Modeling Complex Software Systems Behaviour

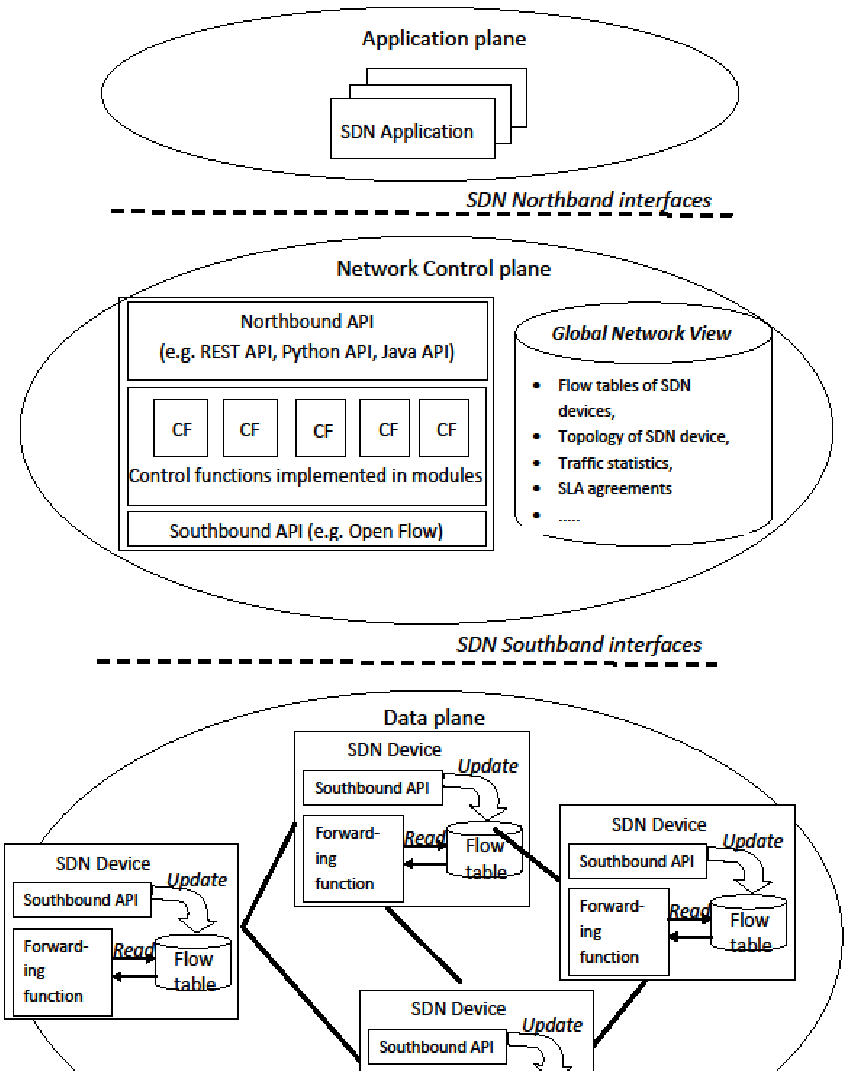
Tihana Galinac Grbac
University of Rijeka

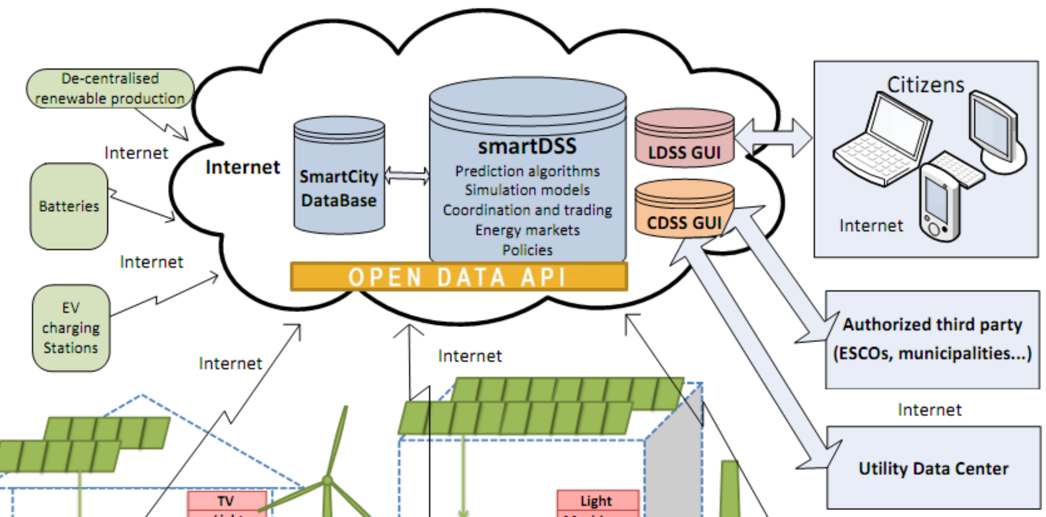
Outline

- Motivation
  - New development trends (IoT, service compositions)
  - Quality of Service
  - Software Technologies for Complex Software Systems
- Modeling software behaviour
  - Empirical study on complex software systems behaviour
  - Structure investigation
  - Modelling approaches
- SEIP Lab environment for research

