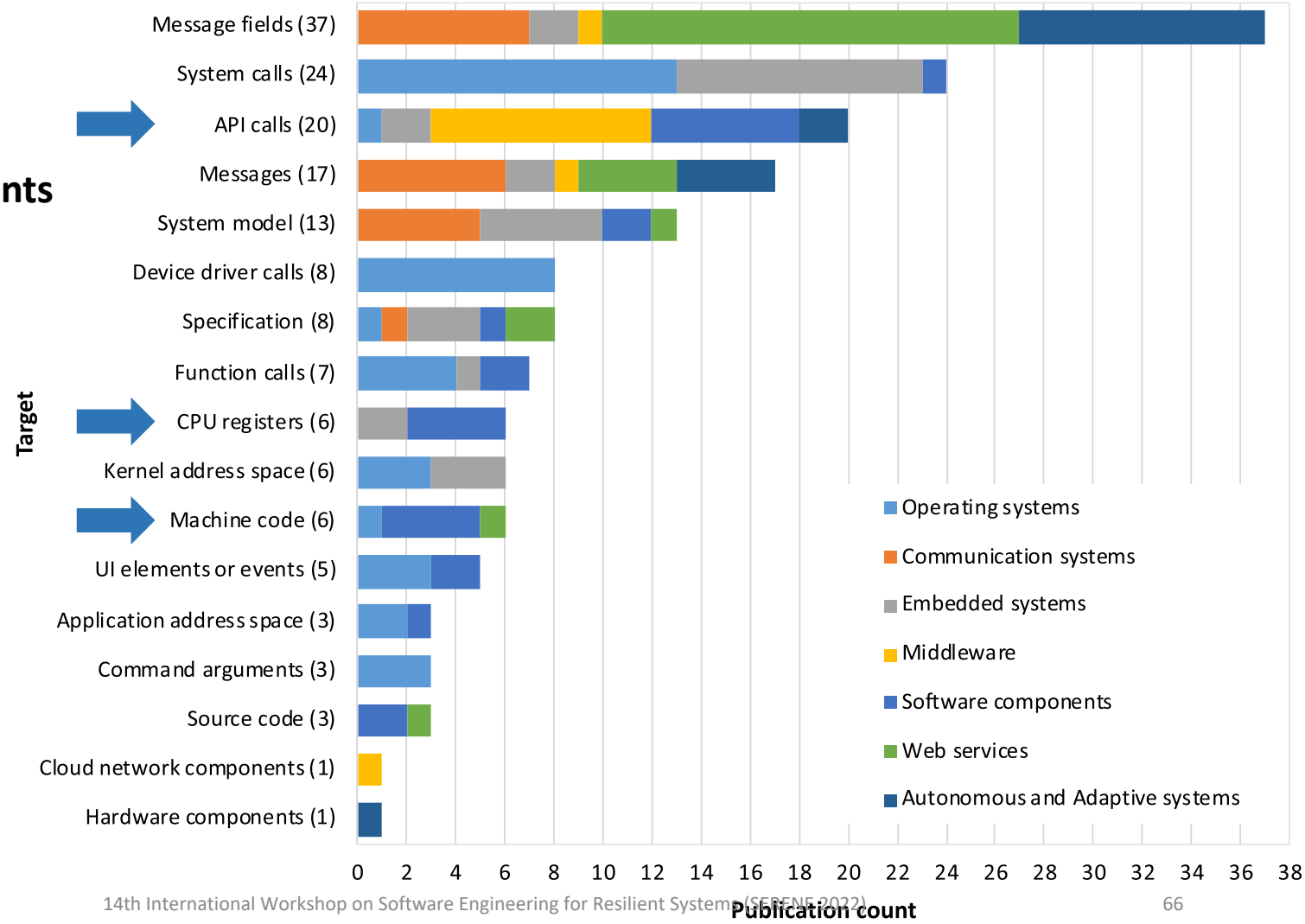


Technique target
Middleware

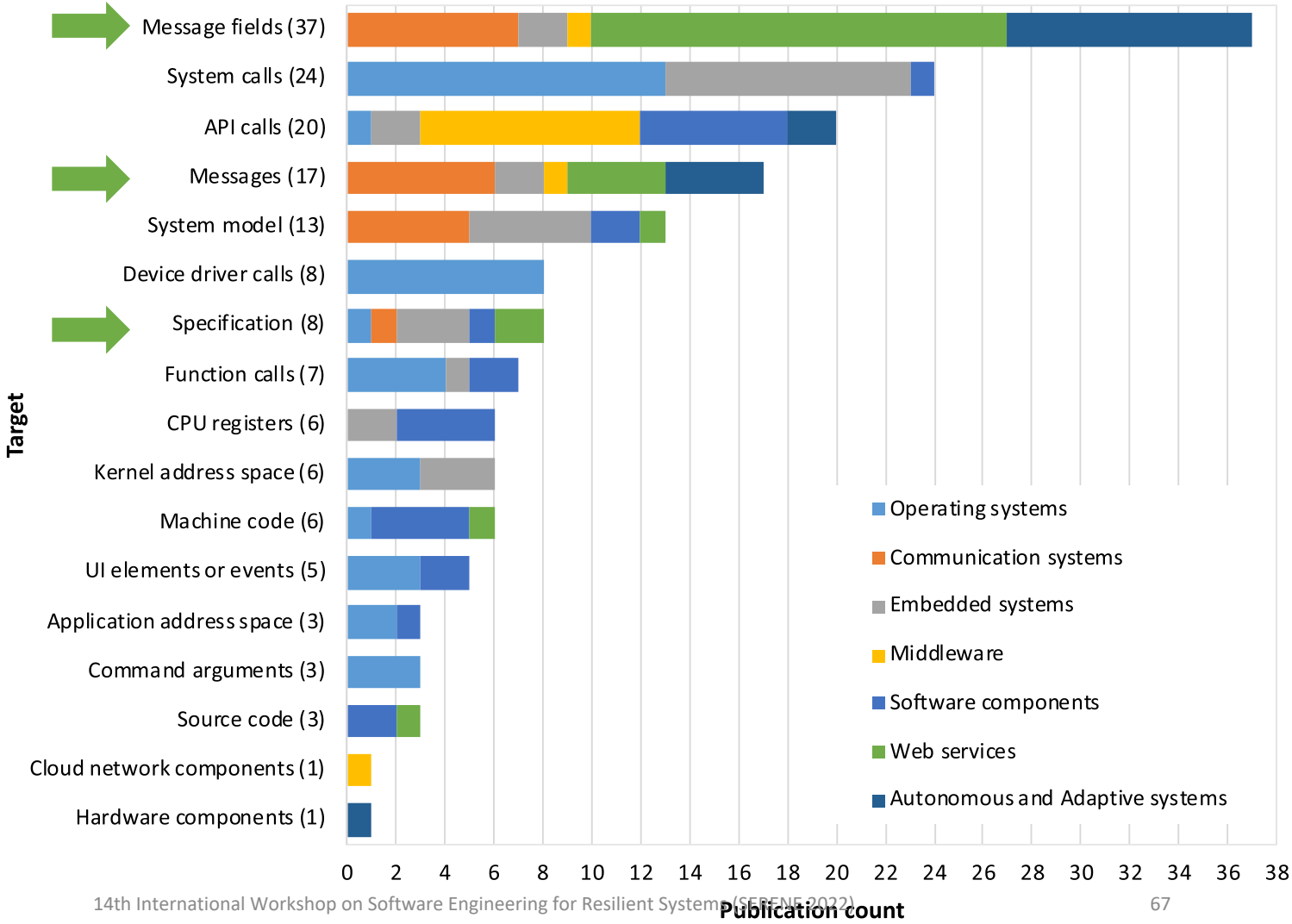


Technique target

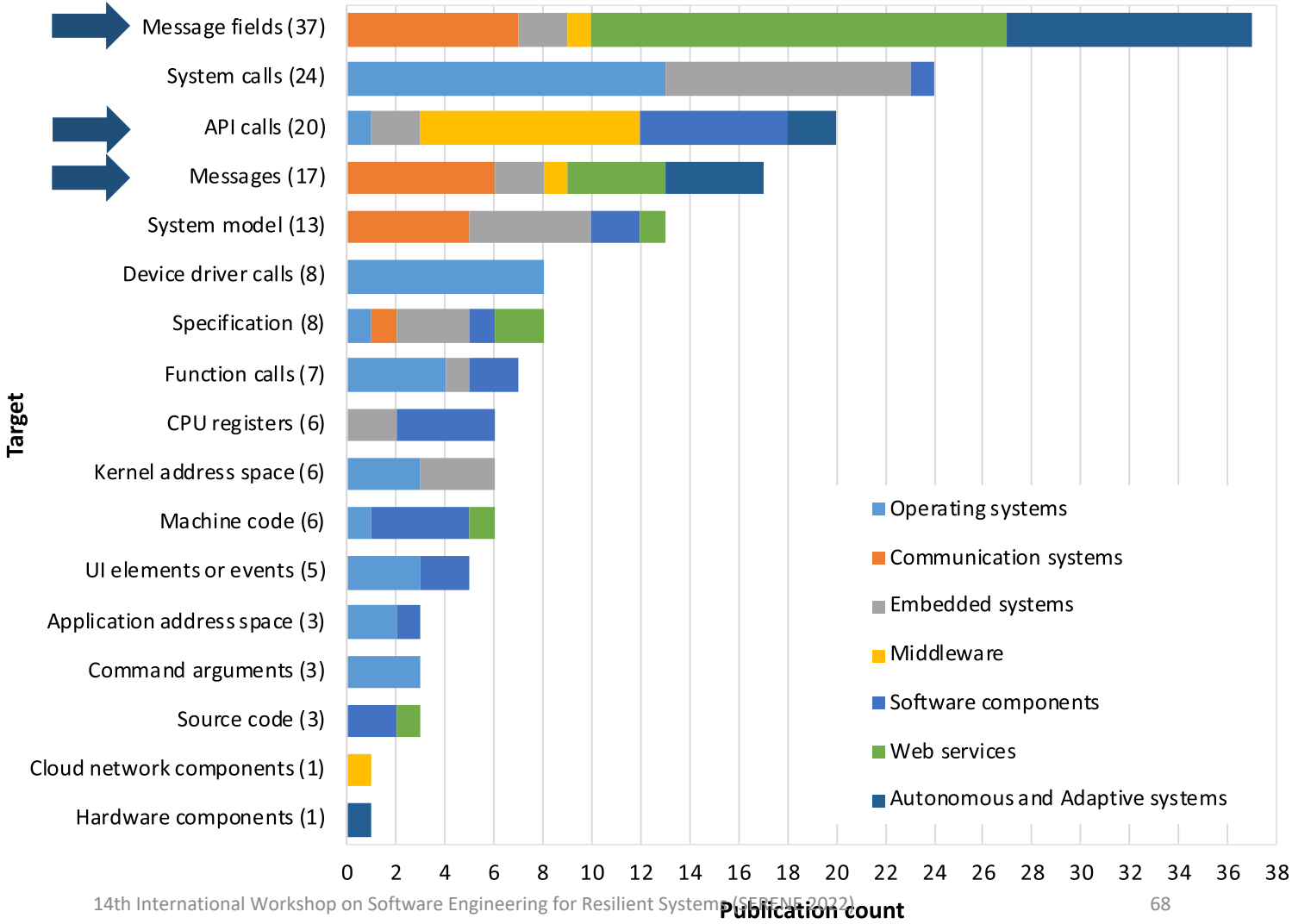
Software components



Technique target
Web services



Technique target
Autonomous and adaptive systems



About the targets…

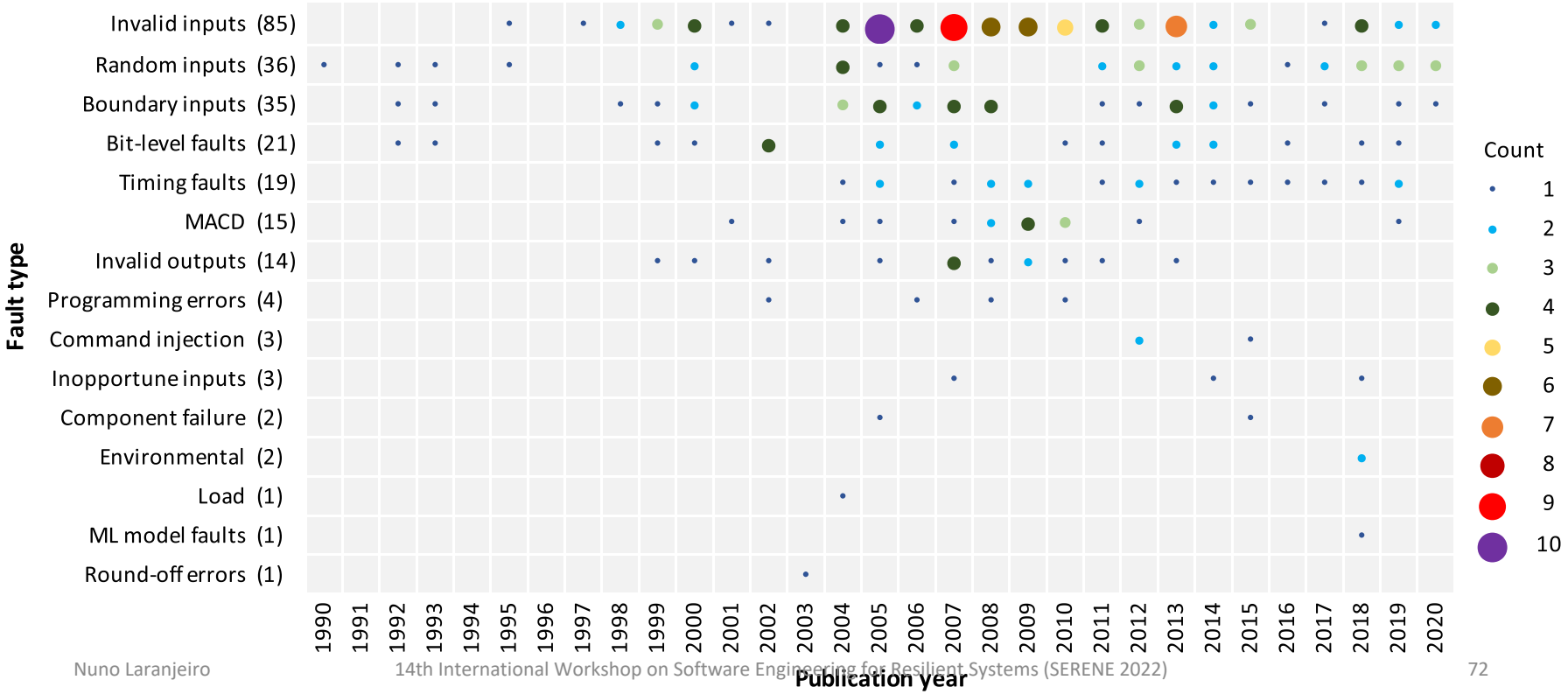
- 5 targets dominate the distribution
  - Message fields
  - System calls
  - Messages
  - API calls
  - System model
- System calls and API calls of frequent usage and distributed along time
- Message and message fields also distributed, but with a more gaps

About the targets (coupling)

- System calls and kernel address space → operating systems and embedded systems
- Messages and fields → WS, communication systems, and AA
  - Exploration of the client-server decoupling
- System model → communication systems and embedded systems
  - Modelling necessary to prove critical properties of the system
- API calls → middleware and Software components
  - Main entry points of this type of software

Which types of faults are being used in software robustness evaluation?

