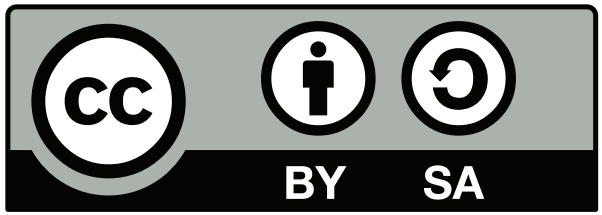
Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

20.3.2018



Lecture is every week Wednesday 09:00 - 11:00.

06.03.2019: topic, teams
13.03.2019: TISS registration, initial PR
20.03.2019: other registrations, guest lecture
27.03.2019: PR for first issue done, second started, HS: kleiner Schiffbau
03.04.2019: first issue done, PR for second
10.04.2019: mid-term submission of exercises
08.05.2019: (HS?)
15.05.2019:
22.05.2019:
29.05.2019:
05.06.2019: final submission of exercises
12.06.2019:
19.06.2019: last corrections of exercises
26.06.2019: exam

Popular Topics

14	tools
9	testability
9	code-generation
7	context-awareness
6	specification
6	misconfiguration
6	complexity reduction
5	validation
5	points in time
5	error messages
5	auto-detection
4	user interface
4	introspection
4	design
4	cascading
4	architecture of access
3	configuration sources
3	config-less systems
2	secure conf
2	architectural decisions
1	push vs. pull
1	infrastructure as code
1	full vs. partial
1	convention over conf
1	CI/CD
0	documentation

Tasks for today

(until 20.03.2019 23:59)

Task

Registration for talk, homework, teamwork and make sure to say which programming languages you know in STUDENTS.ini.

Task

Write text in at least one issue.

Tasks for next week

(until 27.03.2019 23:59, hint: to get help submit at least one day earlier)

Task

Description of homework as pull request in private repo. (Inside a folder for you, use GitHub name.)

Task

Description of teamwork (which application, which CM tool) as pull request in private repo. (Inside a folder for your team.)

Task

Fix at least one issue and write some text in at least one other issue.

Environment Variables

Semantics of Command-line Arguments (cont.)

- passed by main for a new process via (`int argc, char ** argv`)
- visible from other processes (e.g., via `ps aux`)
- could be passed along to subprocesses but hardly done
- need to be parsed by process
- portability: differences in parsing
- cannot be changed from outside (requires restart, no IPC)

Semantics

- are also per-process (`/proc/self/environ`)
- are not visible from other processes
- are automatically inherited by subprocesses
- need to be parsed by process (`[extern] char **environ`) but API is provided (`getenv`)
- cannot be changed from outside (requires restart or an additional IPC mechanism)

Task

What is wrong with the code in the book?

getenv

- is widely standardized, including SVr4, POSIX.1-2001, 4.3BSD, C89, C99 [1],
- is supported by many programming languages, and
- enforces `key=value` convention.

Usage

1. bypassing other configuration accesses (*Q:* 45 %)
2. locating configuration files
3. debugging and testing (*Q:* 55 %, *S:* 1,152, i. e. 43 %)
4. sharing configuration settings across applications (*Q:* 53 %, *S:* 716, i. e. 47 %)
5. for configuration settings unlikely to be changed by a user (*Q:* 20 %)
6. *"even when it is used inside a loop"* (*Q:* 2 %)

