

Assessing Performance Evolution for Configurable Systems

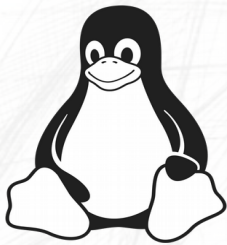


Advisors: Prof. Dr.-Ing. Ina Schaefer, Prof. Dr.-Ing. Norbert Siegmund

September 20, 2017
Stefan Mühlbauer

Configurable Systems

- Software systems provide configuration options (features)
- (De-)selecting options tune, dis- or enable functionality



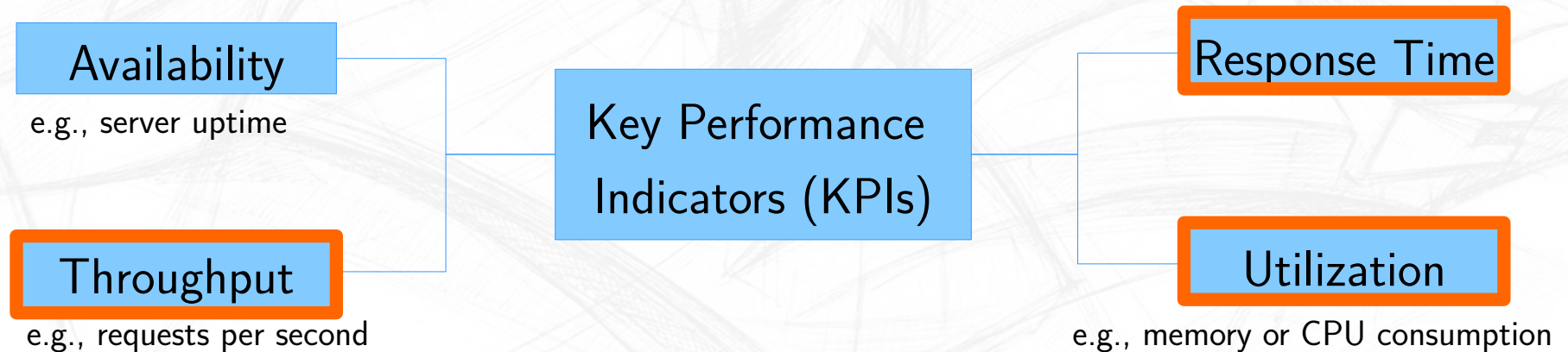
```
~$ zip -e -9 file.txt
```

- Unanticipated behavior can emerge with selections of multiple features (feature interaction)
 - Example: Compression and Encryption

*Compressing encrypted data can be **faster** than compressing raw data, since encrypted data is already more „compact“.*