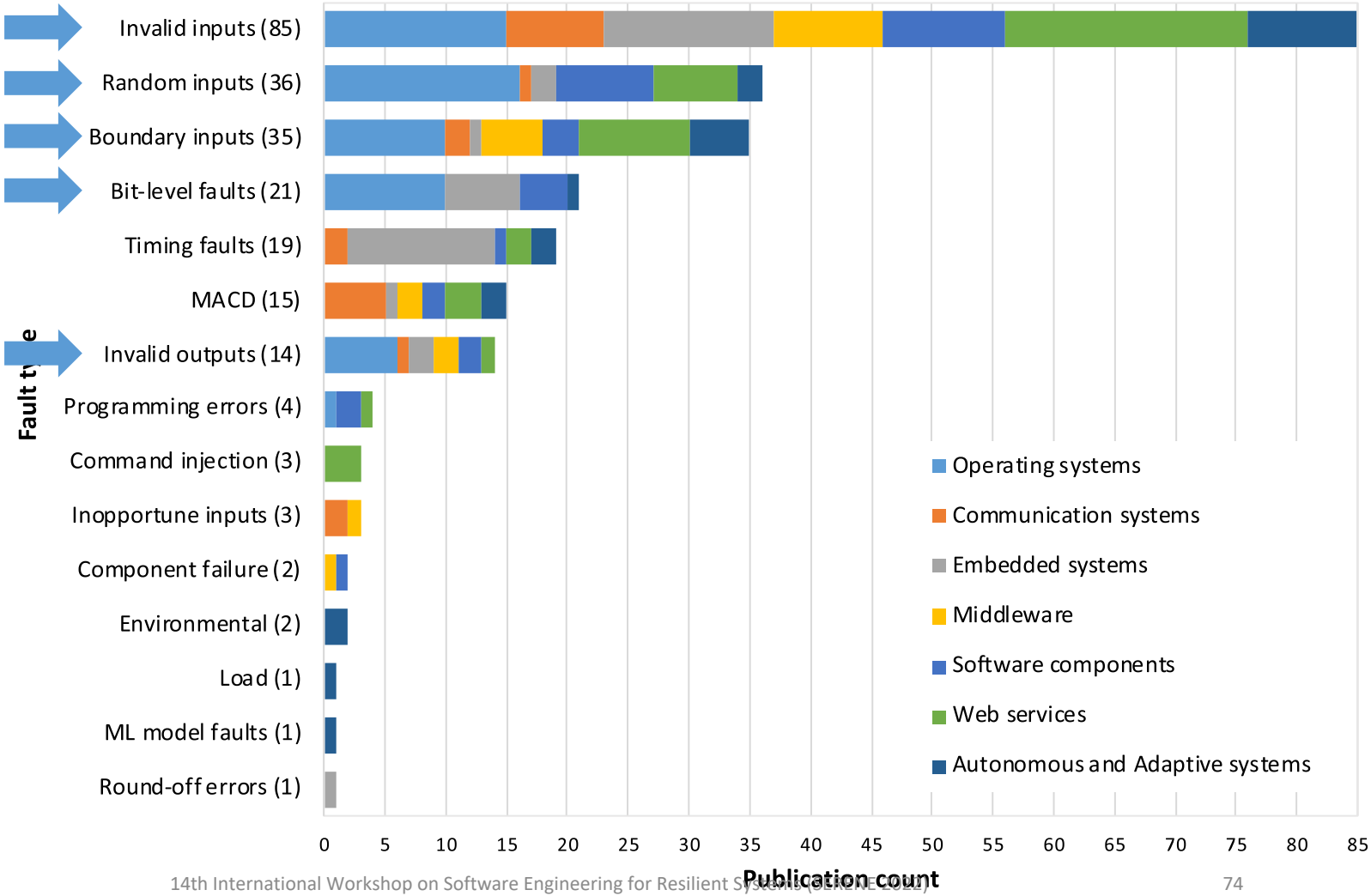
Types of Faults - distribution



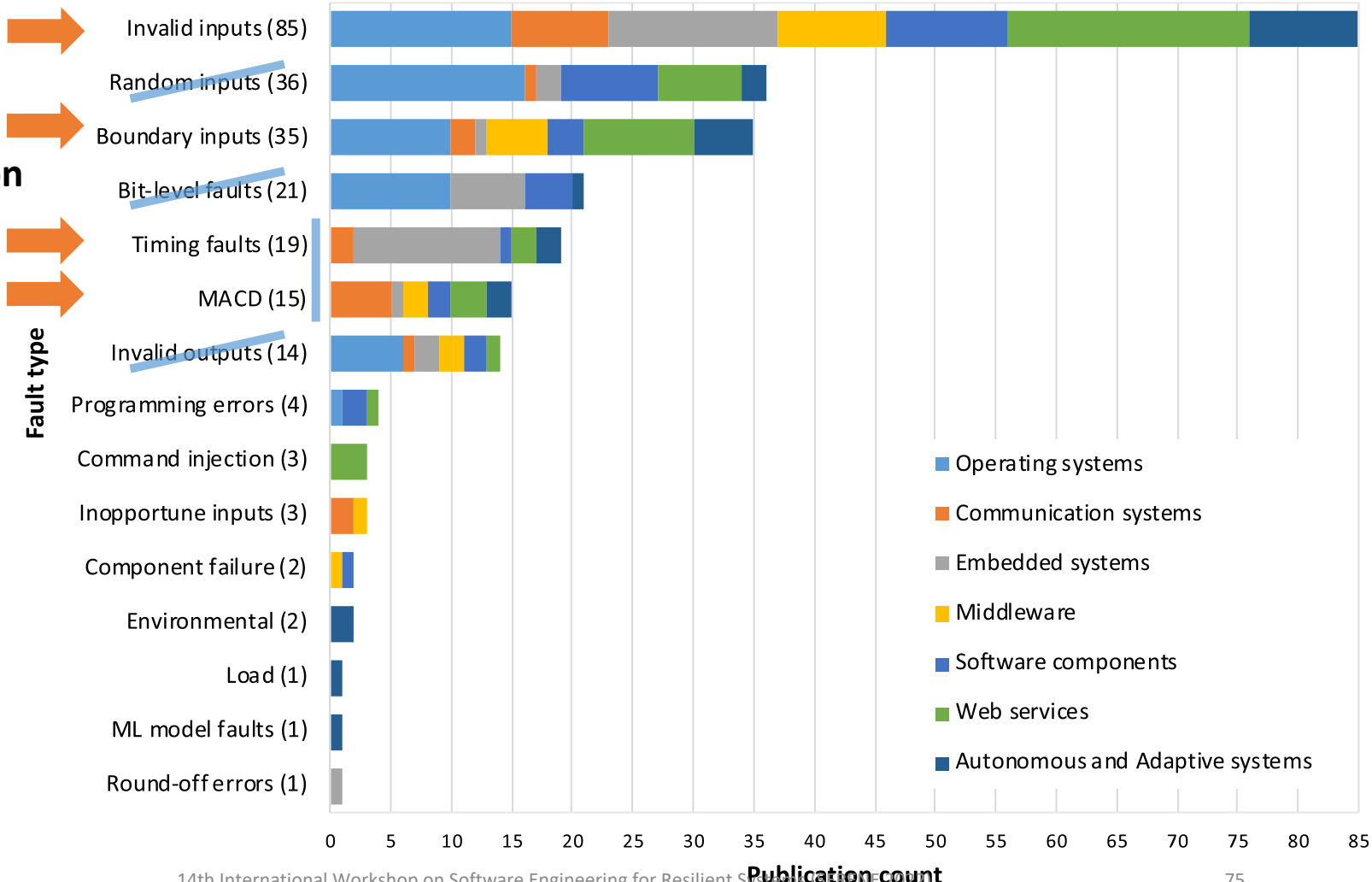
Types of faults

- Invalid inputs dominate the distribution
- Also popular: random and boundary inputs, bit-level faults, timing faults, MACD operations, invalid outputs
- Distribution over time
  - invalid inputs, MACD and invalid outputs concentrating in the late 2000's
  - boundary, random used regularly, bit-level also
  - Timing faults used regularly since the 2000's