Portability

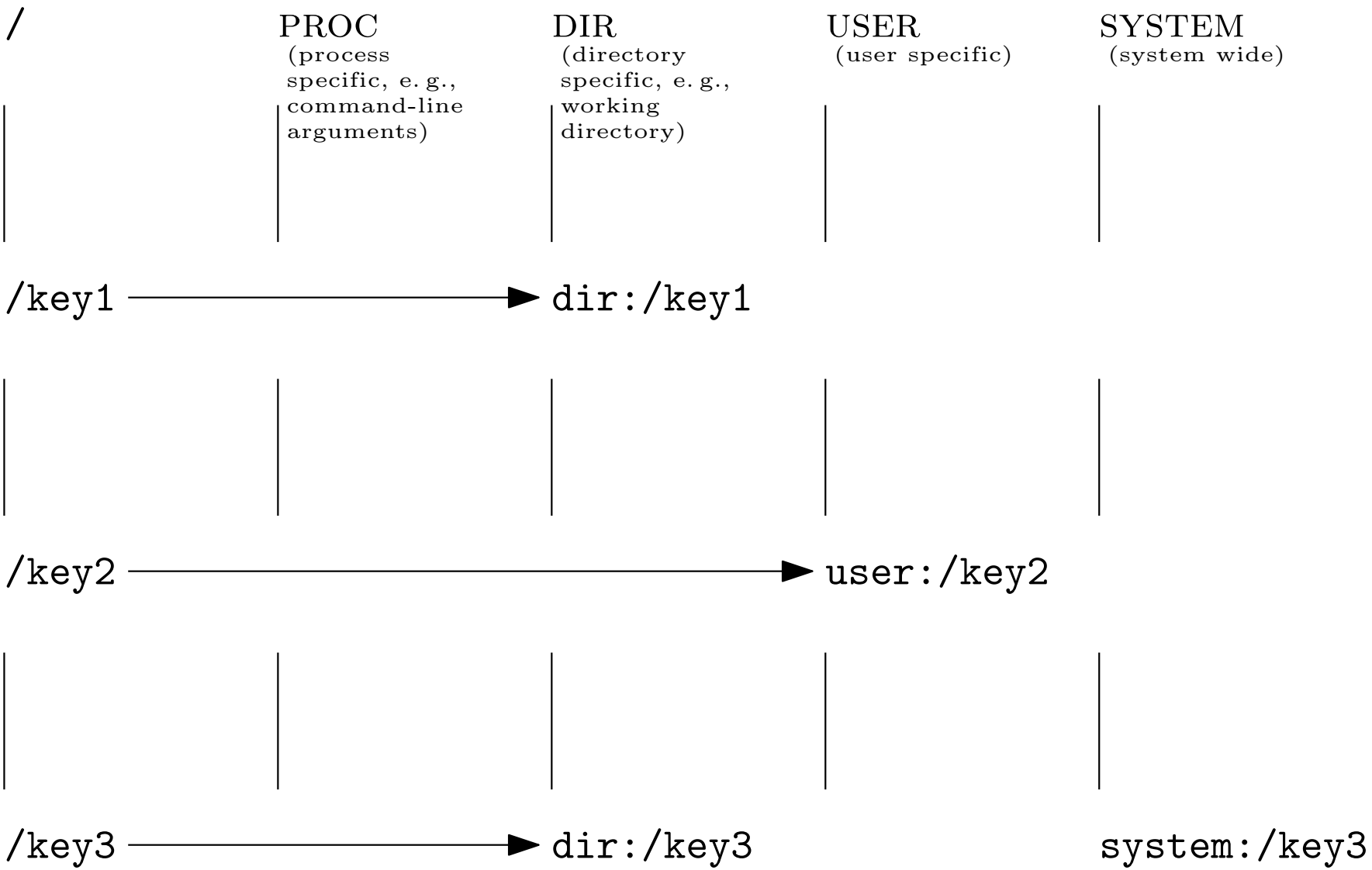
- no separators for values defined
- case sensitivity problems
- often many environment variables for the same purpose: TMP, TEMP, or TMPDIR
- sometimes one environment variable for different purposes: PATH

How can we deal with the many sources?

Requirement

A configuration library must support all three popular ways for configuration access: configuration files, command-line options, and environment variables.

Cascading


/
PROC
(process specific, e. g., command-line arguments)
DIR
(directory specific, e. g., working directory)
USER
(user specific)
SYSTEM
(system wide)
/key1
dir:/key1
/key2
user:/key2
/key3
dir:/key3
system:/key3

Task

Discuss the differences of mounting and cascading with your neighbor.

