SISMA
Solutions for Engineering Microservice Architectures

Guglielmo De Angelis
guglielmo.deangelis@iasi.cnr.it

SAKS IASI

who i am

- name: Guglielmo
- surname: De Angelis
- group:
  - SaKS: SOFTWARE AND KNOWLEDGE-BASED SYSTEMS
  - http://saks.iasi.cnr.it
- topics:
  - software engineering
  - service oriented architecture
  - software testing
  - (software) model-driven engineering

roadmap

- project overview
- context and challenges
- solutions dimensions
- a tech glimpse
  - test program similarities

fact sheet

SISMA is an Italian MIUR financed project

Call:	PRIN 2017
ERC Field:	PE-6 PE-6-3 (i.e., Software engineering) PE-6-2 (i.e., Computer systems, and parallel/distributed systems)
Line:	A
Start Date:	29 Aug. 2019
End Date:	28 Feb. 2023
Target:	architectural design of microservice native applications; migration from monoliths to microservice arch style; CI/CD and runtime management; specific support for automatic testing and failures prediction.
Resources:	http://sisma-prin2017.gitlab.io/ https://gitlab.com/sisma-prin2017

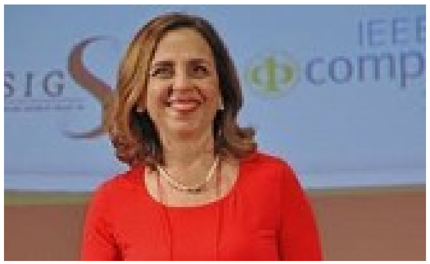
who is SISMA?

SAKS IASI

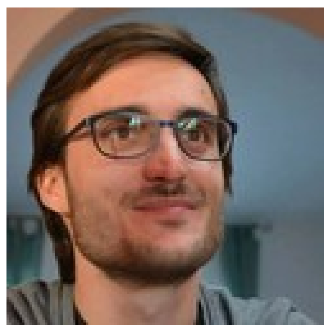
UNIVERSITA' DEGLI STUDI DI MILANO BICOCCA

POLITECNICO MILANO 1863

the team: CNR (IASI+ISTI)

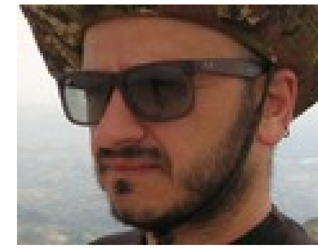


- Antonia Bertolino (ISTI)
- Renan Greca (ISTI)



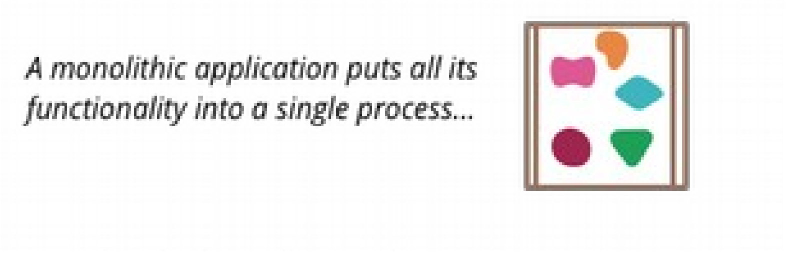
- Morena Barboni (AR IASI)
- Alessandro Pellegrini (PostDoc IASI)

- Emanuele De Angelis (IASI)
- Maurizio Proietti (IASI)



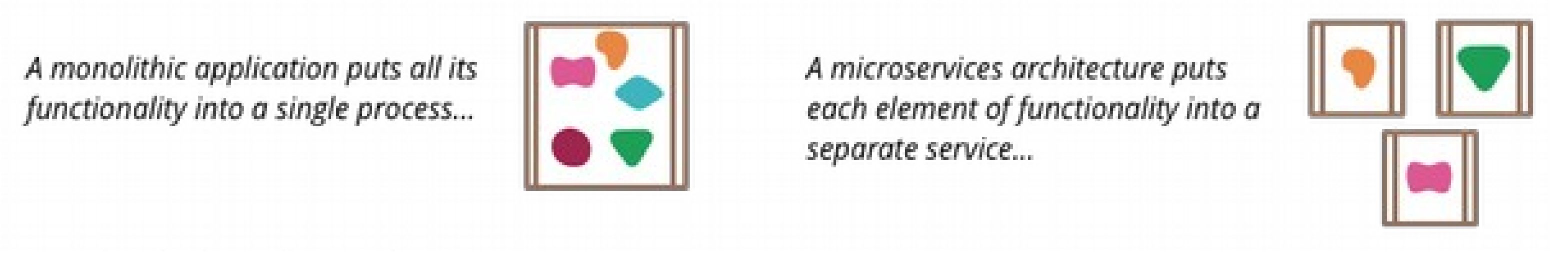
- Guglielmo De Angelis (IASI)