Key problems with software evolution

- More and more software systems tend to evolve towards complex software systems (e.g. IoS) and systems of systems (SoS)
- Interconnection of peripheral systems over distributed network into system of systems (IoT)





Revolution or evolution of software systems

- Future: Communicating software systems distributed over the network, autonomously managed
  - Networks of networks, Systems of systems,
  - Interconnected by Internet network
- Software services realized as service chains ad-hoc established per each user or group of users

Problems with current software technologies

- Currently software and systems are staticaly configured and all software technologies are supporting such statically configured systems
- We need better abstractions that would enable and guide dynamic configuration of systems
  - Need for autonomous system control
- High level of expertise is needed to develop such systems
    - Concurrency, interoperability, scalability, reliability, security

Our motivation is to study

- How to model system behaviour?
  - How to model fault and failure behavior of complex software systems
  - How to model system growth?
  - How to better suport software developers developing ‘new software systems’
- Aiming to develop autonomous QoS/QoE solution for complex software systems

We are doing that in following projects:

- *2007-2013* ***New architectures and protocols in converged telecommunication networks****, Croatian Ministry of Science, Education and Sports*
- 2012 – 2016 ***Behavioral Types for large-scale reliable systems****, COST Action EU project*
- 2013 – 2017 ***Autonomous Control for a Reliable Internet of Services****, COST Action EU project*
- 2013 – 2016 ***Analysis and innovative approaches to development, management and application of complex software systems****,* University of Rijeka
- 2015 – 2018 ***EVOSOFT: Evolving software systems: analysis and innovative approaches for smart management****, Croatian Science Foundation.*

Importance of Software Structure

- Software engineering comunity has lang time ago identified importance of software structure on QoS attributes
- The whole software system design phase is devoted to careful selection and examination of software structure influence on software quality
- Well planed and designed software is precondition for achieving Quality of Service (Telecom example)
- How we can automate part of that process and enable runtime software reconfiguration?

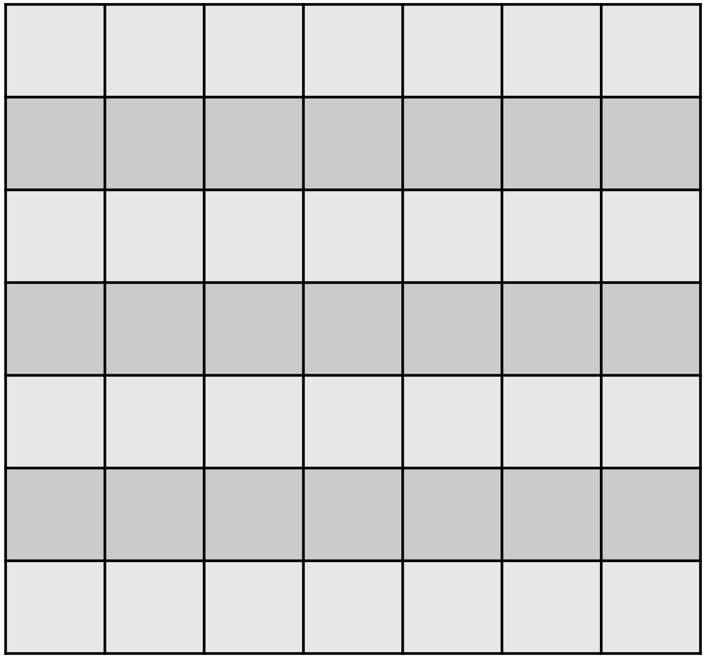
Our approach

- Understand structure and dynamics of networks, software networks and their influence on Quality of Service (QoS)/Quality of Experience (QoE) attributes.
- Could software structure be used as tool for modeling software behaviour?

Complex systems

- Number of levels of abstraction
- Global properties of system and local properties describing component behaviour
- Imposible to derive simple rules from local properties towards global properties*

System and system components



Aims

- Aim 1. To replicate studies aiming to confirm empirical principles proposed and used in software engineering community and to define solid base to ground new theories.
- Aim 2. To define structural dependencies between various empirical principles.
- Aim 3. To define formal models and innovative approaches that would enable accurate modeling of fault distributions and smart quality management of EVOSOFT systems.

Aims

- Aim 1. To replicate studies aiming to confirm empirical principles proposed and used in software engineering community and to define solid base to ground new theories.
- Aim 2. To define structural dependencies between various empirical principles.
- Aim 3. To define formal models and innovative approaches that would enable accurate modelling of fault distributions and smart quality management of EVOSOFT systems.