User View

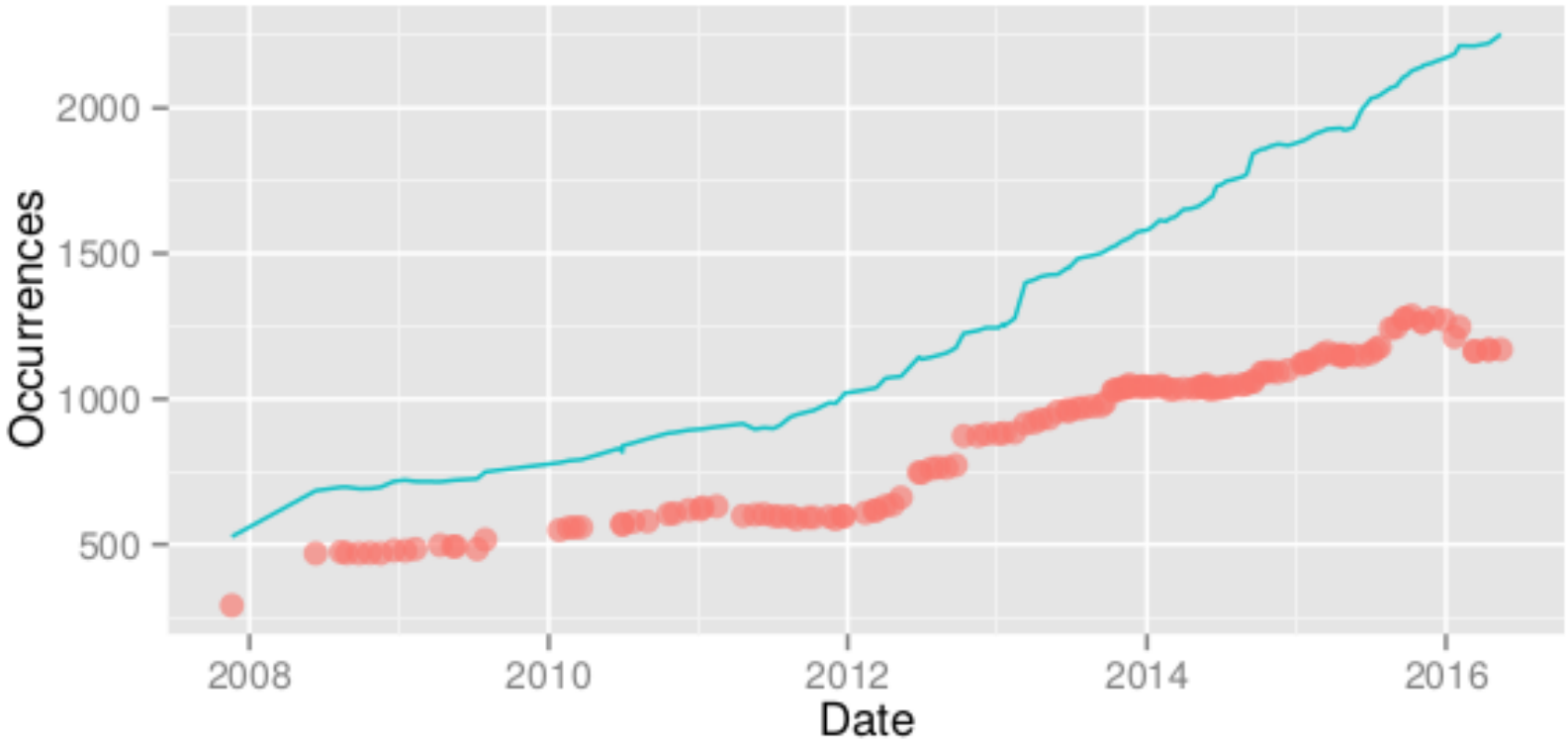
- command-line for trying out configuration settings
- environment variables for configuration settings within a shell
- configuration files for persistent configuration settings