monoliths VS microservices



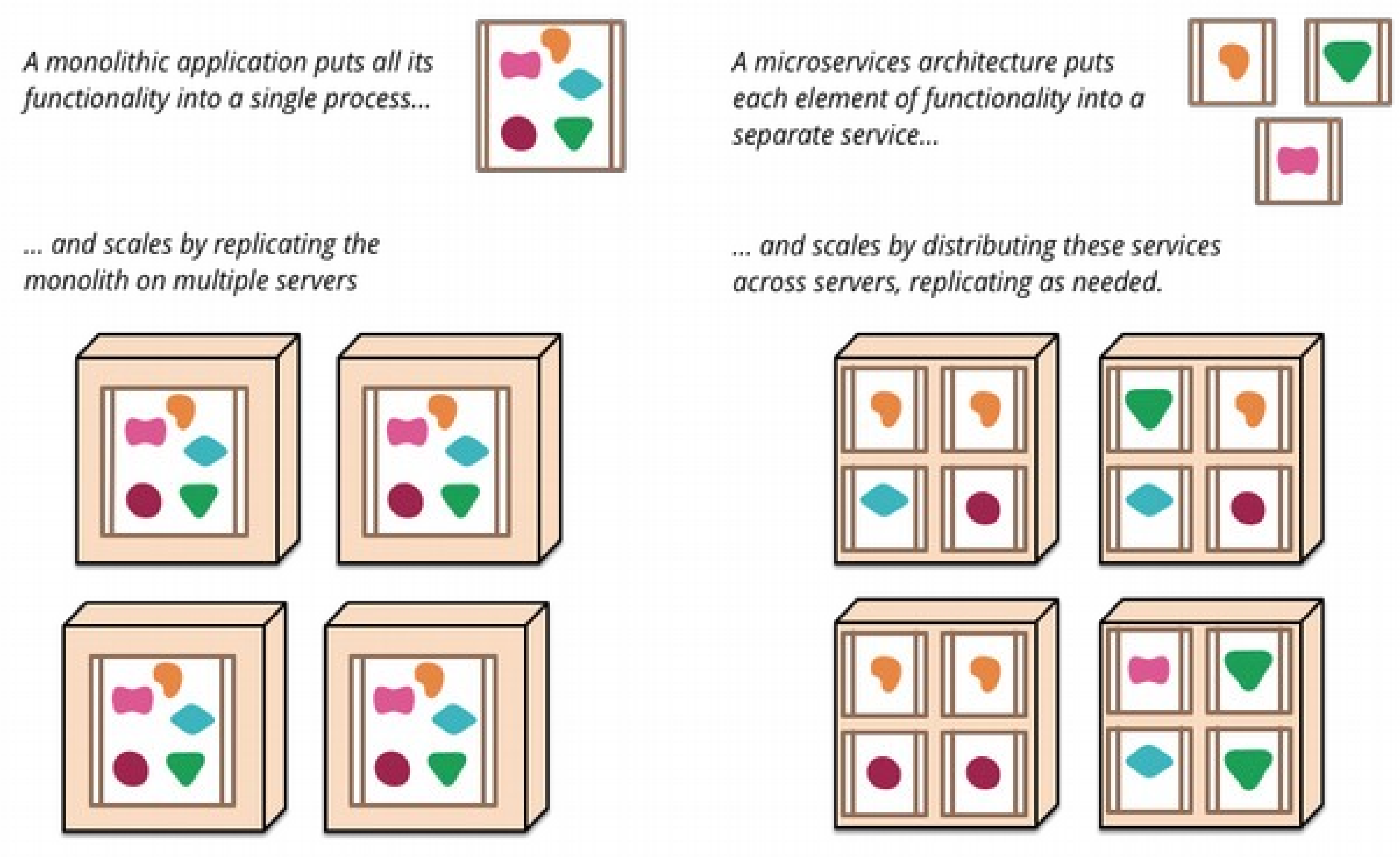
https://martinfowler.com/articles/microservices.html