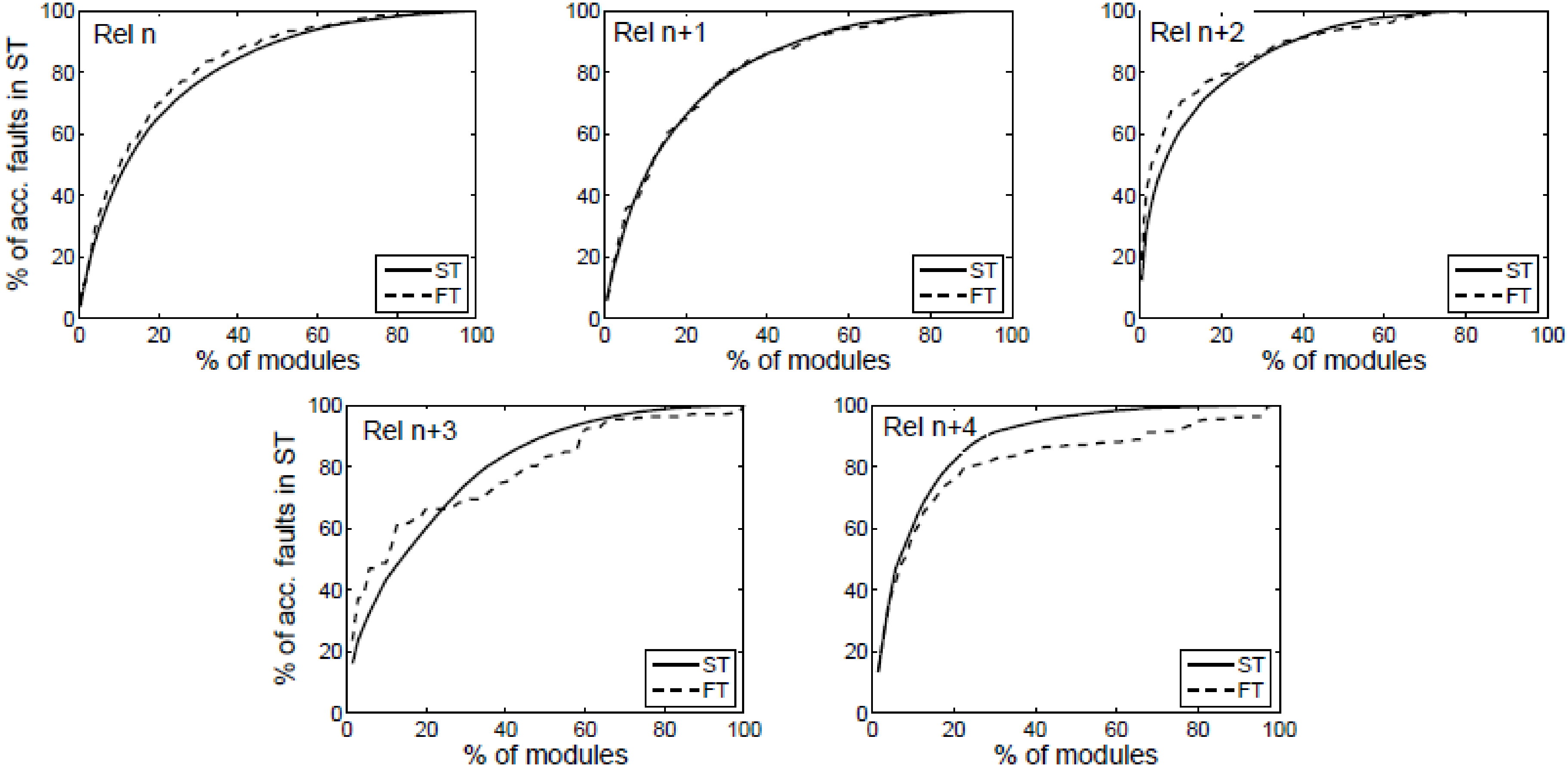
Empirical Fault distributions – Pareto principle

- 1984: V.R. Basili and B.T. Perricone, ”Software Errors and Complexity: an Empirical Investigation,” Commun. ACM.
- 2000: N.E. Fenton and N. Ohlsson. ”Quantitative Analysis of Faults and Failures in a Complex Software System,” IEEE Trans. Softw. Eng.
- 2007: C. Andersson and P. Runeson, ”A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems,” IEEE Trans. Softw. Eng.
- 2013: T. Galinac Grbac, P. Runeson, D. Huljenić, A second replicated quantitative analysis of fault distributions in complex software systems, IEEE Trans. Softw. Eng.