Conclusion

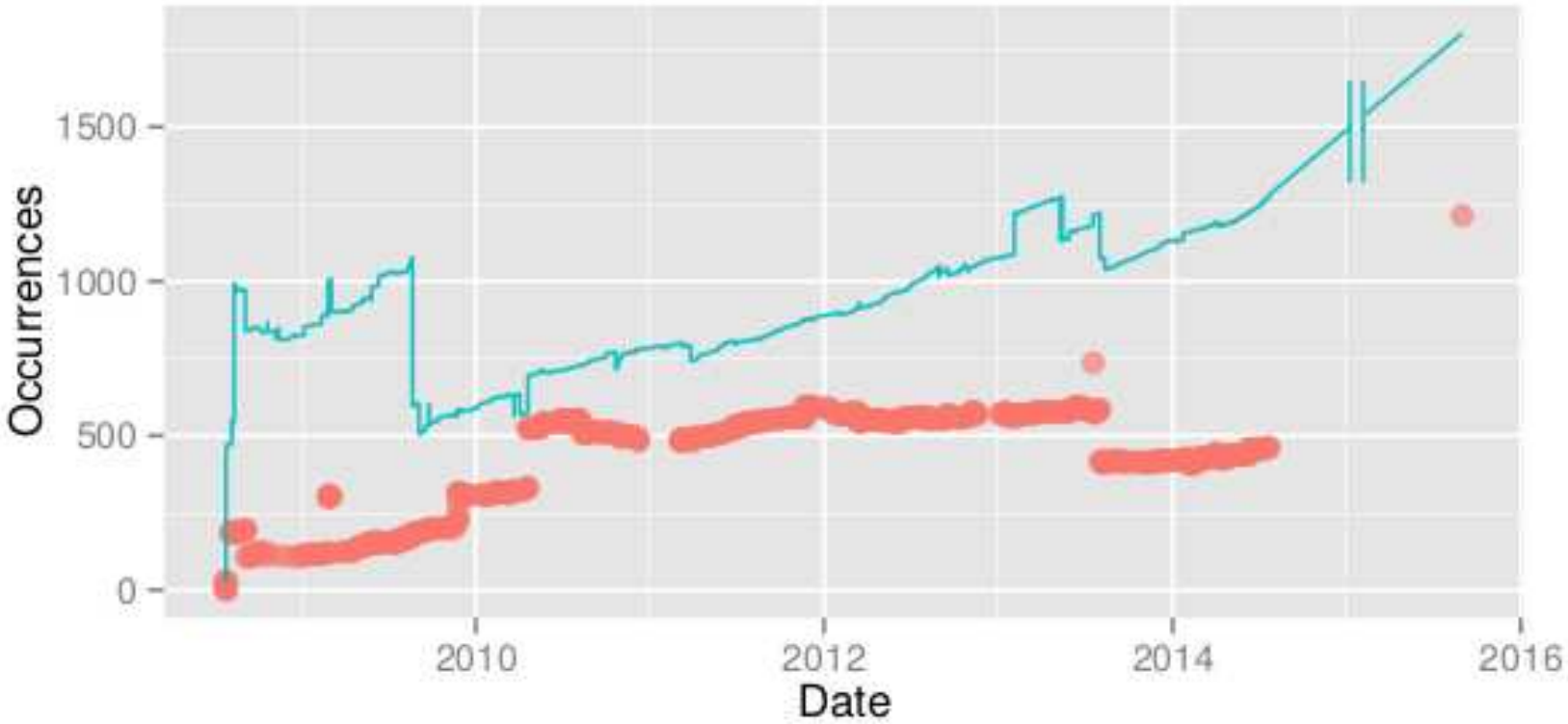
- three different configuration sources widely used
- all three used for different reasons but often for the same configuration settings
- many different configuration file formats
- abstractions: key-value, mounting, and cascading