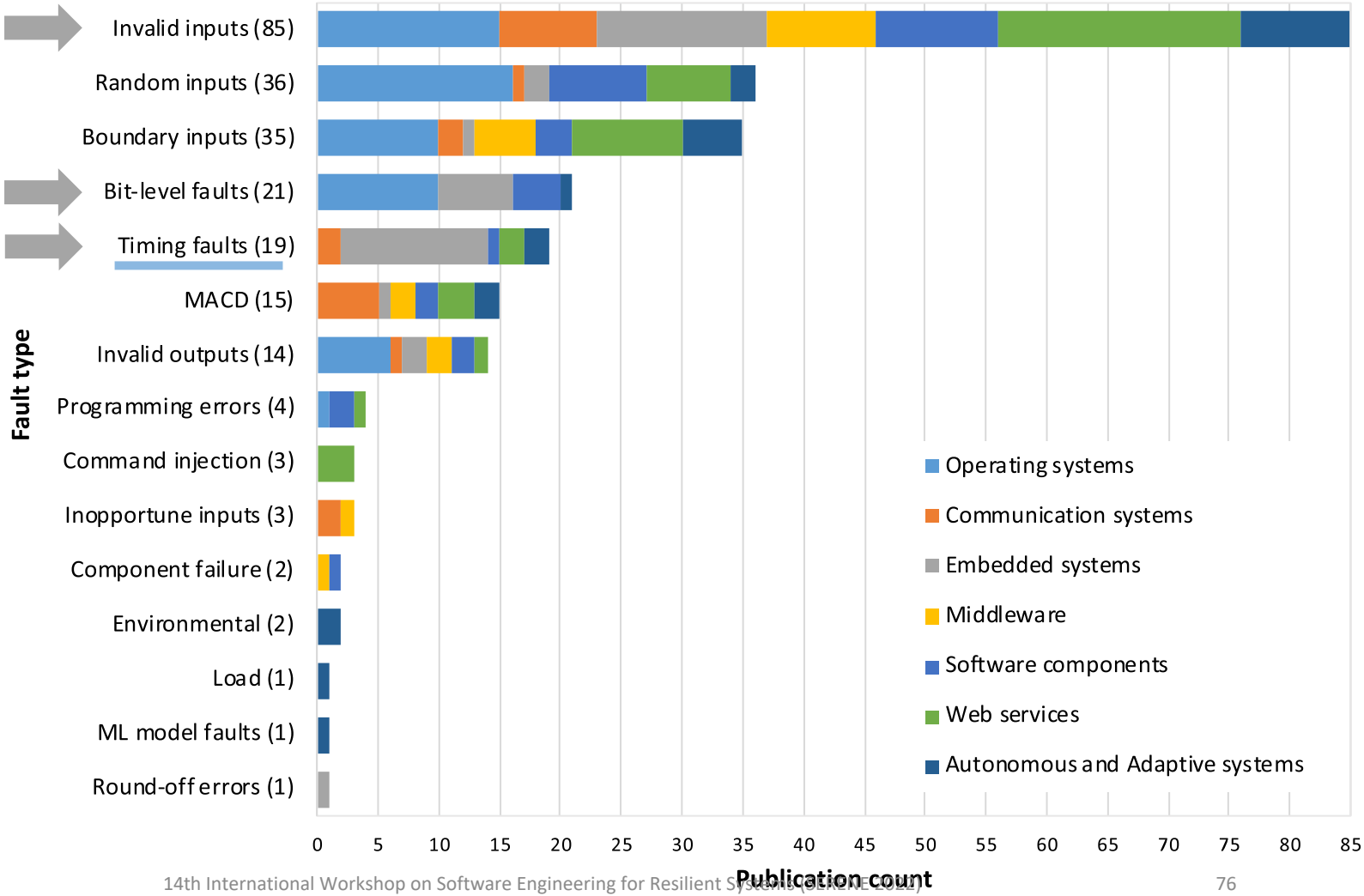
Types of faults
Operating systems



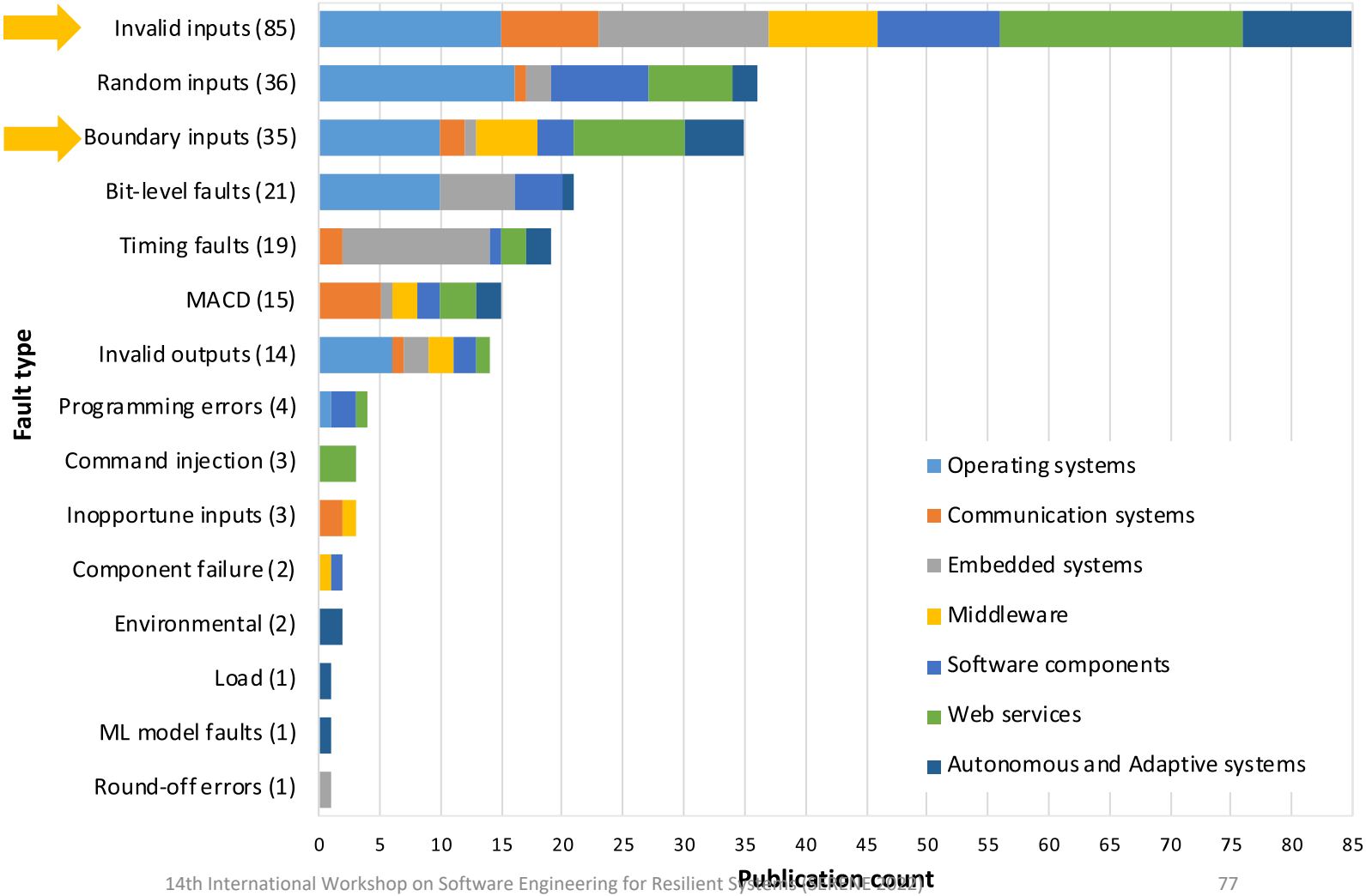
Types of faults
Communication systems



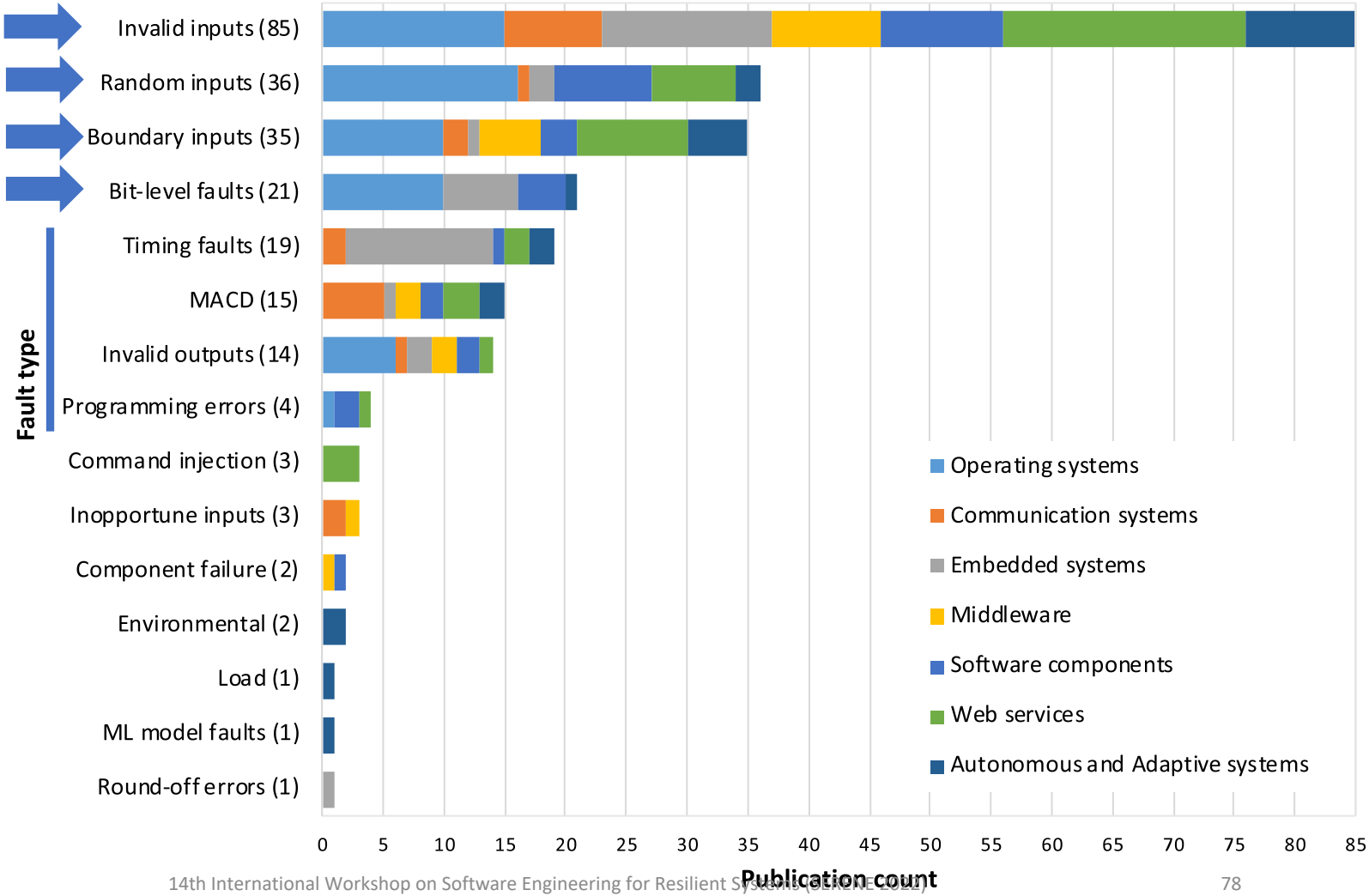
Types of faults **Embedded systems**



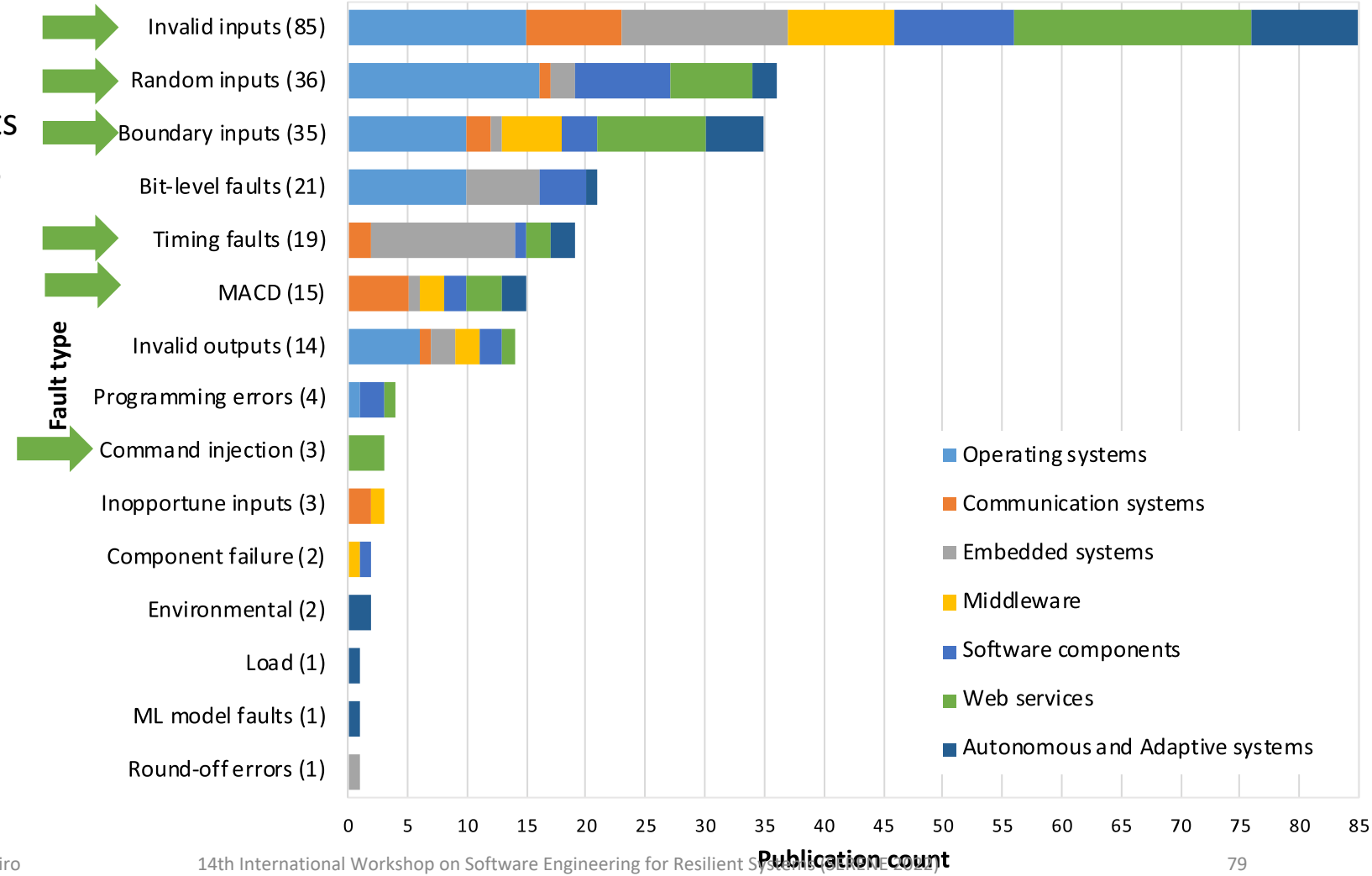
Types of faults
Middleware



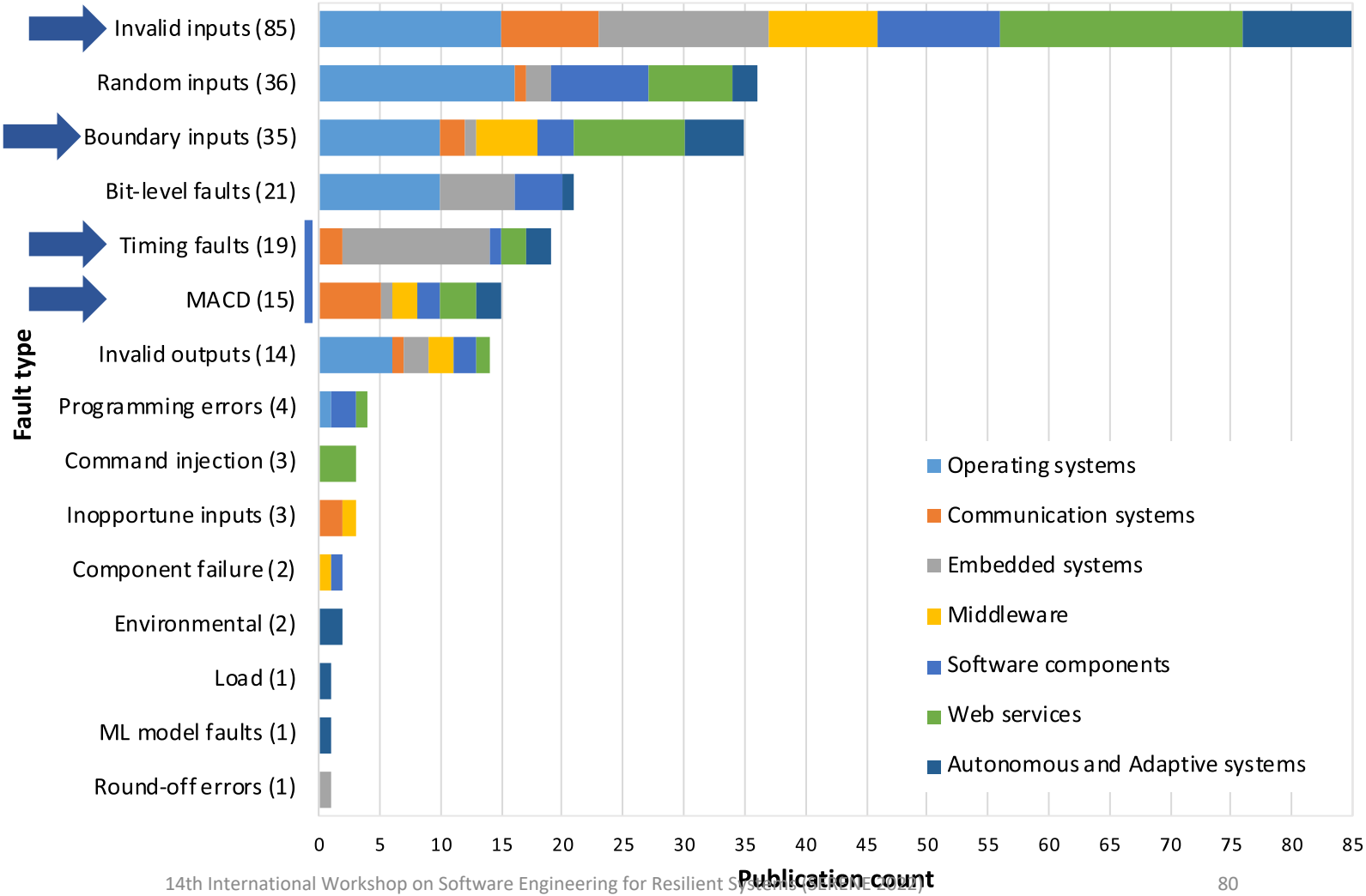
Types of faults
Software components



Types of faults
Web services



Types of faults **Autonomous and Adaptive systems**



About the fault types…

- Invalid inputs and boundary inputs span all system types
  - Usefulness and applicability
- Random inputs touch nearly all system types
- Half of the works using random inputs target operating systems
- Timing faults are prevalent in embedded systems
- MACD are frequent in communication systems
- Bit-level faults are mostly associated with operating systems and embedded systems
- Software components are the category in which the most diverse types of faults have been used

