

## L07 Strategies for Reduction of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

05.05.2021

This work is licensed under a Creative Commons  
“*Attribution-ShareAlike 4.0 International*” license.



# Terms and Properties

- 1 Terms and Properties
- 2 Pitfalls
- 3 Unused Settings
- 4 Meeting
  - Recapitulation
  - Assignments
  - TW

# Learning Outcomes

Students will be able to

- remember terms of properties of CM.
- remember various strategies for reduction of misconfiguration.
- find unused settings.

# Infrastructure as Code

Configuration settings are an instantiation of the configuration specifications.

The instantiation code is implemented by **CM code**.

**Auditability:** Being informed about status and changes in the infrastructure.

## Goal

Single Source of Truth

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)



# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration
- **applying different versions of CM code**

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration
- applying different versions of CM code
- **non-idempotent CM Code**

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration
- applying different versions of CM code
- non-idempotent CM Code
- . . .

# Idempotence

idem + potence (same + power)

Yield same result with any number of applications ( $n \geq 1$ ):

$$f(f(x)) = f(x)$$

## Example

Hummer et al. [1] tested 298 Chef scripts, of which 92 were non-idempotent:

- `/etc/timezone` rewritten by package tzdata

# Example

Hummer et al. [1] tested 298 Chef scripts, of which 92 were non-idempotent:

- `/etc/timezone` rewritten by package `tzdata`
- `tomcat6`: files copied by user if `/etc/tomcat6/tomcat6.conf` does not exist but copy fails because later step creates `/etc/tomcat6/logging.properties` as root.

# Example

Hummer et al. [1] tested 298 Chef scripts, of which 92 were non-idempotent:

- `/etc/timezone` rewritten by package `tzdata`
- `tomcat6`: files copied by user if `/etc/tomcat6/tomcat6.conf` does not exist but copy fails because later step creates `/etc/tomcat6/logging.properties` as root.
- `mongodb`: if installation fails, the group “mongodb” does not exist, failing at later tasks creating directories using this group



Siméon and Wadler [5] describe two further properties:

**Self-describing** means that from the configuration file alone we are able to derive the correct data structure [5].

**Round-tripping** means that if a data structure is serialized and then parsed again, we end up with an identical data structure [5].

The data structure could be a KeySet.

Siméon and Wadler [5] describe two further properties:

**Self-describing** means that from the configuration file alone we are able to derive the correct data structure [5].

**Round-tripping** means that if a data structure is serialized and then parsed again, we end up with an identical data structure [5].

The data structure could be a KeySet.

Round-tripping is a prerequisite of idempotence.

# Examples

XML has neither of the last two properties Siméon and Wadler [5]:

- internal representation crucially depends on XML schema
- union of integer and strings

```
1 intOrStr { "one", "2", 3 }  
2 <fact>one 2 3</fact>  
3 intOrStr { "one", 2, 3 }
```

## L07 Strategies for Reduction of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

05.05.2021

This work is licensed under a Creative Commons  
“*Attribution-ShareAlike 4.0 International*” license.



# Pitfalls

- 1 Terms and Properties
- 2 Pitfalls
- 3 Unused Settings
- 4 Meeting
  - Recapitulation
  - Assignments
  - TW

## Harmful Defaults [7]

- 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

## Question

What do we want to test?

## Question

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [6])
- Regression tests [4]



## Question

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [6])
- Regression tests [4]
- Are all settings implemented?
- Are all settings used in tests?
- Are there unused settings in the code?

Matt Welsh from Google wrote in 2013:<sup>1</sup>

*“Of course we have extensive testing infrastructure, but the ‘hard’ problems always come up when running in a real production environment, with real traffic and real resource constraints. Even integration tests and canarying are a joke compared to how complex production-scale systems are.”*

---

<sup>1</sup>What I wish systems researchers would work on. Retrieved from <http://matt-welsh.blogspot.com/2013/05/what-i-wish-systems-researchers-would.html>.