Hypotheses

1. Pareto principle of fault distributions
2. Persistance of faults
3. Effects of module size and complexity on fault proneness
4. Quality in terms of fault densities

Alberg diagrams: persistance of faults



Accumulated percentage of the number of faults in the system

Summary

- Pareto principle is clearly confirmed
- Modules identified to be fault-prone in one phase tend to be so in subsequent phases
- Size related predictors are not given any support for being good enough to identify fault-prone modules
- Fault density across releases and environments is of the same magnitude, but still varies a lot with factors not under control in the current studies

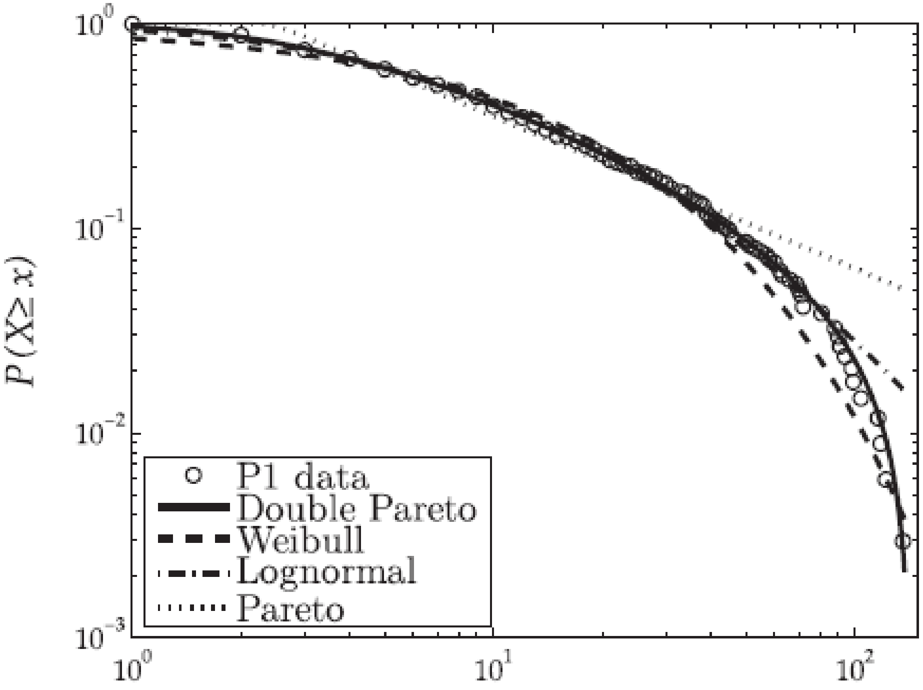
Conclusion

- All such principles ultimately depend on the underlying probability distribution of faults in a software system.
- However, the fulfillment of a certain principle does not determine the probability distribution uniquely.
  - there are several distributions that would result in the Pareto principle.
- empirical evidence in favor of some principle does not imply information on the probability distribution, and, indeed, our knowledge on the probability distribution of faults in software systems is still quite limited.

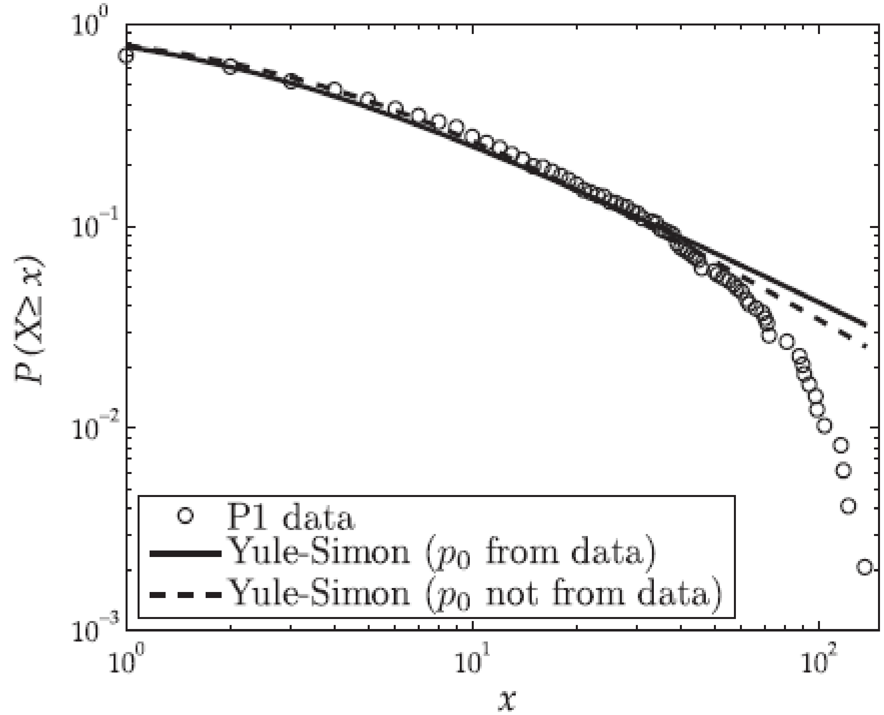
Empirical Fault distributions – Pareto distribution

