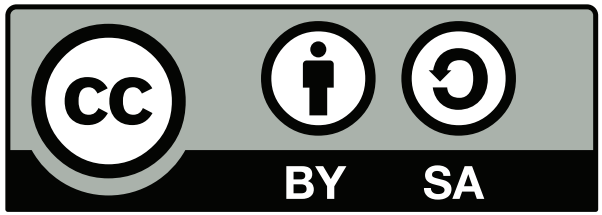
Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

16.3.2018



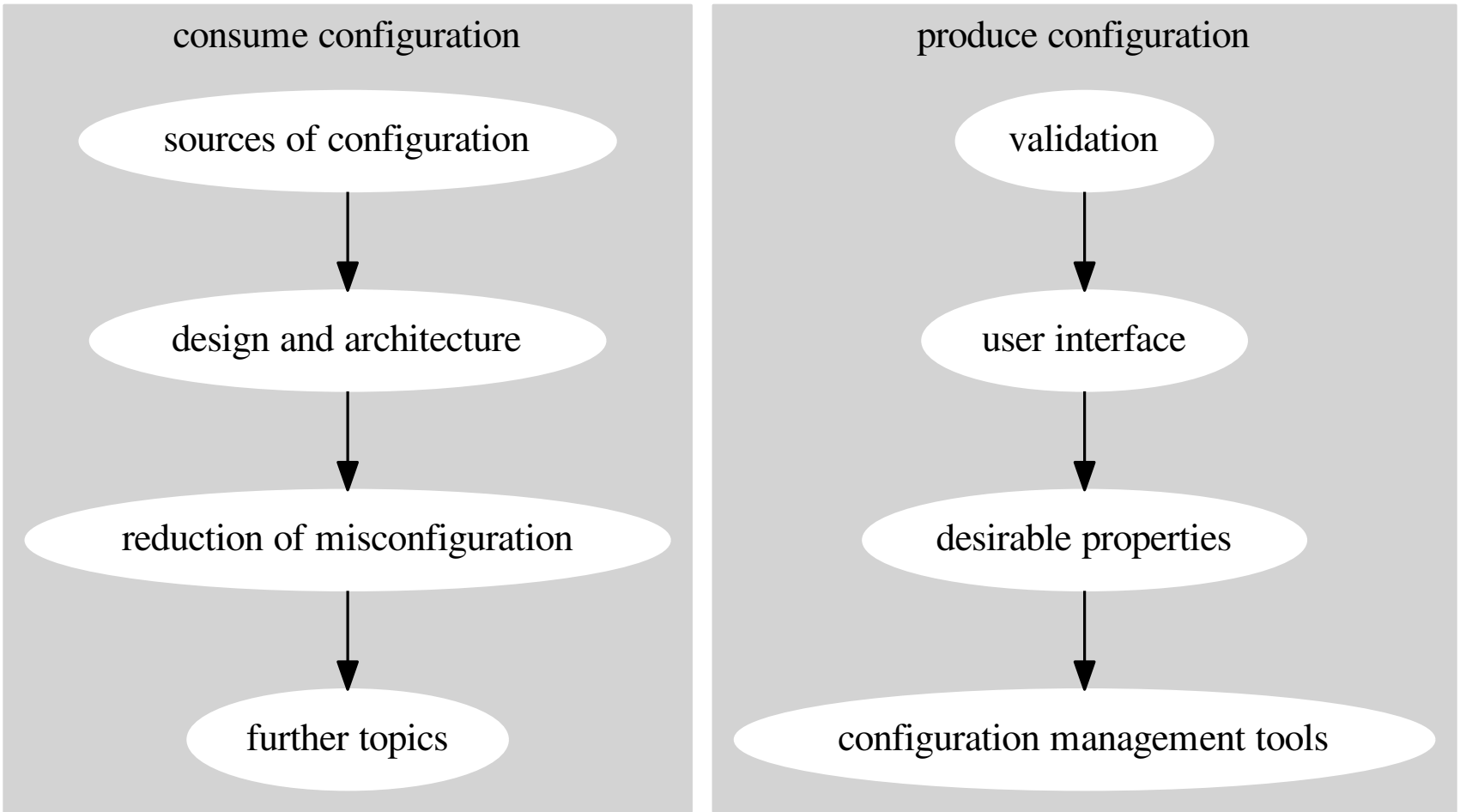
Organization

Next lectures:

- 16.3.2018: **topic homework and talk (GitHub account!)**
- 23.3.2018: teams found together
- 13.4.2018: homework submitted, topics of team exercise
- 20.4.2018: no lecture
- 18.5.2018: guest lecture
- 25.5.2018: team exercise submitted
- 22.6.2018: last corrections of team exercise

Popular Topics

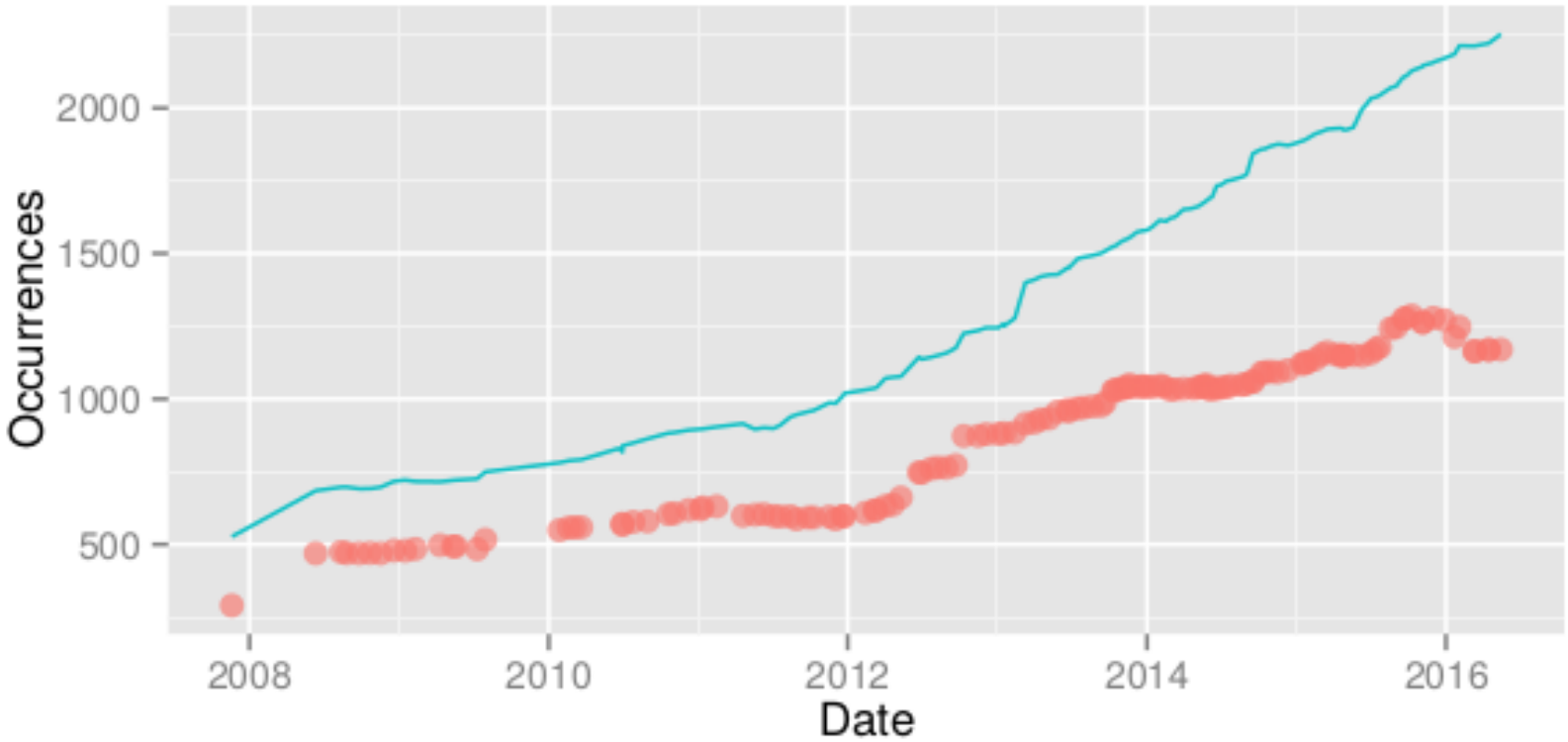
- 4 validation
- 4 user interface
- 3 tools (benefits?)
- 3 testability
- 3 complexity reduction (when conf. needed?)
- 3 architectural decisions
- 2 Puppet
- 2 modularity
- 2 environment variables
- 2 documentation
- 2 configuration specification
- 2 command-line args
- 2 code generation
- 1 variability
- 1 self-description
- 1 round-tripping
- 1 introspection
- 1 early
- 1 dependences
- 1 context-awareness
- 1 auto-detection
- 1 administrators