Complexity

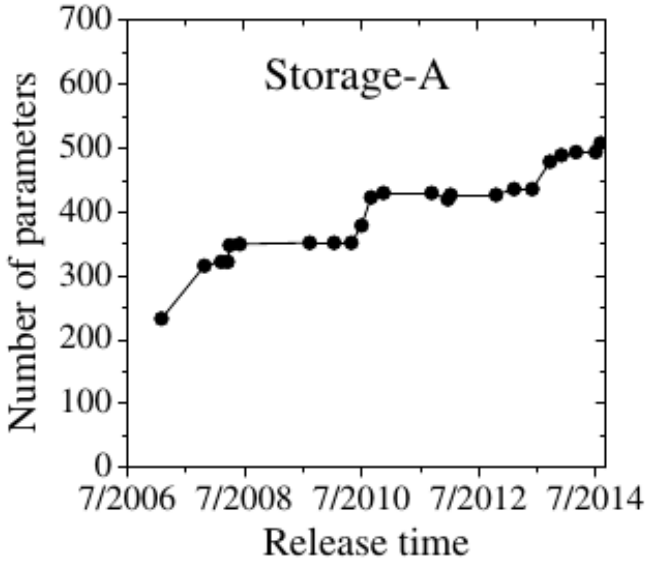
Trend Firefox

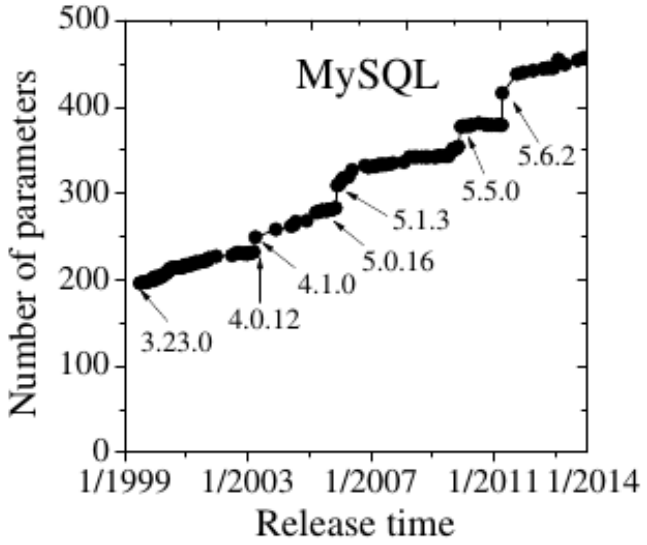


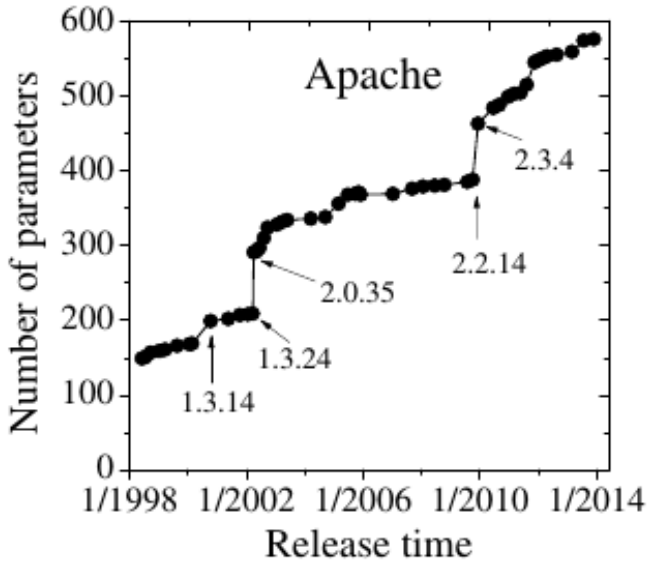
Trend Chromium

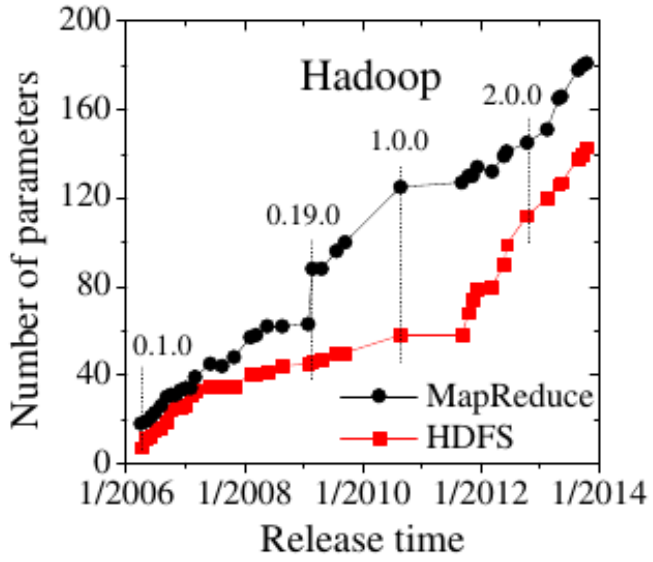


Trend Configuration Files









Xu et al. [9]

Types of Complexity

- complexity in access:
  - many different formats
  - non-uniformity
  - transformations
- configuration settings
  - number of settings s
  - number of values n
  - dependences between settings

Calculation of Complexity

Using enumerative combinatorics:

- number of configurations: n^s
- for N groups of different n and s (i.e., $n_1 \ldots n_N$ with $s_1 \ldots s_N$ occurrences):

$$\prod_{i=1}^{N} n_i^{s_i}$$

- more difficult to calculate (or unbounded) for dependences, module instantiations, arrays, ...

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd (let us assume $n = 2$): $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- 2000 boolean settings in Firefox [5]: $2^{2000} \approx 10^{602}$

Calculation of Complexity (cont.)

Examples:

- for 20 boolean and 20 enums with 5 possibilities:

$$2^{20} * 5^{20} = 10^{20}$$

- MySQL has 461 settings, of which 216 are non-simple types [9] (let us assume $n = \{3, 20\}$): $3^{245} * 20^{216} \approx 10^{397}$ (settings are explained in 5560 pages[1])
- an array with $1 - 20$ boolean settings: 2^{20}

[1] https://downloads.mysql.com/docs/refman-5.7-en.pdf

Task

Calculate complexity for some tool you know.

Task

Possible Homework: Write tool to calculate complexity with a given configuration specification.

Decision Tree

- configuration settings may depend on each other
- form a decision tree [2, 8]
- the decision tree is an instantiation of chosen configuration settings
- calculation only needs to consider instantiations which make a difference:
  essential configuration complexity [6]

Harmful Defaults [9]

- Problem: Two major data losses on a dozen machines.
- Cause: Stayed with the default values of the data-path settings (e.g., `dfs.name.dir`, `dfs.data.dir`) which point to locations in `/tmp`. Thus, after the machines reboot, data losses occur. "One of the common problems from users." (from Cloudera)
- up to 53 % of misconfigurations is due to staying at defaults
- 17 % to 48 % of configuration issues are about difficulties in finding settings

Unnecessary Settings [9]

- Configuration Parameter: `dfs.namenode.tolerate.heartbeat.multiplier`
- Developers' Discussion: Since we are not sure what is a good choice, how about making it configurable? We should add a configuration option for it. Even if it's unlikely to change, if someone does want to change it they'll thank us that they don't have to change the code/recompile to do so.
- Real-World Usage:
  - No usage found by searching the entire mailing lists and Google.
  - No usage reported in a survey of 15 Hadoop users in UCSD.