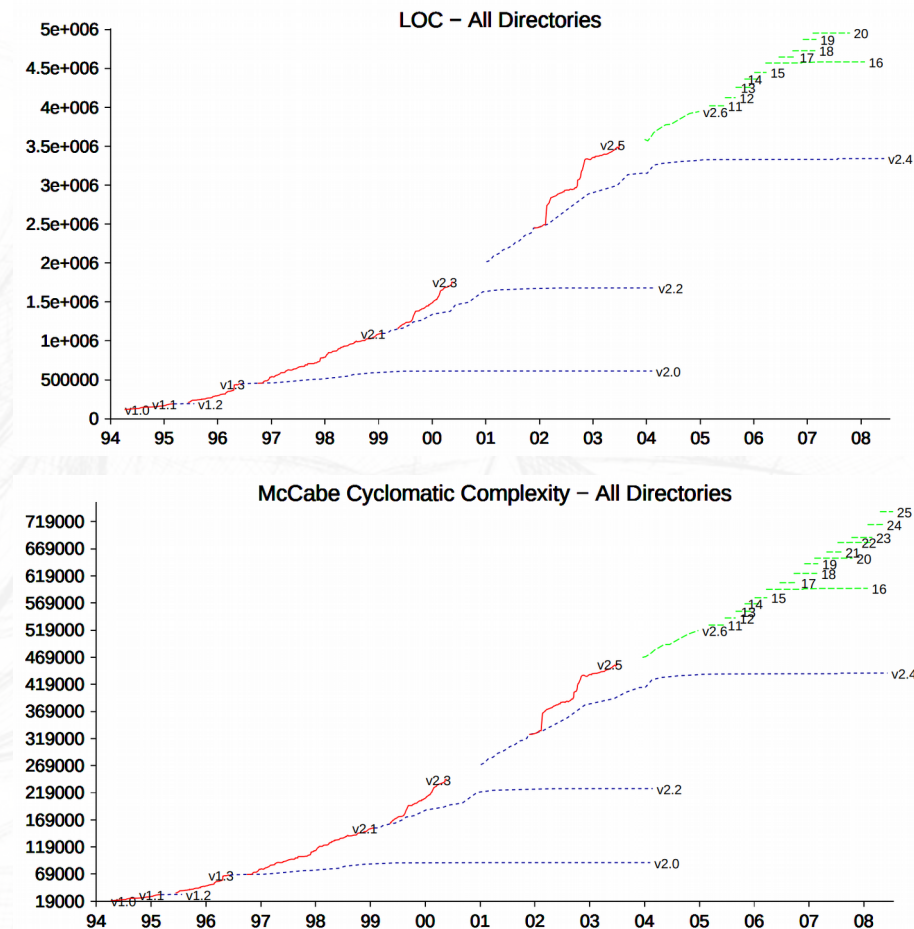
Performance

- Performance: How successfully is a task being performed?
- Software performance is described by four categories:



Software and Performance Evolution

- Evolution: Adaption to changing contexts/requirements
- Example: The Linux Kernel
- Performance regression
 - A symptom of software evolution
 - Degradation of performance quality of software over time



Source: Fig. 2 and 4 of [5]