consume configuration
sources of configuration
design and architecture
reduction of misconfiguration
further topics
produce configuration
validation
user interface
desirable properties
configuration management tools

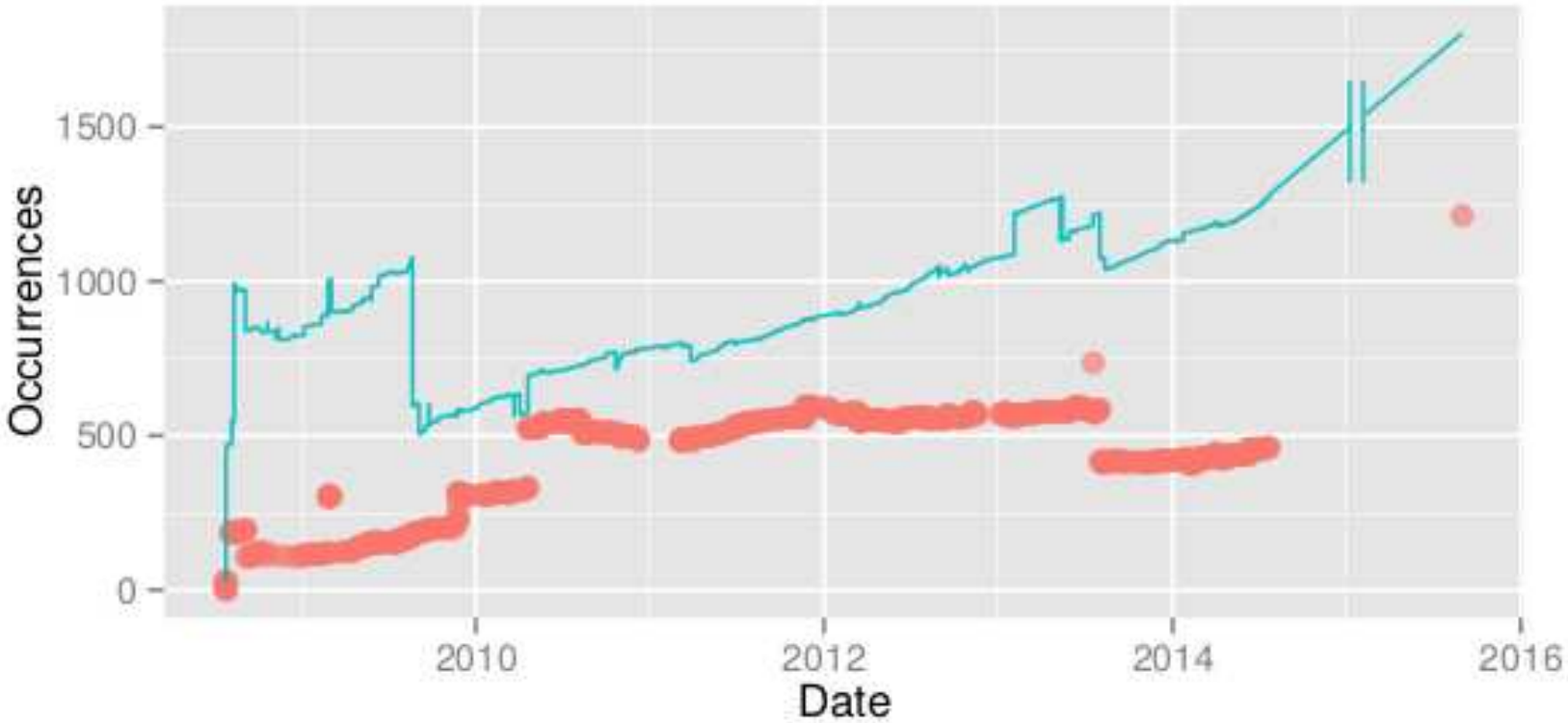
Complexity

1. Complexity
   - Trend
   - Calculation
   - Usage
2. Configuration Specification
   - Why?
   - How?
   - Visibility
   - Calculate Default Values
3. Architectural Decisions