## Jin et al. [2]

- Wants to improve configuration-aware testing and debugging
- Manual investigations for three applications
- Finds 1957 settings in Firefox ( $2^{846} * 3^{1111}$ ) and 36322 in LibreOffice ( $2^{4433} * 3^{31889}$ )
- Finds unused settings: settings only in the source code
- Finds configuration settings which are not specified

### Requirement

*Configuration setting traceability is a necessity.*

### Idea

Code generation helps to trace settings and to find unused settings.

## Testing by developers:

- ConfErr [3] uses models of key board layout, psychology and linguistics. Tool injects possible misconfiguration.

## Testing by developers:

- ConfErr [3] uses models of key board layout, psychology and linguistics. Tool injects possible misconfiguration.
- Spex [6] analyzes the source code to find misconfigurations. As by-product it extracts internal specifications (including transformation bugs).

## Testing by developers:

- ConfErr [3] uses models of key board layout, psychology and linguistics. Tool injects possible misconfiguration.
- Spex [6] analyzes the source code to find misconfigurations. As by-product it extracts internal specifications (including transformation bugs).
- External specification can be directly used to generate test cases.

## Testing by developers:

- ConfErr [3] uses models of key board layout, psychology and linguistics. Tool injects possible misconfiguration.
- Spex [6] analyzes the source code to find misconfigurations. As by-product it extracts internal specifications (including transformation bugs).
- External specification can be directly used to generate test cases.
- Find unused configuration settings.

## L07 Strategies for Reduction of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

05.05.2021

This work is licensed under a Creative Commons  
“*Attribution-ShareAlike 4.0 International*” license.

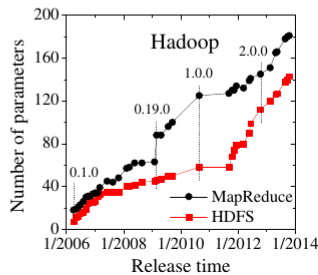
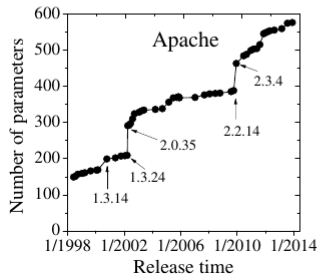
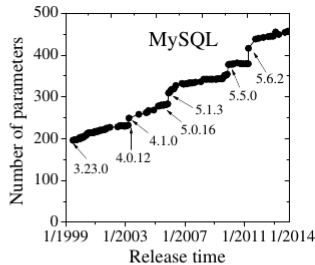
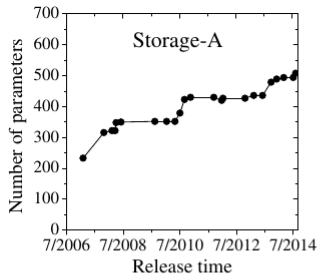




## Unused Settings

- 1 Terms and Properties
- 2 Pitfalls
- 3 Unused Settings**
- 4 Meeting
  - Recapitulation
  - Assignments
  - TW

# Trend Configuration Files (Recapitulation)



Xu et al. [7]

## Unnecessary Settings [7]

- 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 [7]

- 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 %)

# Limitations of Zero-Configuration

- e.g. gpsd<sup>1</sup>

---

<sup>1</sup>[www.aosabook.org/en/gpsd.html](http://www.aosabook.org/en/gpsd.html)

# Limitations of Zero-Configuration

- e.g. `gpsd`<sup>1</sup>
- broken hardware or protocols
- auto-detection may go wrong
- the configuration actually lives elsewhere (e.g., in the GPS devices)

---

<sup>1</sup>[www.aosabook.org/en/gpsd.html](http://www.aosabook.org/en/gpsd.html)

# 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 [7]
- 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

# Visibility

- idea: show only relevant settings for specific user group
- or disallow editing: accessibility
- requires user-feedback loops [7]
- 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
- visibility should not be an excuse to add not-needed settings

# Example

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

# Find Unused Settings

The first (optional) step of the algorithm is:

- Run all tests with code coverage.
- Check if generated code is executed.
- If it is, we know that the configuration setting is used in a test case. Otherwise, we know it is not tested by the test suite. All these untested configuration settings are remembered as candidates for the second step.

```
1 KeySet findUnusedSettings (KeySet untestedSettings,
2                             KDB kdb,
3                             Builder build)
4 {
5     KeySet unusedSettings = {};
6     KeySet configurationSpecification;
7     kdb.get (configurationSpecification);
8
9     for (candidate: untestedSettings)
10    {
11        configurationSpecification.remove (candidate);
12        kdb.set (configurationSpecification);
13        build.recompile ();
14        if (build.wasSuccessful ())
15        {
16            unusedSettings.append (candidate);
17        }
18        configurationSpecification.append (candidate);
19    }
20
21    kdb.set (configurationSpecification);
22    return unusedSettings;
23 }
```

# Conclusion

- Definition and challenges in configuration management.

# Conclusion

- Definition and challenges in configuration management.
- Properties: self-describing, idempotent, round-tripping.

# Conclusion

- Definition and challenges in configuration management.
- Properties: self-describing, idempotent, round-tripping.
- Awareness and responsibility needed.



## Conclusion

- Definition and challenges in configuration management.
- Properties: self-describing, idempotent, round-tripping.
- Awareness and responsibility needed.
- Avoidance of misconfiguration is combined effort of devs and admins.

## L07 Strategies for Reduction of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

05.05.2021

This work is licensed under a Creative Commons  
“*Attribution-ShareAlike 4.0 International*” license.



# Meeting

- 1 Terms and Properties
- 2 Pitfalls
- 3 Unused Settings
- 4 **Meeting**
  - Recapitulation
  - Assignments
  - TW

# Infrastructure as Code

# Infrastructure as Code

Configuration settings are an instantiation of the configuration specifications.

The instantiation code is implemented by **CM code**.

**Auditability:** Being informed about status and changes in the infrastructure.

## Goal

Single Source of Truth

# Configuration Drift

# Configuration Drift

is a derivation of the “Single Source of Truth” (the CM code).

It is caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration
- applying different versions of CM code
- non-idempotent CM Code
- ...

# Properties (Recapitulation)

## Task

What is idempotent, self-describing, round-tripping configuration?



# Properties (Recapitulation)

## Task

What is idempotent, self-describing, round-tripping configuration?

**Idempotent** yield the same configuration with any number of applications from CM code ( $n \geq 1$ ) [1]:

$$f(f(x)) = f(x)$$

needed to guarantee repeatability

**Self-describing** means that from the configuration file alone we are able to derive the correct data structure [5].

**Round-tripping** means that if a data structure is serialized and then parsed again, we end up with an identical data structure [5].

The data structure could be a KeySet.

# Pitfalls of CM

# Pitfalls of CM

- too many (unused) settings
- no external specification
- too complicated configuration design
- harmful defaults
- no configuration-aware tests
- settings and specs not synchronized

# Reduction of Configuration Settings

# Reduction of Configuration Settings

- Zero-Configuration (auto detection)
- Visibility
- Find unused settings (code generator needed)

Slides for talk in TUWEL **and** private git repo:

- 12 May: submission
- 26 May: peer review

Today: Correction H2

Upcoming: H3, T3

# Today

- 1 Give yourselves roles (admin, dev, user)
- 2 Split the goals and tasks among them.



# Preparation

- 1 Do homework as described in L08 materials.

# Outlook

In the next lecture (LTW):

- 1 Create together an architecture that fulfils the goals.
- 2 Present that architecture, explain the concepts, show which technologies you found.

- [1] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In David Eyers and Karsten Schwan, editors, *Middleware 2013*, pages 368–388, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45065-5.
- [2] 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>.
- [3] Lorenzo Keller, Prasang Upadhyaya, and George Candea. Conferr: A tool for assessing resilience to human configuration errors. In *Dependable Systems and Networks With FTCS and DCC, 2008.*, pages 157–166. IEEE, 2008.

- [4] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 75–86, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-050-0. doi: 10.1145/1390630.1390641. URL <http://doi.acm.org/10.1145/1390630.1390641>.
- [5] Jérôme Siméon and Philip Wadler. The essence of xml. pages 1–13, 2003. doi: 10.1145/604131.604132. URL <http://dx.doi.org/10.1145/604131.604132>.
- [6] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.

- [7] 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>.