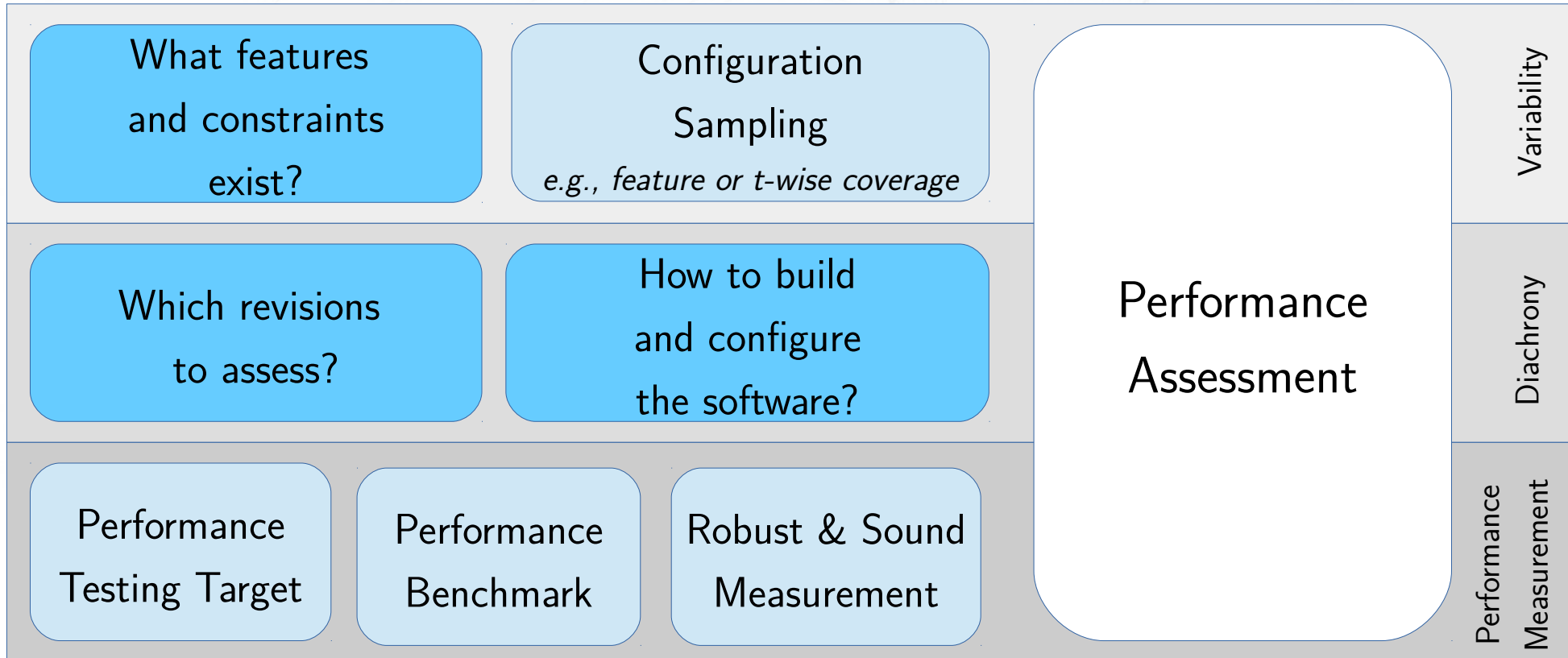
Performance Assessment

- Performance evolution is the change of performance over time
- Performance assessment for multiple revisions **required** to capture performance evolution

Motivation

- Problem: Performance Assessment – How?
- Goal: Description of performance assessment process with regard to variability, evolution, and statistical accuracy.
- Objectives
 - How to derive variability models and select configurations to assess?
 - How to select revisions to assess from revision history?
 - How to select robust and sound statistical measures?

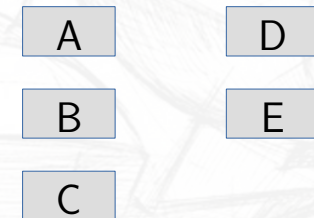
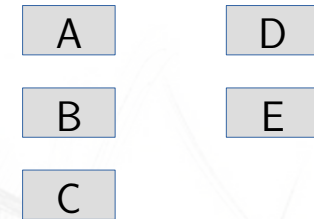
Performance Assessment Process



Variability: Feature Model Synthesis



- Feature Extraction [2]
 - Exploit used configuration APIs
 - Recover feature names, types and domains
- Constraint Extraction [3]
 - Infer constraints from rule violations:
 - “Every valid configuration compiles”
 - “Every valid configuration yields a different product”
- *Optional*: Semi-automatically organize features and constraints to feature diagram [4]



$B \rightarrow A \vee \neg D$

$D \rightarrow C \vee \neg E$

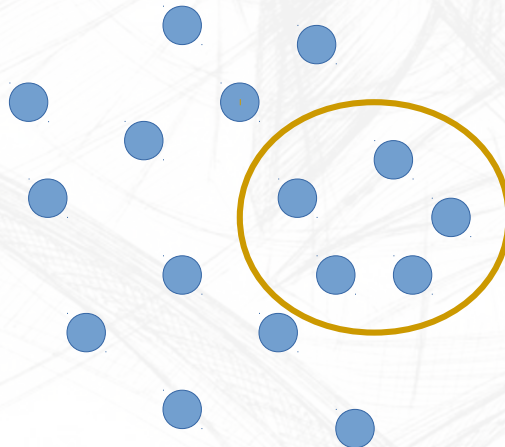
$E \rightarrow C \vee \neg D$

$C \rightarrow A$

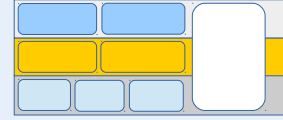
Variability: Configuration Sampling



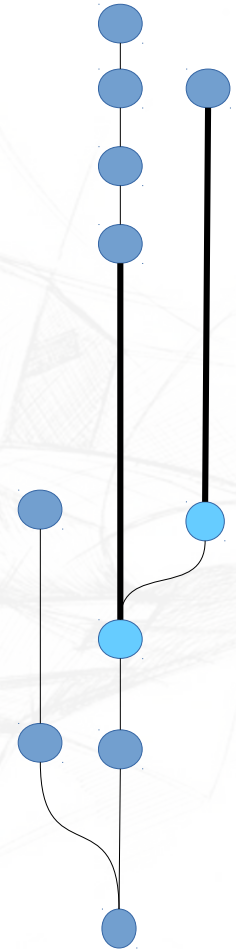
- Sampling: Finding a 'representative' subset of configurations
- Sampling strategies: Isolated solutions with coverage criteria
 - Pair- or t-wise: Coverage of simple and higher feature interactions
 - Feature coverage: Every feature selected at least once
 - ...