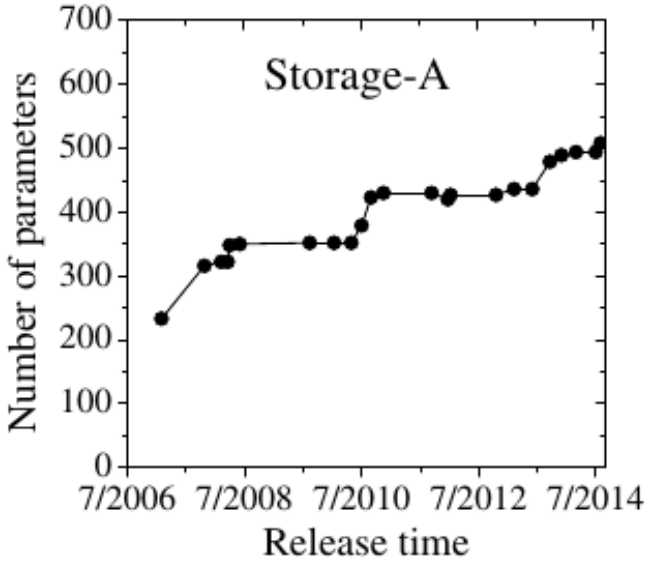
Trend Firefox

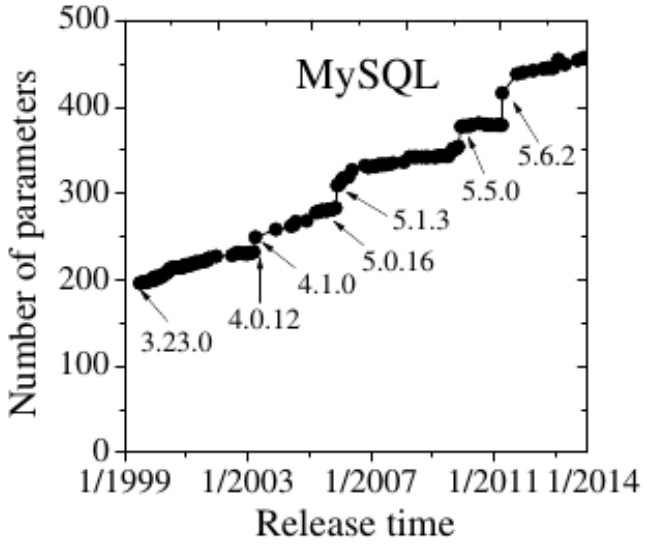


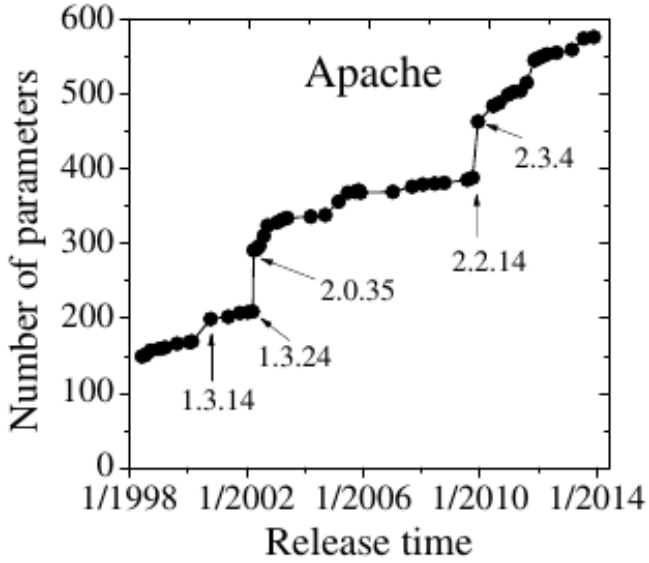
Trend Chromium

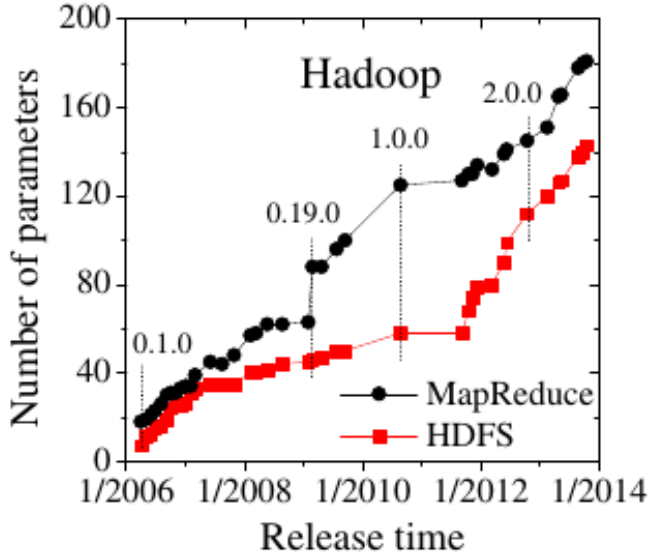


Trend Configuration Files









Xu et al. [6]

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., $n_1 \dots n_N$ with $s_1 \dots s_N$ 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:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd: $2^{600} \approx 10^{180}$
- 19 integer settings:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd: $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- 2000 boolean settings in Firefox:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd: $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- 2000 boolean settings in Firefox: $2^{2000} \approx 10^{602}$
- for 20 boolean and 20 enums with 5 possibilities:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd: $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- 2000 boolean settings in Firefox: $2^{2000} \approx 10^{602}$
- for 20 boolean and 20 enums with 5 possibilities:

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

- an array with $1 - 20$ boolean settings:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd: $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- 2000 boolean settings in Firefox: $2^{2000} \approx 10^{602}$
- for 20 boolean and 20 enums with 5 possibilities:

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

- an array with $1 - 20$ boolean settings: 2^{20}

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 [1, 5]
- the decision tree is an instantiation of chosen configuration settings
- calculation only needs to consider possible instantiation

Unnecessary Settings [6]

- 6 % to 17 % of settings set by majority
- up to 54 % are seldom set
- up to 47 % of numeric parameters have no more than five distinct values
- 17 % to 48 % of configuration issues are about difficulties in finding settings

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?

Question

How to specify reduction strategies of configuration settings?

Answer

Configuration Specification

Configuration Specification

1. Complexity
   - Trend
   - Calculation
   - Usage
2. Configuration Specification
   - Why?
   - How?
   - Visibility
   - Calculate Default Values
3. Architectural Decisions

Rationale

- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss

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. [3]
- the foundation for any advanced tooling like configuration management tools

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. [3]
- 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.