Which are the methods used to characterize robustness?

Classifying robustness

- 13 structures
- 33 different classification schemes
- From binary to 4 categories + 12 subcategories

Kernel failure, Workload failure, File system corruption, No impact
Crash, Fatal error, Application not responding, No failure
No problems detected (FM1), System or applications hang (FM2), System crashes and reboots (FM3), Same as FM3 but there are corrupted files (FM4)
Reboot, Crash, Application not responding, No effect
Detected failure, Silent failure, Hang failure, Crash failure
Operating System exception, Timeout, Correct result, Silent data corruption
No failure, Class1 (no specification violation), Class 2 (specification violation), Class 3 (crash or hang)
Correct, Timeout, Error, Erratic
Mission success rate, Traffic violations per km, Accidents per km, Time to traffic violation
Correct output, Wrong result, System hang, Exception
No failure, Application error, Application hang, System crash

About the classification models…

- Heterogeneous!
- General concern with the severity of the failure
- Complex structures
  - Finer grain
  - Classification difficulties → error prone
  - Tend to be more system-specific
- Binary classification is prevalent (more than half of the works)
- CRASH is prevalent among the non-binary (in 14% of the works)
- Huge heterogeneity among the remaining

Highlights

Highlights (1)

- First works focus on operating systems
- Fault injection and model-based testing are the main techniques used
- Fuzzing, code changes injection, mutation testing, or model-based analysis
- Message fields are the main target, although messages are also used
- Function invocations are popular (API, function, system calls, driver calls)

Highlights (2)

- Invalid inputs dominate the types of faults
- Random, boundary, bit-level, and timing faults also relevant
- Faults at the message-level and invalid values returning function calls
- Correct / Incorrect behavior
- Many use adaptations of CRASH

Research challenges

Challenge – Systems (1)

- There are types of systems for which robustness evaluation techniques are unknown or rising

- Blockchain systems
  - Complexity
  - Strong integrity concerns
  - Timing requirements
  - Recent work on fuzzing smart contracts

Challenge – Systems (2)

- REST services

- Cyber-physical systems
  - Strong interaction between physical and computation parts
  - Uncertainty of the environment and nature of the system

Challenge – Interplay

- Interplay between robustness and safety
- Autonomous driving cars or Unmanned Aerial Vehicles
  - Strong safety concerns
  - Highly dynamic and uncertain environments
- How to characterize robustness in perspective with the different safety requirements of such systems?