Unnecessary Settings [9]

- 6 % to 17 % of settings set by majority
- up to 54 % are seldom set
- up to 47 % of numeric settings have no more than five distinct values

Reduction

Q: "Why do you think configuration should be reduced?"

- to simplify code maintenance (50 %),
- to prevent errors and misconfiguration (43 %),
- to provide better user experience (40 %),
- ***"I do not think it should be reduced"*** **(30 %)**,
- because they prefer auto-detection (29 %)
  (with a possibility to override configuration settings: 32 %),
- *"because use-cases which are rarely used should not be supported"* (13 %),
- *"never find time for this task"* (9 %), and
- *"because only standard use-cases should be supported"* (1 %)

Question

How to specify reduction strategies of configuration settings?

Answer

Configuration Specification

Configuration Specification

Rationale

- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss
- essential for ***no-futz computing*** Holland et al. [4]
- the foundation for any advanced tooling like configuration management tools
- needed as communication of producers and consumers of configuration

Task

Brainstorming: What can be part of a configuration specification?

Task

Advantages/Disadvantages?

Task

Alternatives?

Q: "Configuration specification (e.g. XSD/JSON schemas) allows you to describe possible values and their meaning. Why do/would you specify configuration?"

- 58 % for *"looking up what the value does"*,
- 51 % it helps users to avoid common errors (*"so that users avoid common errors"*),
- 46 % to simplify maintenance,
- 40 % for rigorous validation,
- 39 % for documentation generation (for example, man pages, user guide),
- 30 % for external tools accessing configuration,
- 28 % for generating user interfaces,
- 25 % for code generation, and
- 24 % for specification of links between configuration settings.

Limitations of Schemata designed for Data

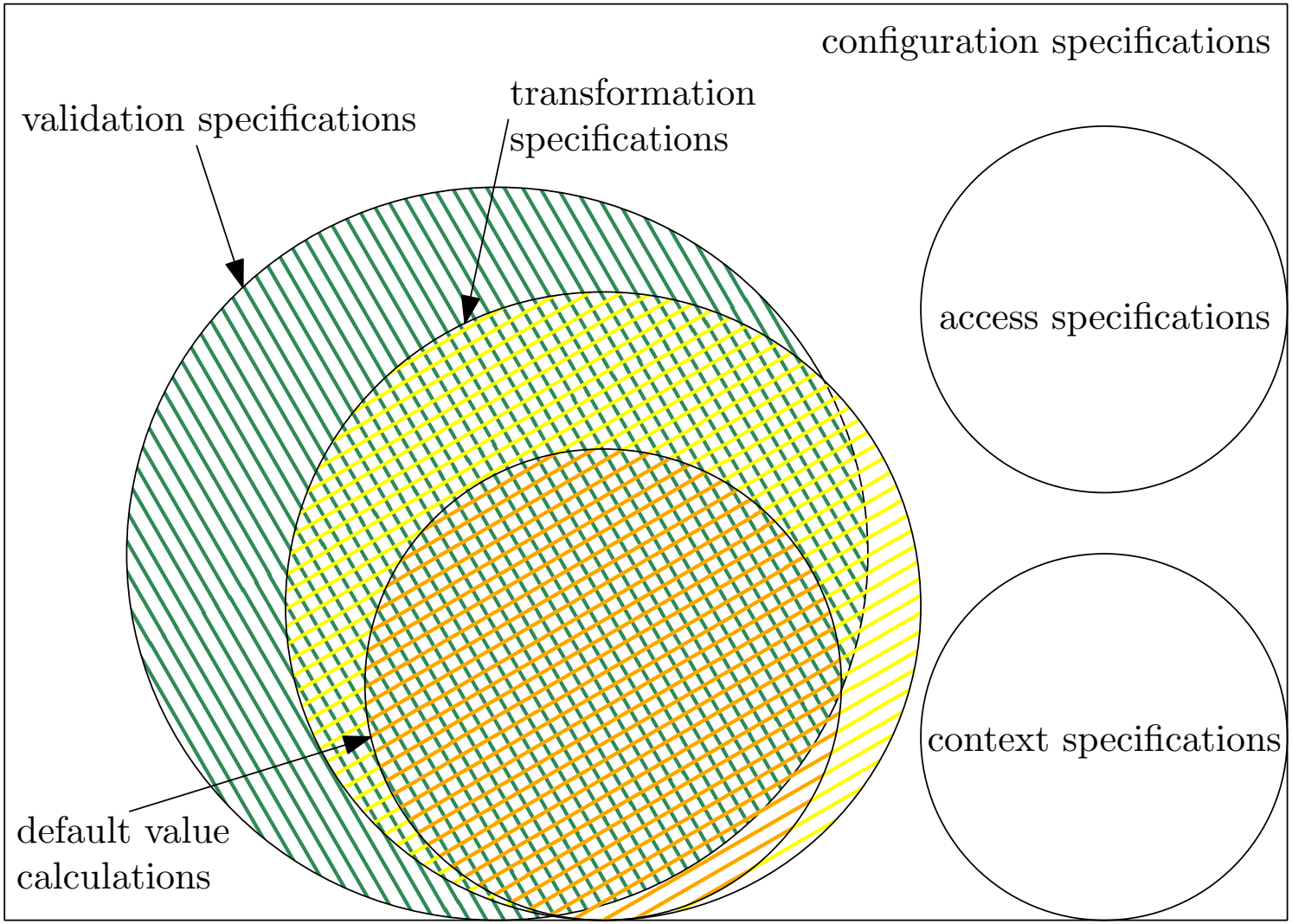
- like XSD/JSON schemas
- they are already very helpful but:
- not key-value based
- not easy to introspect
- designed to validate data without semantics:
  file path vs. presence of file
- not always possible to extend with plugins
- tied to specific formats (e.g. XML/JSON)