monoliths VS microservices



https://martinfowler.com/articles/microservices.html

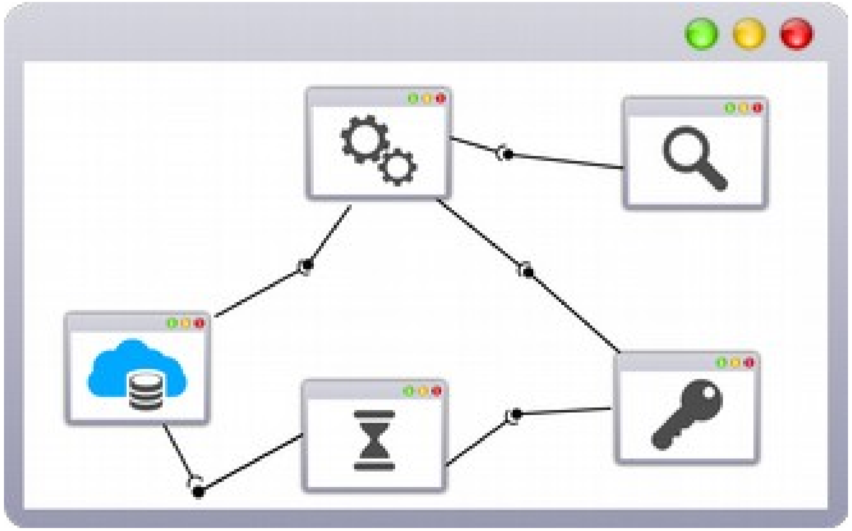
monoliths VS microservices



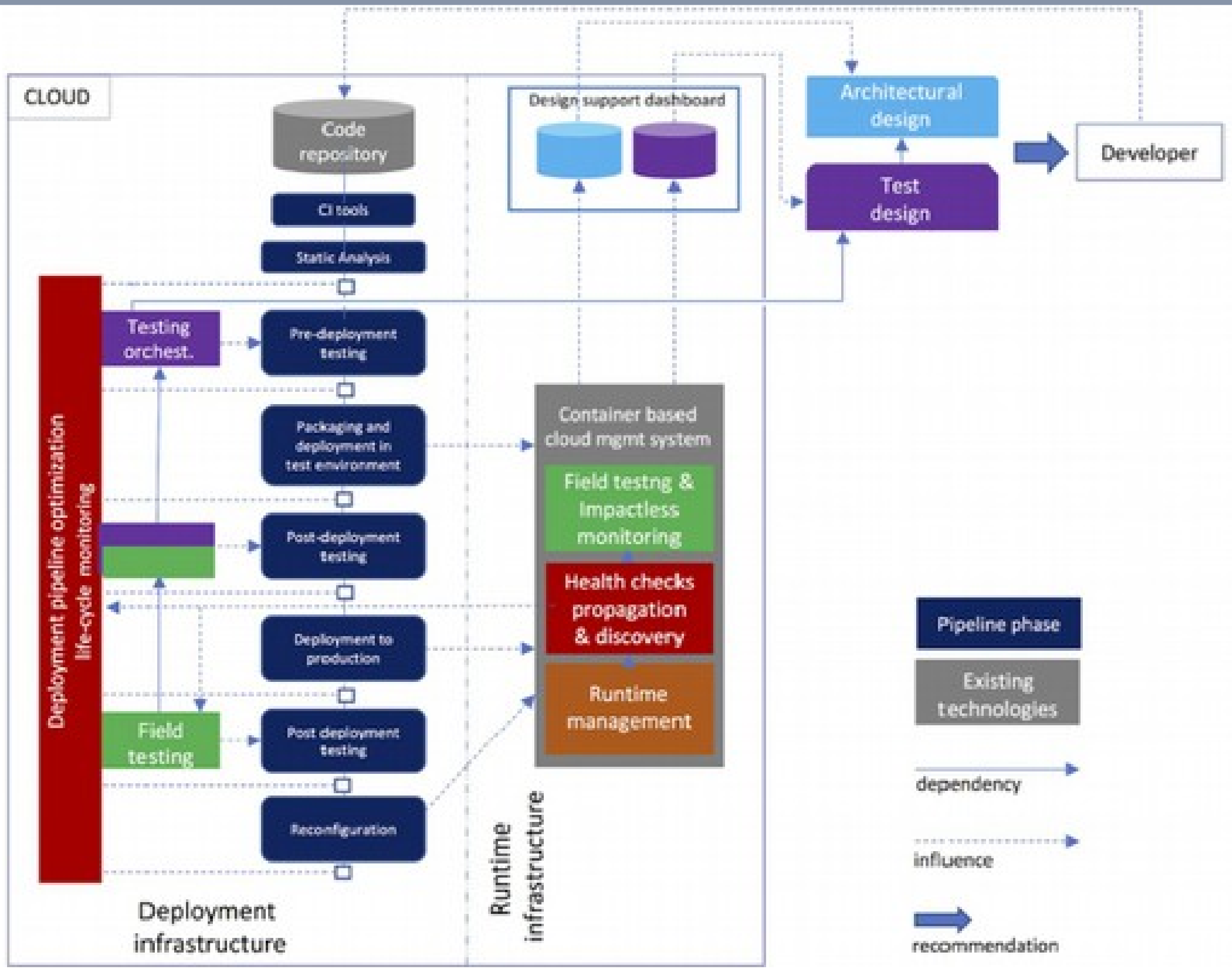
https://martinfowler.com/articles/microservices.html

microservice arch style ...