- 2008: H. Zhang, On the distribution of software faults, IEEE Trans. Softw. Eng.
- 2011: G. Concas, M. Marchesi, A. Murgia, R. Tonelli, I. Turnu, On the distribution of bugs in the Eclipse system, IEEE Trans. Softw. Eng.
- 2015: T. Galinac Grbac, D. Huljenic:
  On the probability distribution of faults in complex software systems. Information & Software Technology

Results of distributions fit



Nonlinear regression fit for Pareto, double Pareto, Weibull and Lognormal distribution



Nonlinear regression fit for Yule Simone with and without a

Results of all studies

Ranking the probability distributions with respect to their performance in the non-linear regression fitting of the empirical samples for the random variable counting the number of faults in a software module

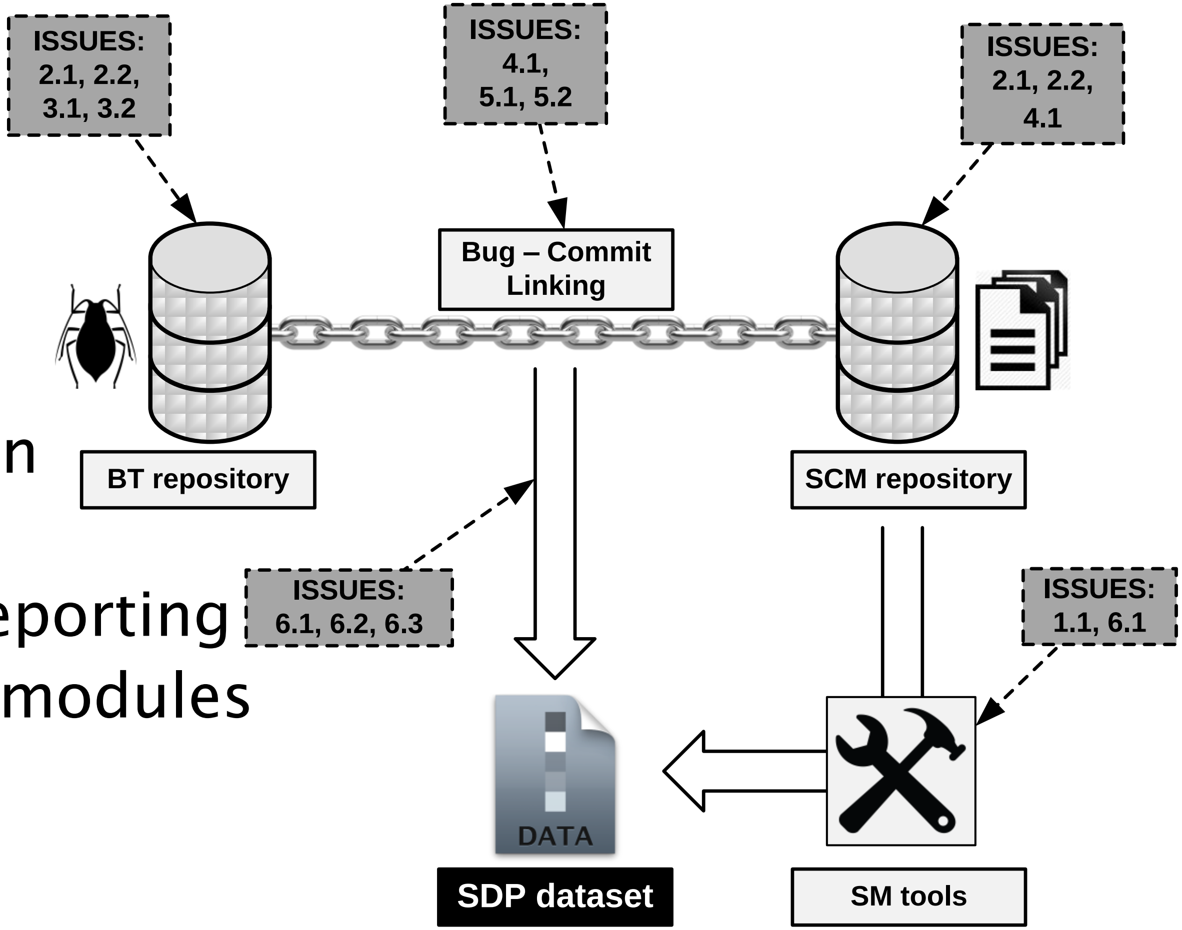
Rank	Galinac Grbac 2015	Concas et al 2011	Zhang 2007
1	Double Pareto	Yule–Simon	Weibull
2	Lognormal	Double Pareto	Pareto
3	Yule–Simon	Lognormal	–
4	Weibull	Weibull	–
5	Pareto	–	–

Compare probability distributions of faults in complex software system in different equilibration stages

- 2009 Hatton, L., Power-Law Distributions of Component Size in General Software Systems. IEEE Trans. Softw. Eng.
- One possible explanation for the difference is that the systems may be in a different stage of equilibration.
- The software system may be considered as a discrete complex system and studied as a physical system.
- It is in perfect equilibrium when there are no new faults reported.
- At that stage the discrete conservation laws may be imposed, just as in the continuous physical systems (e.g. conservation of energy).
- Our approach would be to compare probability distributions obtained by fitting to empirical fault distributions but at different time intervals.
- To determine how close to equilibrium a given software system is, we use the reliability growth models - The system is in equilibrium, when the reliability curve stabilizes.