Limitations of Zero-Configuration

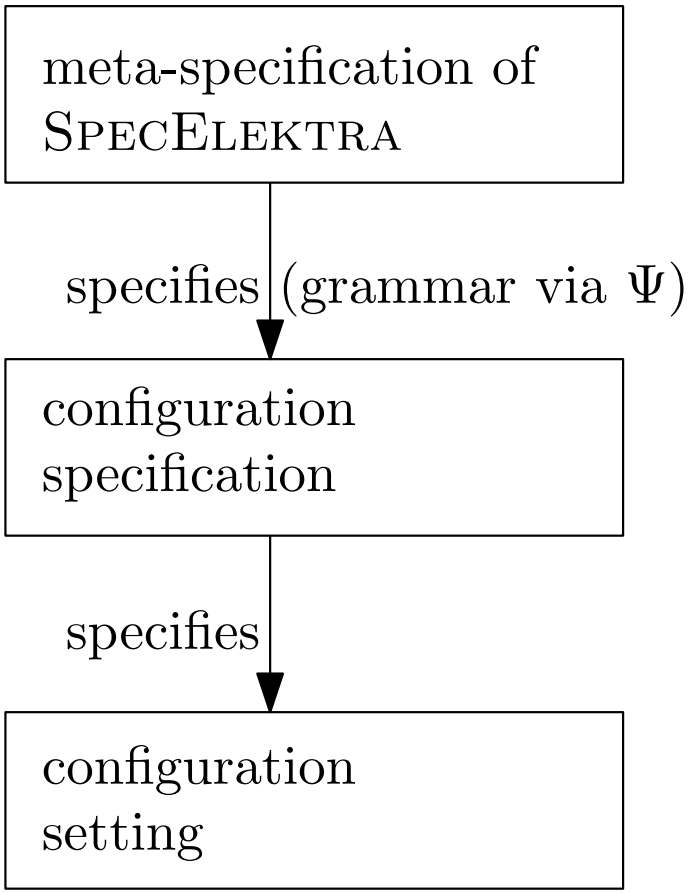
- e.g. gpsd[1]
- broken hardware or protocols
- auto-detection may go wrong
- the configuration actually lives elsewhere (e.g., in the GPS devices)

[1] www.aosabook.org/en/gpsd.html

Types of Specifications



Metalevels


meta-specification of SpecElektra
specifies (grammar via Ψ)
configuration specification
specifies
configuration setting

Task

What do we mean with a configuration specification?

Task

Which requirements do we have for a configuration specification?