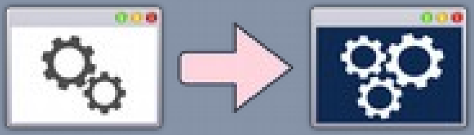
- high modularity
- usually different provider per microservice
- deployment logically distributed
- all the interactions take place over some kind of network abstraction



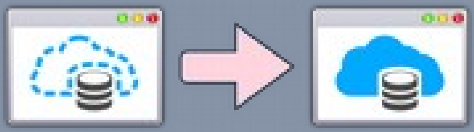
SISMA: solution dimensions



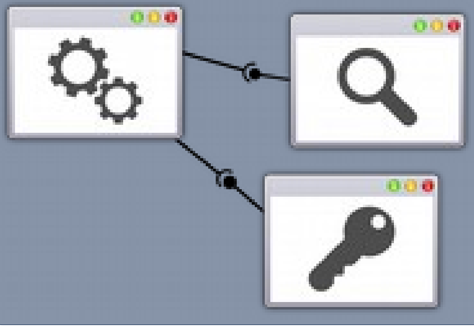
(some) types of evolution



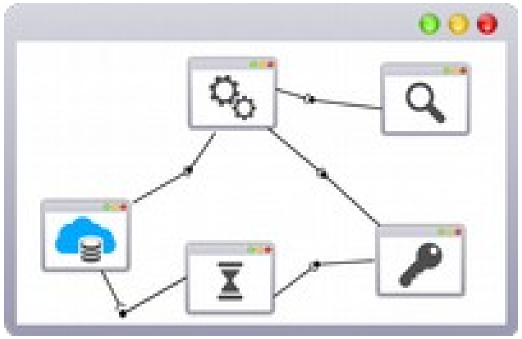
- evolution of a constituent



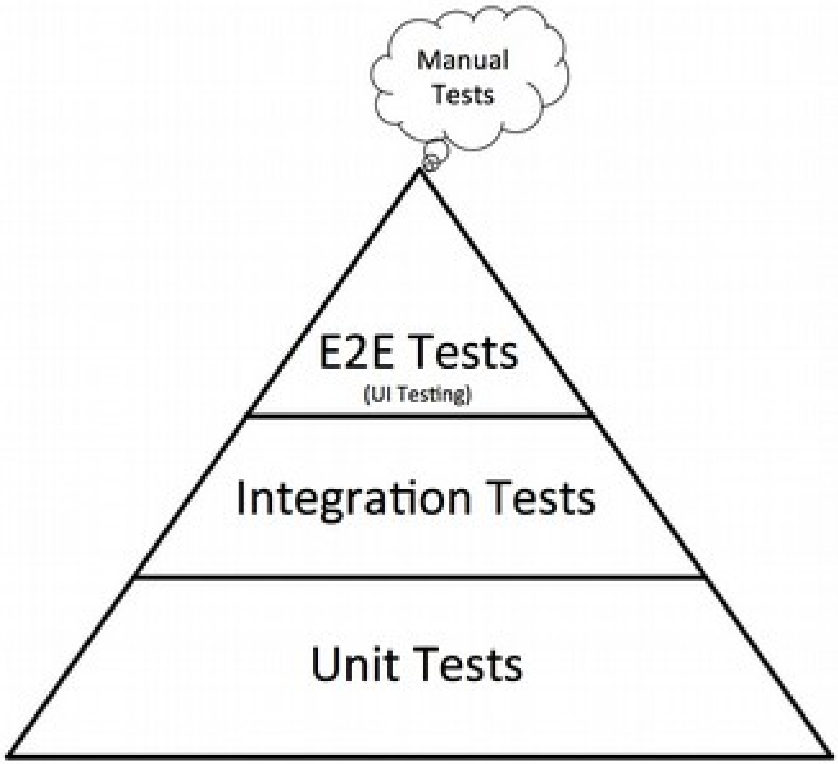
- usage of some additional functionality