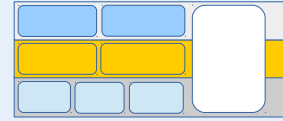
Diachrony: Revision Sampling (1)



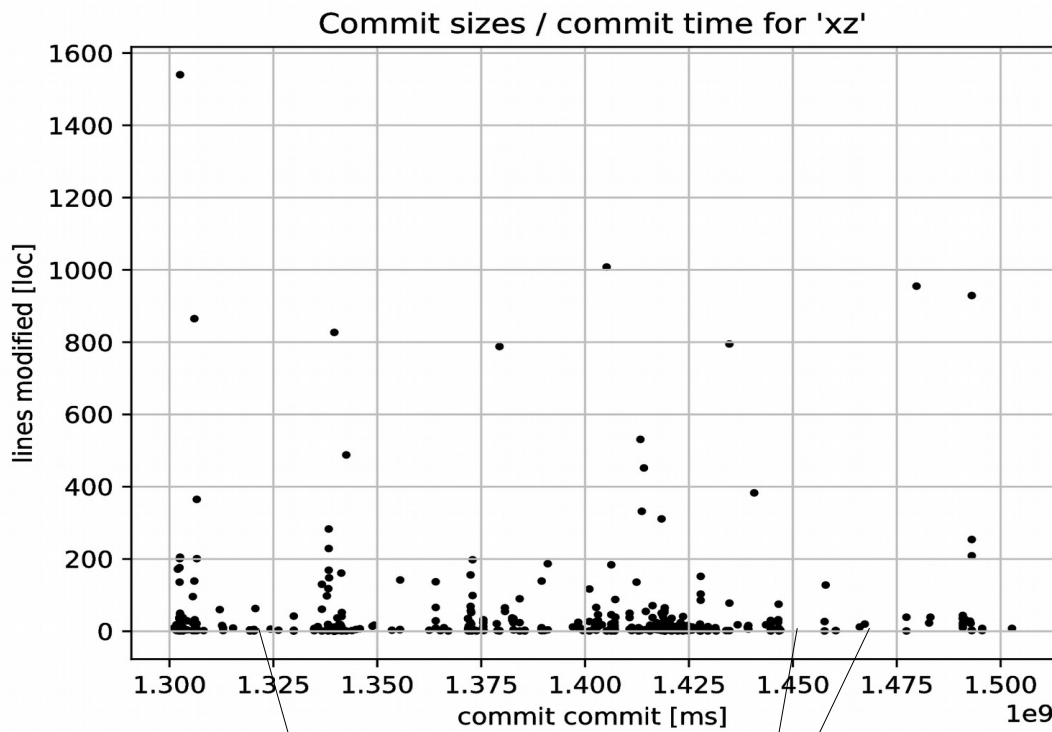
- Which revisions do we want to assess performance for?
 - Releases, release candidates,
 - bugs, and corresponding bug-fixes
- Information we can use to classify include
 - Revision history metadata (commit messages, ...)
 - Release notes
 - ...
- Which revisions best describe performance evolution?