Challenge – Machine Learning

- Heterogeneous terminology
  - Resilience, reliability, adversarial robustness, trustworthiness
- Non-determinism
- Explainability should be considered
- Rising methods and tools
  - Many quite different from classic methods
  - Target is sometimes the machine learning model, or the system
  - Training phases are to be considered
  - Changes in the environment also

Challenge – Autonomous systems

- Machine learning parts along with other engineered components (sometimes distributed)
- Handle strict requirements regarding reliability or safety
- Lack of robustness may compromise other system properties (e.g., timeliness, security)

Challenge – Classification

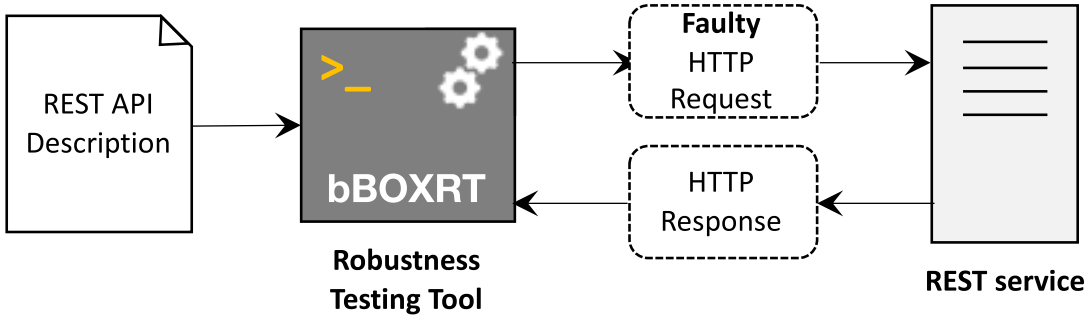
- Standardized methods for classifying robustness across (heterogeneous) systems
- One size fits all?
- Foster comparability of results

Filling one of the gaps… REST case study

REST

- Major companies now provide a REST interface to their services
- Interface description document is not required, although OpenAPI is increasingly being adopted
- Less rigid access opens space for unexpected inputs to reach the service
- Client mistakes may be acceptable, but not server mistakes
- Developers have additional tasks
  - Matching HTTP verbs
  - Selecting how inputs should be specified (body, path, query)
  - Responses include two main parts (header / body) that may not be consistent

A simple approach



- Invalid + boundary + random inputs
- CRASH for classifying failures + behavior tags

- Can we use this to evaluate the robustness of REST services?
- Can we use this to trigger failures in business critical services?
- Which kind of issues can we detect?

Results overview (1)

- 52 services public and in-house
- Examples: Google drive, Google Calendar, Spotify, Trello, Slack, Figshare, Docker Engine API
- Private company services

- Failures triggered in half of the 52 services
- 12% of the 1352 operations tested showed at least one problem
- Could happen in in-house or in services built with no robustness requirements, how about in business-critical services?

Results overview (2)

- 52 services public and in-house
- Examples: Google drive, Google Calendar, Spotify, Trello, Slack, Figshare, Docker Engine API
- Private company services

- Failures triggered in half of the 52 services
- 12% of the 1352 operations tested showed at least one problem
- Could happen in in-house or in services built with no robustness requirements, how about in business-critical services?

Private company services

- An empty value in an argument caused:
  - 503 service unavailable + datastore fatal error
- A few other similar failures
- All issues were confirmed by developers

What have we learned? (1)

- REST services are being made available on-line, carrying residual bugs that affect the overall robustness of the services
- Bugs disclosed at the service implementation and middleware levels
- Security issues were triggered
  - Malicious inputs
  - wrong input usage or missing validation
- Information disclosure was frequent
  - Code structure, SQL commands, database structures, or database vendor.
- Null, empty, and string-related faults were the most effective faults
  - Strings: Random characters and malicious were quite effective.

What have we learned? (2)

- Frequent problems observed included storage operations, null references, and conversion issues
- Contrary to previous work in SOAP, the null/empty value faults that triggered issues
  - Did not actually directly led to the disclosure of null references problems.
  - Triggered other kinds of problems (e.g., Data Access Operations)
  - Triggered issues that were masked by services and resulted in vague responses
- Only Abort and Hindering failures were triggered (remaining seem difficult to trigger in this context)

What have we learned? (3)

- Only Abort and Hindering failures were triggered (remaining seem difficult to trigger in this context)
- Mismatches between the interface description and the actual service implementation were detected
- Current OpenAPI specifications are being written without attention to basic operation details (e.g., missing data type details)
  - Several of these cases turned out to be associated with robustness problems
- OpenAPI specifications lack complete information regarding the expected behavior of the service (e.g., when in presence of invalid inputs),
  - Doubts when analyzing tests results
  - Issues for application integration

What have we learned? (4)

- In almost half of the services tested, we found non descriptive error messages (accompanied with a poor specifications)
  - Do not allow clients to gain much insights regarding the real issues
- Access to server logs was not sufficient to understand the root cause of failures in the Docker Engine.
- Useful even in services with high reliability requirements
- Missing validation is the main cause for problems in in-house services
  - Although some related with poor practices
  - Some obvious to avoid by senior programmers (e.g., using prepared statements)
  - Others would be difficult to detect (e.g., the use of a driver holding a bug).
- Robustness testing results were highly repeatable

Wrapping up…
The road ahead

- REST is the de facto interface of many systems and system parts
- Worthwhile exploring in the context of more complex systems
  - Other properties involved, safety, timeliness,…
- Can robustness assessment techniques help in more reliable and secure blockchain systems?
- Systems using machine learning models
  - Non-determinism
  - Models and engineered parts
  - New methodologies required

Questions?