- interdependencies among constituents

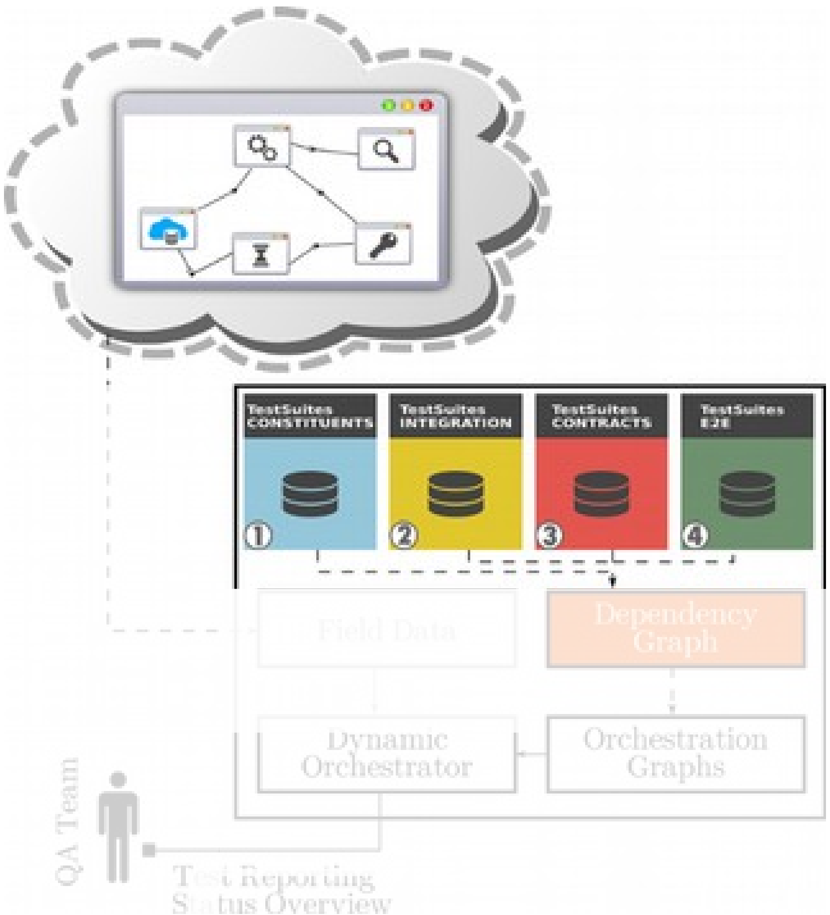


a long story ... short



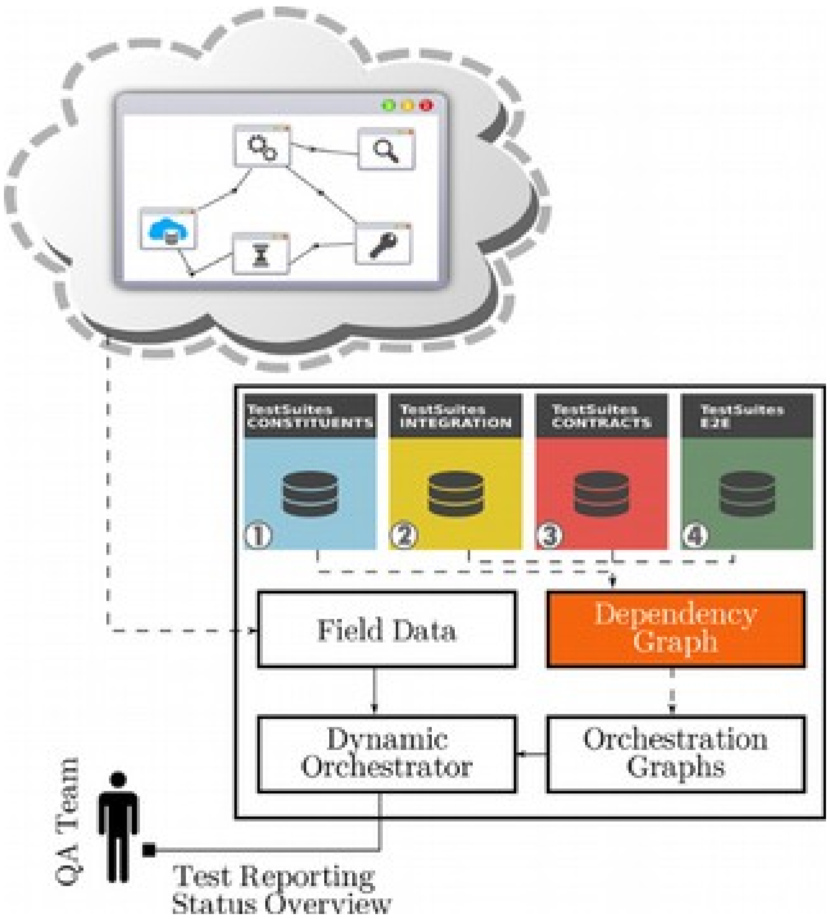
- different target means:
  - different level of abstraction
  - different level of automation
  - different number of test available
- however
  - tests in all these levels have to be properly designed and launched
  - always assure good quality to the tests
    - e.g., by assuming effective and realistic operative conditions