Linking software repositories

- Linking issues:
  - No formal link
  - no standardized Procedure
  - Huge data collection Bias
  - Huge diversity of reporting and linking faults to modules



Bug – Code (BuCo) Analyzer Tool

–**Systematic literature review**

(**36** papers from [2] **+ 35** / 136 / 4447)

–**Exploratory study**

(12 studenats, observer triangulation, 5 projects, 4

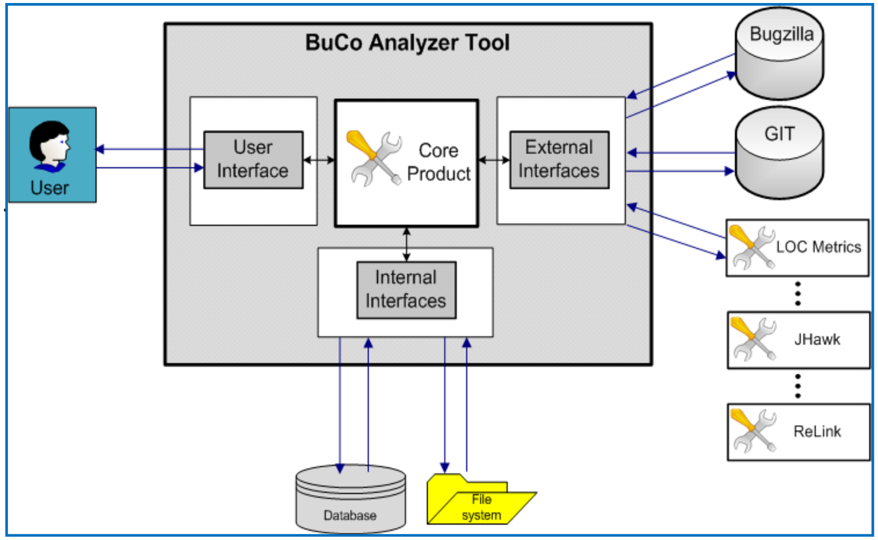
–**Software metrics tools analysis**

(iterativne assesment of 35 / 19 / 5 / **2** tools)

–**Iterative development process**

–**Systematic dana collection comparison**

(7 techniques, 5 projects, 37 releases)



Mauša G., Galinac Grbac T., Dalbelo Bašić B. : “Software defect prediction with bug-code analyzer – a data collection tool demo”, In: Proceedings of SoftCOM ’14, Split, Croatia, 2014

Aims

- Aim 1. To replicate studies aiming to confirm empirical principles proposed and used in software engineering community and to define solid base to ground new theories.
- **Aim 2. To define structural dependencies between various empirical principles.**
- Aim 3. To define formal models and innovative approaches that would enable accurate modelling of fault distributions and smart quality management of EVOSOFT systems.

Aim 2 To define structural dependencies between various empirical principles

- We have addressed four questions which lead us to six hypotheses which are finally grouped in following categories:
  1. subgraph presence
  2. structural evolution
  3. effects of structural evolution on defects
  4. motif stability in software structures

Petric J., Galinac Grbac T., *Software structure evolution and relation to system defectiveness,* Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering *EASE2014*

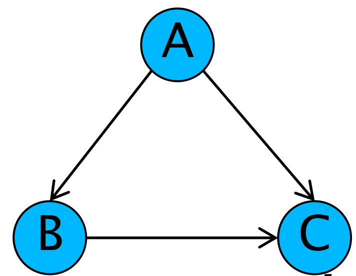
Approach to problem

- Present software as graph -> we developed

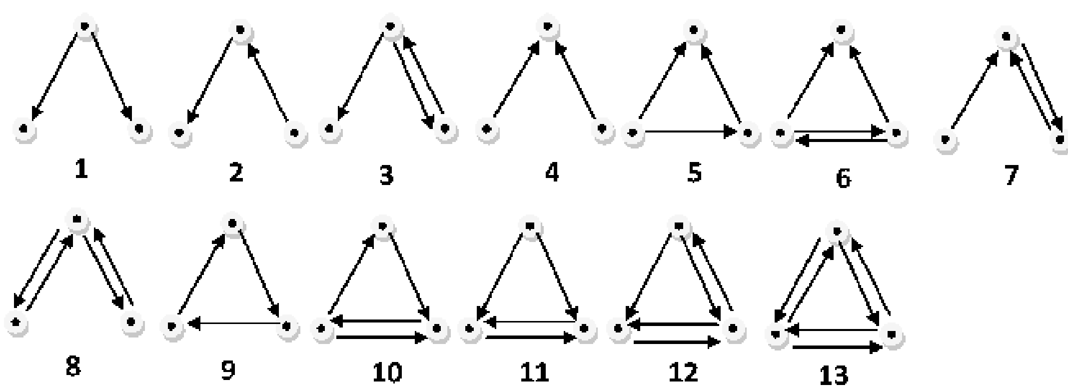
```
Class A {
        ...
        B.methodB();
        C.methodC();
        ...
}
```

```
Class B {
        ...
        C.method
        ...
}
```