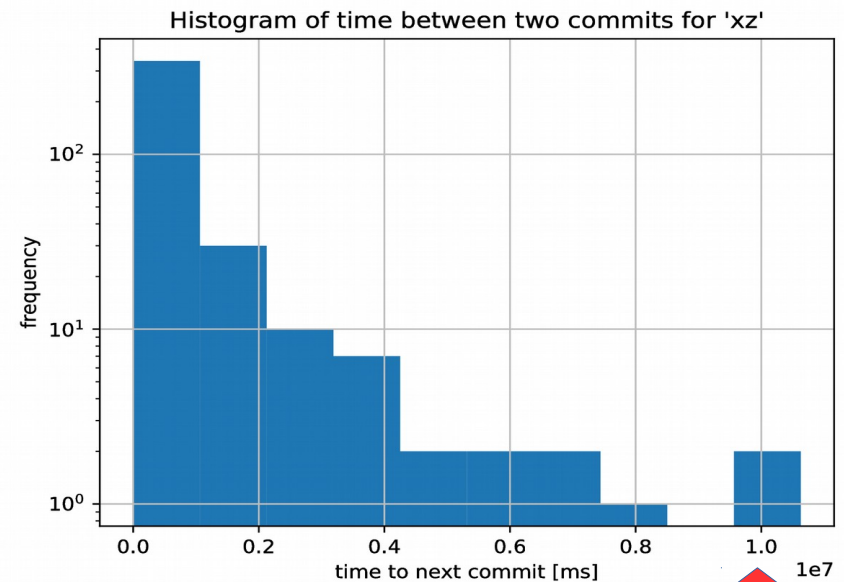
Diachrony: Revision Sampling (2)



- Idea: Revisions which have been the latest ones for a long time



„commit streaks“ followed by sparse sections



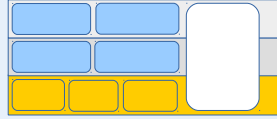
Very few commits with high temporal distance to next commit

Semi-automated Integration



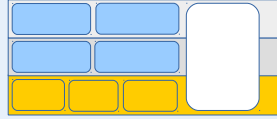
- How to automatically build a software system?
 - Usually: Manually predefined build routine, e.g., shell script
 - Possible extensions: Pattern matching for 'characteristic' files?
- How are configurations read by the software system?
 - Usually: Manually predefined templates, e.g., .properties file
 - Possible extensions: Pattern matching for 'characteristic' files?

Measurement: Soundness



- How to describe performance statistically accurately?
- Soundness: Are conclusions drawn from measurements correct?
 - Example: Use harmonic mean for rates, not the arithmetic mean
 - Two machines processing 100 requests with 100 and 10 hits/s each
 - Arithmetic mean: 55 hits/s, harmonic mean: 18.182 hits/s
 - Time required for processing requests: $1\text{ s} + 10\text{ s} = 11\text{ s}$
 - Two 'average' machines should take 11 seconds
 - Arithmetic mean: 3.64 s, harmonic mean: 11 s

Measurement: Robustness



- Robustness: Is the measure affected by outliers?
 - Example: Series 1, 1, 2, 3, 8, arithmetic mean of 5, median of 2
- Robust measure of central tendency: Median (2nd quartile)
 - Bigger than the lower 50% of measurements
 - Smaller than the upper 50% of measurements
- Robust measure of dispersion: Inter quartile range (IQR)
 - Difference between the 3rd and 1st quartile

Outlook for Evaluation



- Questions to address:
 - Do “commit streaks” and sparse sections exist? Is this a good idea?
 - How can we detect release commits?
 - Can we assess performance evolution with our strategy?
- Evaluation corpus
 - GNU XZ Utils (free compression tool)
 - X264 (free audio and video encoder)
 - SQLite (embedded relational DBMS)

Literature

- [1] Molyneaux, I. (2014). *The Art of Application Performance Testing: Help for Programmers and Quality Assurance* (2nd ed.). O'Reilly Media, Inc.
- [2] Rabkin, A., & Katz, R. (2011). *Static extraction of program configuration options*. In Proceedings of the 33rd ICSE (pp. 131–140). ACM.
- [3] Nadi, S., Berger, T., Kästner, C., & Czarnecki, K. (2015). *Where do configuration constraints stem from? an extraction approach and an empirical study*. IEEE Transactions on Software Engineering, 41(8), 820–841.
- [4] She, S., Lotufo, R., Berger, T., Wasowski, A., & Czarnecki, K. (2011). *Reverse engineering feature models*. In Software Engineering (ICSE), 2011 33rd ICSE (pp. 461–470). IEEE.
- [5] Israeli, A., & Feitelson, D. G. (2010). *The Linux kernel as a case study in software evolution*. Journal of Systems and Software, 83(3), 485–501.