some ideas driving our research

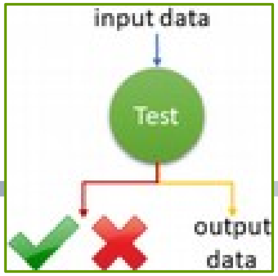


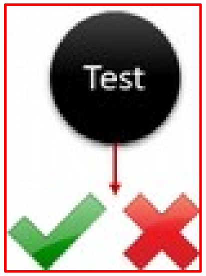
- overall intents:
  - increase the quality, and the reusability of testing artifacts
  - better support of automation for non-unit tests
- <u>test orchestration strategy:</u> creation of complex test suites as the composition of simple testing bundles
- <u>test orchestration policies:</u> QA Team governs test activities and plans test cases aggregation by reasoning on
  - relations among the microservices
  - relations among test cases
  - observed information/interactions

overall strategy

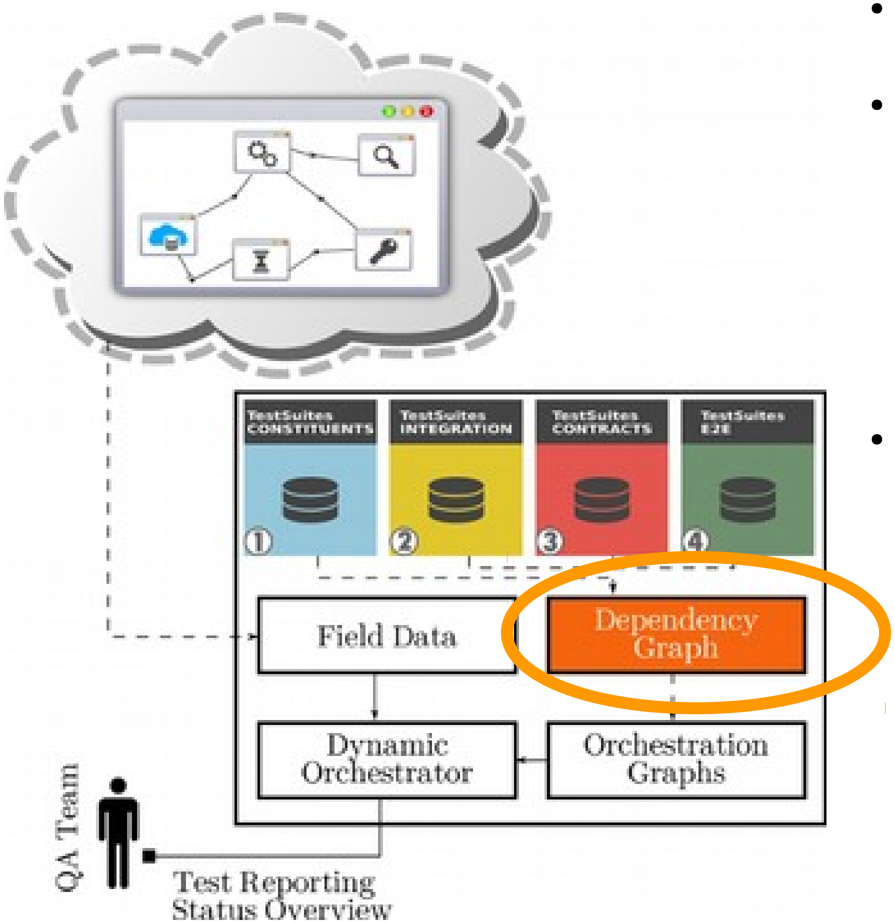


- TestSuites dependencies enable the declaration of orchestration graphs
- orchestration graphs aim to address specific testing objectives
  - reflecting potential evolution
  - focusing only on a sub-set of microservices
  - avoiding to launch test cases considered not relevant by each specific testing goals
- orchestration graphs
  - sequential, alternative, or parallel combination of test cases
  - next test is decided on-line taking into account the outcome from the previous one
  - extension of both selection and prioritization
  - decision is either *verdict-driven*, or *data-driven*

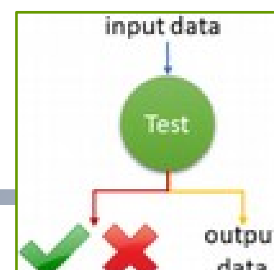


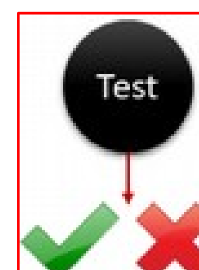


overall strategy



- TestSuites dependencies enable the declaration of orchestration graphs
- orchestration graphs aim to address specific testing objectives
  - reflecting potential evolution
  - focusing only on a sub-set of microservices
  - avoiding to launch test cases considered not relevant by each specific testing goals
- orchestration graphs
  - sequential, alternative, or parallel combination of test cases
  - next test is decided on-line taking into account the outcome from the previous one
  - extension of both selection and prioritization
  - decision is either *verdict-driven*, or *data-driven*





what makes test programs “similar” in microservice applications?