rFind tool



- Determining subgraph frequencies and motifs with graphic tools -> we used mFinder/Kavosh



Conclusion

- We showed few things:
  - we observe that same set of subgraphs are present in all versions of system evolution
  - we proved that analyzed systems evolve continuously and the change in their structure is statistically significant
  - defectiveness is correlated with some subgraphs
  - motifs are shown to be consistent across system versions

Future work

- We will go deeper in finding how defect on class have influence on system structure
- We work on including different application domains
- In future we will also include time-period of software releases
- We will expand our rFind tool to work on different languages

Aims

- Aim 1. To replicate studies aiming to confirm empirical principles proposed and used in software engineering community and to define solid base to ground new theories.
- Aim 2. To define structural dependencies between various empirical principles.
- Aim 3. To define formal models and innovative approaches that would enable accurate modeling of fault distributions and smart quality management of EVOSOFT systems.

Structure and performance

- Tankovic, N; Galinac Grbac, T; Truong, H-L.; Dustdar, S: Transforming vertical Web applications into Elastic Cloud Applications, Proc. of IC2E 2015, 2015, Phoenix, USA.
  - H1: Structure of software distribution across the logical nodes influences software system elasticity
  - *Explanation: Distributed systems may be easier to expand and scale then vertical systems from performance and resource utilization cost perspective*
  - H2: The way how application is distriuted may provide some benefits for easier dynamic resource scaling

Tankovic, N; Galinac Grbac, T; Truong, H-L.; Dustdar, S: Transforming vertical Web applications into Elastic Cloud Applications, Proc. of IC2E 2015, 2015, Phoenix, USA.

ElaClo

- Web page: http://elaclo.com/

Acknowledgements: The work presented in this paper is supported by COST action 1304 Autonomous Control of Reliable Internet of Services (ACROSS) and the research grant 13.09.2.2.16 from University of Rijeka, Croatia

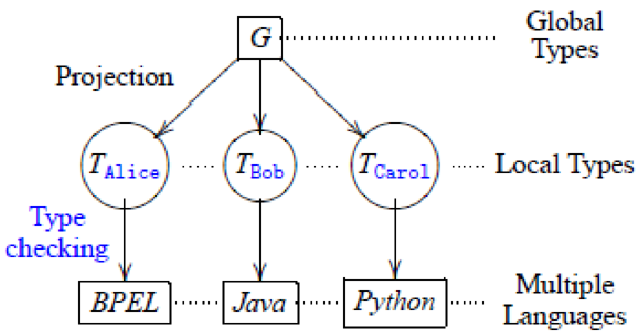
Future Work

- Develop framework for autonomous application transformation
  - measurement model (End user and Cloud provider - scale)
  - Autonomous decision model for application slicing and deployment
  - Run-time self-adaptation scaling and reconfiguration mechanism
  - Provide more exact benefits on family of experiments
- New programming abstactions and API's for autonomous decision model
- Tool development
  - Scaling the tool to bigger applications
  - Introduce more autimatization
  - Flexibility in cloud provider selection and different PAYG models adaptation