Requirements

- formal/informal?
- complete?
- should be extensible
- should be external to application
- open for introspection (for tooling)
- should talk to users
- should allow generation of artefacts

Grammar

$\langle \textit{configuration specifications} \rangle ::= \{ \langle \textit{configuration specification} \rangle \}$

$\langle \textit{configuration specification} \rangle ::= \text{'['} \langle \textit{key} \rangle \text{']'} \langle \textit{properties} \rangle$

$\langle \textit{properties} \rangle ::= \{ \langle \textit{property} \rangle \}$

$\langle \textit{property} \rangle ::= \langle \textit{property name} \rangle \text{':='} [\langle \textit{property value} \rangle]$

Example

```
1 [slapd/threads/listener]
2 default:=1
3 type:=int
```

Visibility

- idea: show only relevant settings for specific user group
- or disallow editing: accessibility
- requires user-feedback loops [9]
- most-used settings should be best visible (or even enforce them to be changed: against harmful defaults)
- think of your users (administrators), only expose what users need
- write an rationale why someone needs it

Example

```
1 [slapd/threads/listener]
2 visibility:=developer
3
4 [slapd/access/#]
5 visibility:=user
```

Task

Brainstorming: Now, how do we implement such a specification?

Implementations

For example:

- generate examples/documentation
- auto-completion/syntax highlighting/IDE support
- tooling (GUI, Web UI)
- validate configuration files
- visudo-like
- plugins in configuration framework

- idea: make default value better
- is the generalization of sharing configuration values
- can be combined with visibility
- can be derived from other configuration settings
- can be derived from context [7]
- can be derived from hardware/system (problem with dependences)
- XServer vs. gpsd

Examples

Sharing:

```
1 [slapd/threads/listener]
2 fallback/#0:=slapd/threads
```

Percentages
(e.g., configured image should be additionally cropped):

```
1 [image/width]
2 type:=int
3
4 [crop]
5 type:=int
6 check/range:=0-100
```

Examples

Context:

```
1 [slapd/threads/listener]
2 context :=/slapd/threads/%cpu%/listener
```

Calculation with Context
(e.g., switch off GPS if battery low):

```
1 [gps/status]
2 assign :=(battery > 'low') ? ('on') : ('off')
```

Architectural Decisions

Software Architecture

- architecture is high-level description of the overall system
- use ready-made patterns and templates for architecture
- e.g., `http://arc42.org/`
- architectural decisions [3] essential (e.g., Chapter 9 in arc42)

Architectural Decisions

- describe decisions that lead to the architecture
- open decisions are high-level configuration
- useful to have patterns [3] and templates, too
- template: problem, constraints, assumptions, considered alternatives, decision, rationale, implications, related, notes

Why are configuration settings added?

The typical reasons are:

1. a requirement,
2. an architectural decision,
3. a technical need, and
4. an ad hoc decision.

in Configuration Specification

```
1 [slapd/threads/listener]
2 description:=adjust to use more threads
3 rationale:=needed for many-core systems
4 requirement:=1234
5 visibility:=developer
```

Conclusion

- alarming trend in number and complexity of configuration settings
- sharing, visibility and default value calculation often helps
- needs abstraction: configuration specification
- but also more courageous decisions and periodical reevaluation
- different ways to reduce configuration space

[1] *getenv(3) Linux User's Manual*, March 2017.

[2] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: A comparison of variability modeling approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1058-1. doi: 10.1145/2110147.2110167. URL `http://dx.doi.org/10.1145/2110147.2110167`.

[3] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *Software, IEEE*, 24(4):38–45, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.124.

[4] David A. Holland, William Josephson, Kostas Magoutis, Margo I. Seltzer, Christopher A. Stein, and Ada Lim. Research issues in no-futz computing. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 106–110. IEEE, May 2001. doi: 10.1109/HOTOS.2001.990069.

[5] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson. Configurations everywhere: Implications for testing and debugging in practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 215–224, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591191. URL http://dx.doi.org/10.1145/2591062.2591191.

[6] J. Meinicke, C. P. Wong, C. Kästner, T. Thüm, and G. Saake. On essential configuration complexity: Measuring interactions in highly-configurable systems. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 483–494, Sept 2016.

[7] Markus Raab and Gergö Barany. Introducing context awareness in unmodified, context-unaware software. In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,*, pages 218–225. INSTICC, ScitePress, 2017. ISBN 978-989-758-250-9. doi: 10.5220/0006326602180225.

[8] Mark-Oliver Reiser. *Core Concepts of the Compositional Variability Management Framework (CVM): A Practitioner's Guide*. TU, Professoren der Fak. IV, 2009.

[9] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs! Understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 307–319, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786852. URL http://dx.doi.org/10.1145/2786805.2786852.