- test programs similar if they:
  - involve the same microservice instance, or they connect to the same remote API
  - locally activate overlapping API
  - raise similar kinds of errors

test program analysis

- guide the definition of orchestration graphs by supporting the identification of dependencies among test programs
- assume the availability of the test program source-code
  - explicit declaration of the test programs to consider
  - scan source-code repositories looking for test programs
- rely on the symbolic execution of the considered test program to
  - exercise (parametric) test programs *independently* of their arguments
  - carve test data by exploring admissible executions subsumed by a test program
- collect “relevant” information from the symbolic execution of the test programs
  - the analysis of the collected information is decoupled from the execution phase
  - symbolic execution only produces assertions about the explored configurations
  - post processing analysis of the collected assertions reveals existing dependencies

test program analysis: overall schema

- for each test program:
  - prepare the test program
  - run the test program in a Symbolic Executor Engine
  - track specific statements of interest in distinguished reports
- query the reports
  - extracting dependencies among the test programs
  - inferring similarities among the test programs
  - carving admissible test data

test program analysis: tech stack

- guides the definition of orchestration graphs by supporting the identification of dependencies among test programs
- assumes the availability of the test program source-code
  - explicit declaration of the test programs to consider
  - scan of some source-code repository looking for test programs
- re ogram to
  - e
  - c gram
- co on of the te
  - the analysis of the collected information is decoupled from the execution phase
  - symbolic execution only produces assertions about the explored configurations
  - post processing analysis of the collected assertions reveals existing similarities

- reference testing framework Junit
- test program identified by “@Test”

test program analysis: tech stack

Java Bytecode Symbolic Executor (JBSE)

- … ting the
- … e
  - scan of some source-code repository looking for test programs
- relies on the symbolic execution of the considered test program to
  - exercise (parametric) test programs *independently* of their arguments
  - carve test data by exploring admissible executions subsumed by a test program
- collects “relevant” information from the symbolic execution of the test programs
  - the analysis of the collected information is decoupled from the execution phase
  - symbolic execution only produces assertions about the explored configurations
  - post processing analysis of the collected assertions reveals existing similarities

test program analysis: tech stack

- g
  i
- a
  -
  -
- r … to
  -
  -

SWI Prolog

- collected assertions → Prolog facts
- similarities → Prolog rules
- analysis → Prolog queries

- collects “relevant” information from the symbolic execution of the test programs
  - the analysis of the collected information is decoupled from the execution phase
  - symbolic execution only produces assertions about the explored configurations
  - post processing analysis of the collected assertions reveals existing similarities

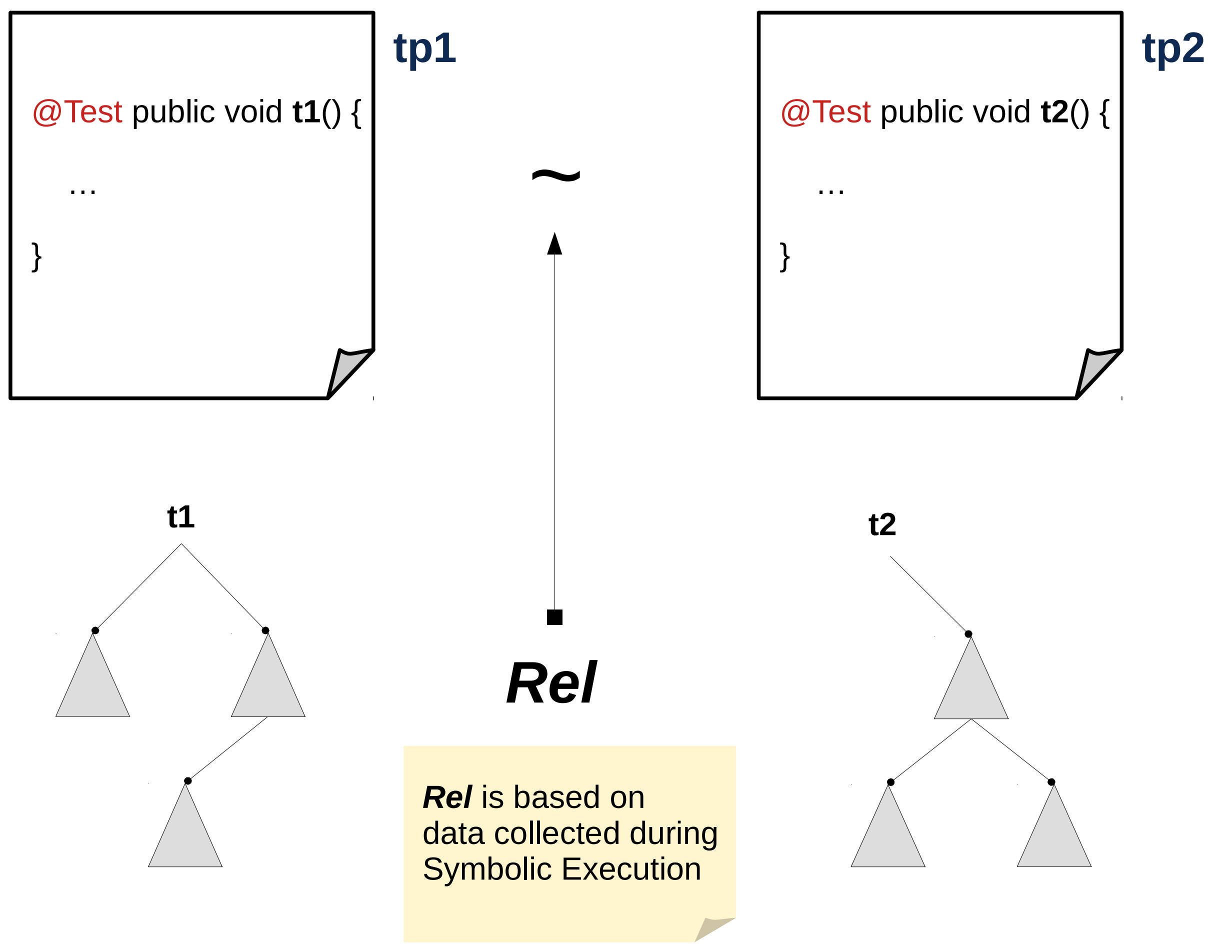
what makes test programs "similar" in microservice applications?