Session types for telecom. services

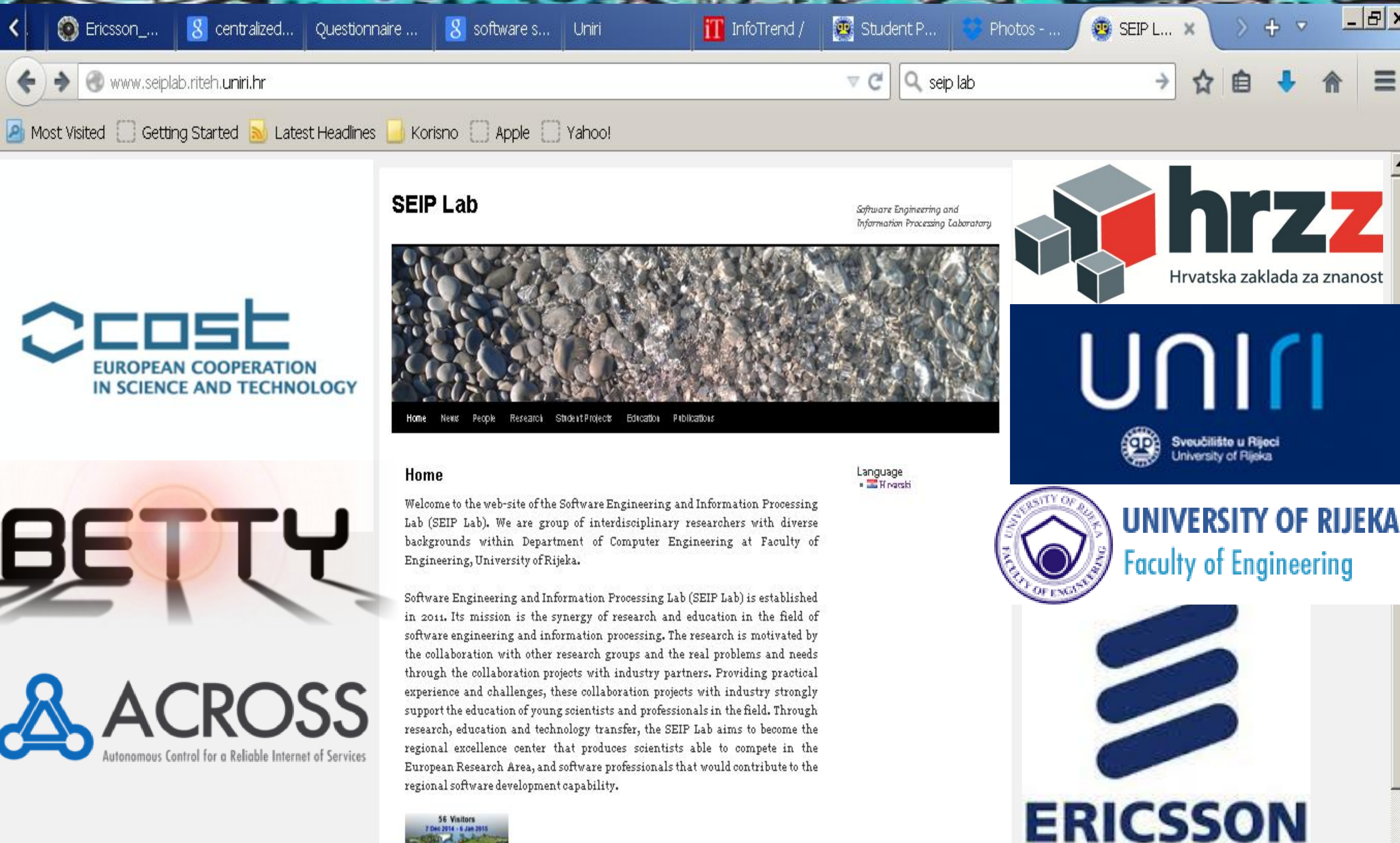
- Scribble programming language allows certification of global protocol interaction and projection onto local protocol implementation.
- tools for editing, verifying and projecting, numerous libraries that allow its integration with some general purpose languages such are java or python.

Acknowledgements: The work on this problem is supported by COST action 1201 *Behavioural Types for large-scale reliable systems (BETTY)*



SEIPLAB@UNIRI-RITEH

- We have established environment
  - Cloud environment
  - SDN network
  - Reconfiguration tools
  - Our data collection and analysis tools
- We would like to colaborate on this issues!



Ericsson_...
centralized...
Questionnaire ...
software s...
Uniri
InfoTrend /
Student P...
Photos - ...
SEIP L...
www.seiplab.riteh.uniri.hr
seip lab
Most Visited
Getting Started
Latest Headlines
Korisno
Apple
Yahoo!
SEIP Lab
Software Engineering and Information Processing Laboratory
Home
News
People
Research
Student Projects
Education
Publications
Home
Language
Hrvatski
Welcome to the web-site of the Software Engineering and Information Processing Lab (SEIP Lab). We are group of interdisciplinary researchers with diverse backgrounds within Department of Computer Engineering at Faculty of Engineering, University of Rijeka.
Software Engineering and Information Processing Lab (SEIP Lab) is established in 2011. Its mission is the synergy of research and education in the field of software engineering and information processing. The research is motivated by the collaboration with other research groups and the real problems and needs through the collaboration projects with industry partners. Providing practical experience and challenges, these collaboration projects with industry strongly support the education of young scientists and professionals in the field. Through research, education and technology transfer, the SEIP Lab aims to become the regional excellence center that produces scientists able to compete in the European Research Area, and software professionals that would contribute to the regional software development capability.
56 Visitors
cost
EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY
BETTY
ACROSS
Autonomous Control for a Reliable Internet of Services
hrzz
Hrvatska zaklada za znanost
uniri
Sveučilište u Rijeci
University of Rijeka
UNIVERSITY OF RIJEKA
Faculty of Engineering
ERICSSON