Limitation of Schemata designed for Data

- like XSD/JSON schemas
- they are already very helpful but:

Limitation 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 write own plugins
- tied to specific formats (e.g. XML/JSON)

Limitation of Zero-Configuration

- e.g. gpsd[1]

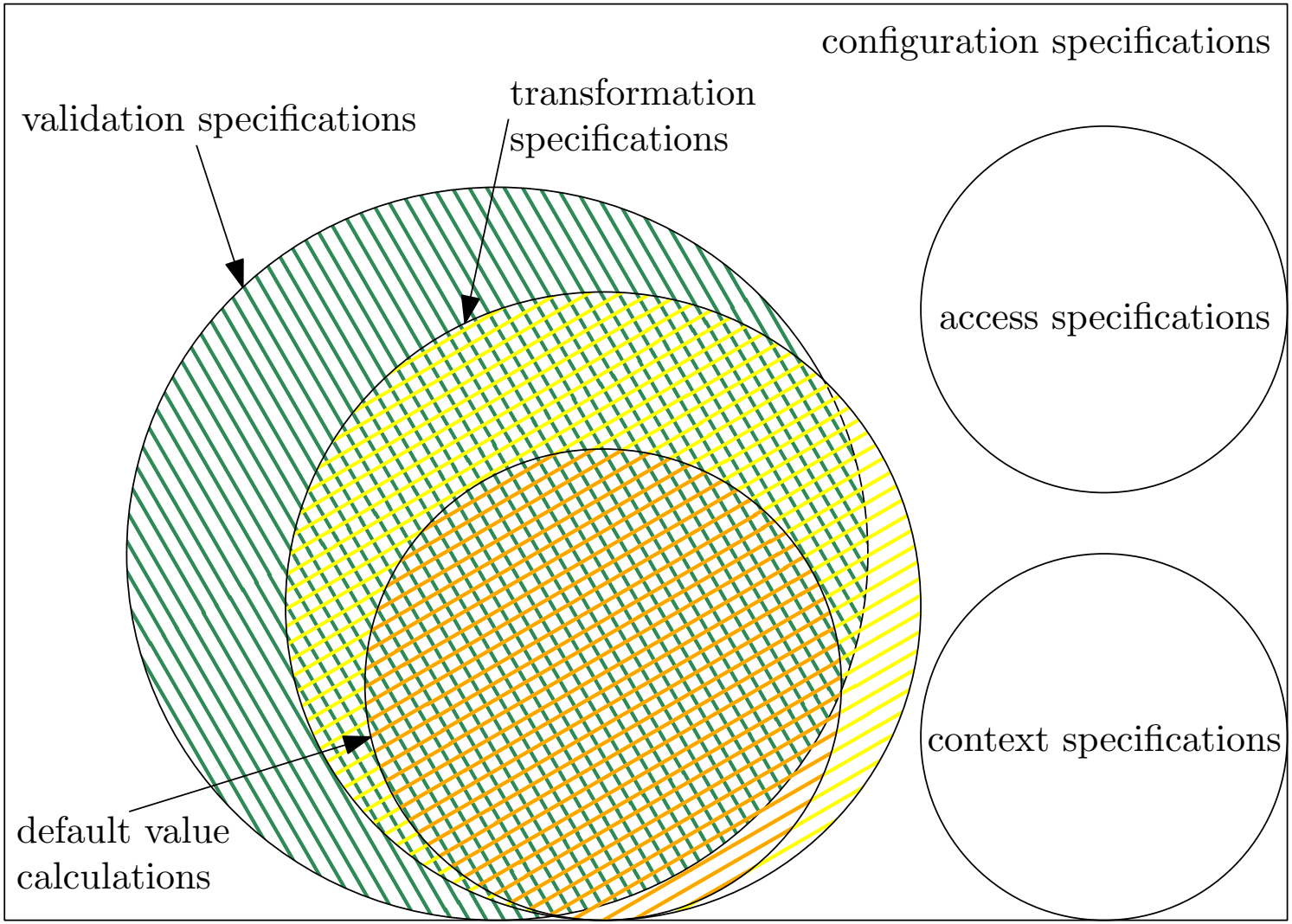
[1] www.aosabook.org/en/gpsd.html

Limitation of Zero-Configuration

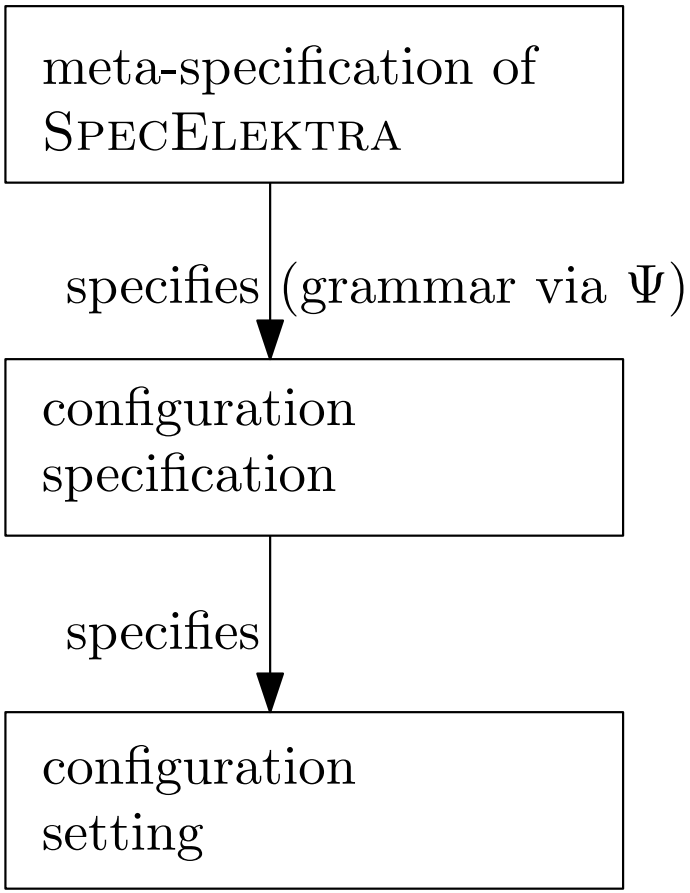
- e.g. gpsd[1]
- broken hardware or protocols
- auto-detection may go wrong
- the configuration actually lives elsewhere (e.g., in the GPS devices)

[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?

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

Visibility

- idea: show only relevant settings for specific user group
- or disallow editing: accessibility
- requires user-feedback loops [6]
- most-used settings should be best visible
- 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

- idea: make default value better
- can be combined with visibility

- idea: make default value better
- can be combined with visibility
- can be derived from other configuration settings
- can be derived from context [4]
- can be derived from hardware/system (problem with dependences)

- idea: make default value better
- can be combined with visibility
- can be derived from other configuration settings
- can be derived from context [4]
- can be derived from hardware/system (problem with dependences)
- XServer vs. gpsd

Examples

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

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

Architectural Decisions

1. Complexity
   - Trend
   - Calculation
   - Usage
2. Configuration Specification
   - Why?
   - How?
   - Visibility
   - Calculate Default Values
3. Architectural Decisions

software architecture

- architecture is high-level description of the overall system
- use ready-made patterns and templates for architecture

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 [2] 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 [2] and templates, too
- template: problem, constraints, assumptions, considered alternatives, decision, rationale, implications, related, notes

Why are configuration settings added?

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
- visibility and default value calculation helps
- needs abstraction: configuration specification
- but also more courageous decisions and periodical reevaluation
- different ways to reduce configuration space

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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`.