- test programs similar if they:
  - involve the same microservice instance, or they connect to the same remote API
    - `endpoint(test program, [branch point], seqnum, caller, callerPC, pathCondition, URI, parameters)`
  - locally activate overlapping API
    - `invokes(test program, [branch point], seqnum, caller, callerPC, frameEpoch, pathCondition, callee, parameters)`
  - raise similar kinds of errors
    - `exception(test program, [branch point], seqnum, caller, callerPC, pathCondition, class, message)`

inferring dependencies and similarity

Test programs

Symbolic Execution of Test programs



current achievements

- HYPERION
  - http://saks.iasi.cnr.it/tools/hyperion/
- started preliminary experimentation against FullTeaching

```
invokes('com.fullteaching.backend.e2e.rest.UserRestTest:testCreateUserOk', [1], 2,
'com/fullteaching/backend/e2e/rest/UserRestTest:testCreateUserOk:()V', 1, 2, [pointsTo('this', 'Object[10925]')], 'com/
fullteaching/backend/e2e/rest/UserRestTest:setUp:()V', []).
```

```
invokes('com.fullteaching.backend.e2e.rest.UserRestTest:testCreateUserOk', [1, 1, 1, 1, 2, 1], 285,
'java/net/URI$Parser:checkChars:(IIJJLjava/lang/String;)V', 40, 8854, [pointsTo('this', 'Object[10925]'),
constr('((1099511627776L)&([java/net/URI].java/net/URI:H_ALPHA))!=(0L)'),
constr('((4503599627370496L)&([java/net/URI].java/net/URI:H_SCHEME))!=(0L)'),
constr('((281474976710656L)&([java/net/URI].java/net/URI:H_SCHEME))!=(0L)'),
constr('((2251799813685248L)&([java/net/URI].java/net/URI:H_SCHEME))==(0L)'),
constr('(([java/net/URI].java/net/URI:L_SCHEME)&(1L))!=(0L)')], 'java/net/URI$Parser:fail:(Ljava/lang/String;I)V',
['Illegal character in scheme name', 0]).
```

future directions

- enable *dynamic orchestration* of test programs in available repositories
  - investigate sets of alternative strategies inferring dependencies and similarity
  - combining different regression techniques into an applicable approach
- improve the framework HYPERION
- validate the proposed ideas against some available case studies

thank you

SISMA: Solutions for Engineering Microservice Architecturesis

Call:	PRIN 2017
ERC Field:	PE-6 PE-6-3 (i.e. Software engineering) PE-6-2 (i.e., Computer systems, and parallel/distributed systems)
Line:	A
Start Date:	29 Aug. 2019
End Date:	28 Feb. 2023
Target:	architectural design of microservice native applications; migration from monoliths to microservice arch style; CI/CD and runtime management; specific support for automatic testing and failures prediction.
Resources:	http://sisma-prin2017.gitlab.io/ https://gitlab.com/sisma-prin2017