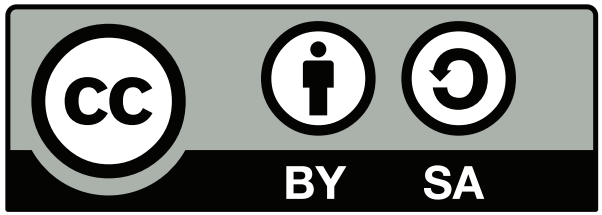
Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

8.6.2018



Organization

Schedule:

- 8.6.2018: lecture
- 15.6.2018: last corrections of team exercise
- 22.6.2018: oral test

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 early detection
- 1 introspection
- 1 dependences
- 1 auto-detection
- 1 context-awareness
- 1 administrators

Recapitulation

Introspection (Recapitulation)

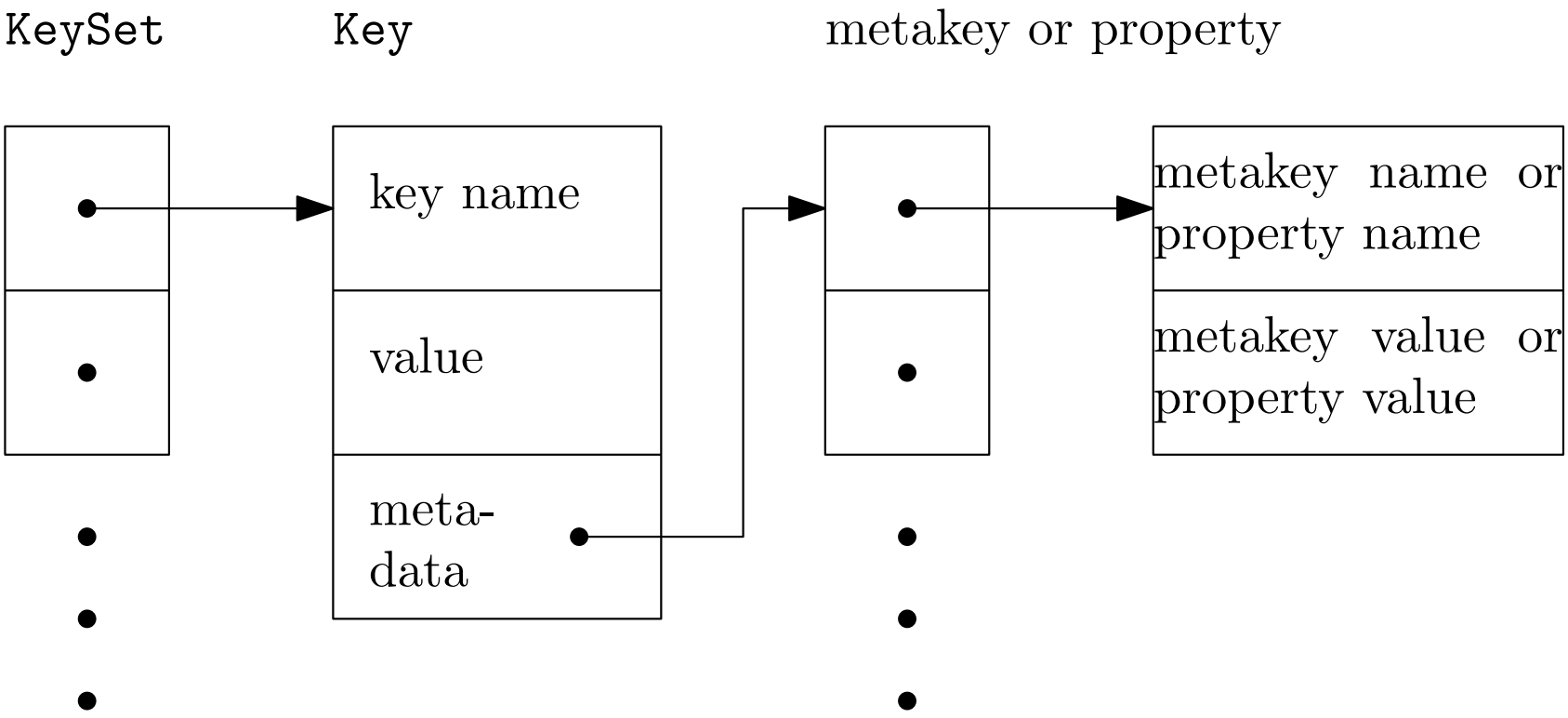
Task

What is internal and external specification? What is introspection?

- *internal*: within applications' source code
- *introspection*: unified get/set access to (meta*)-key/values
- access via applications, CLI, GUI, web-UI, ...
- access via any programming language (similar to file systems)
- GUI, web-UI can semantically interpret metadata
- assemble modular parts (validation, logging, ...)
- needed as communication between producers and consumers
- essential for ***no-futz computing*** Holland et al. [6]

KeySet (Recapitulation)

The common data structure of Elektra:



Testing (Recapitulation)

Task

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [17])
- Regression tests (devs [11])

- Are all settings implemented? (devs)
- Are all settings used in tests? (devs)
- Are there unused settings in the code? (devs)
- Do the chosen settings work? (admins)

Early detection (Recapitulation)

Task

When do we want to detect misconfiguration?

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)

Problem

Earlier versus more context.

Notification (Recapitulation)

Task

Why do we need notification?

1. to keep transient and persistent configuration settings always in sync [8]
2. to avoid polling of configuration settings
3. to better integrate into already existing mechanisms (main loops)[1]

Requirement

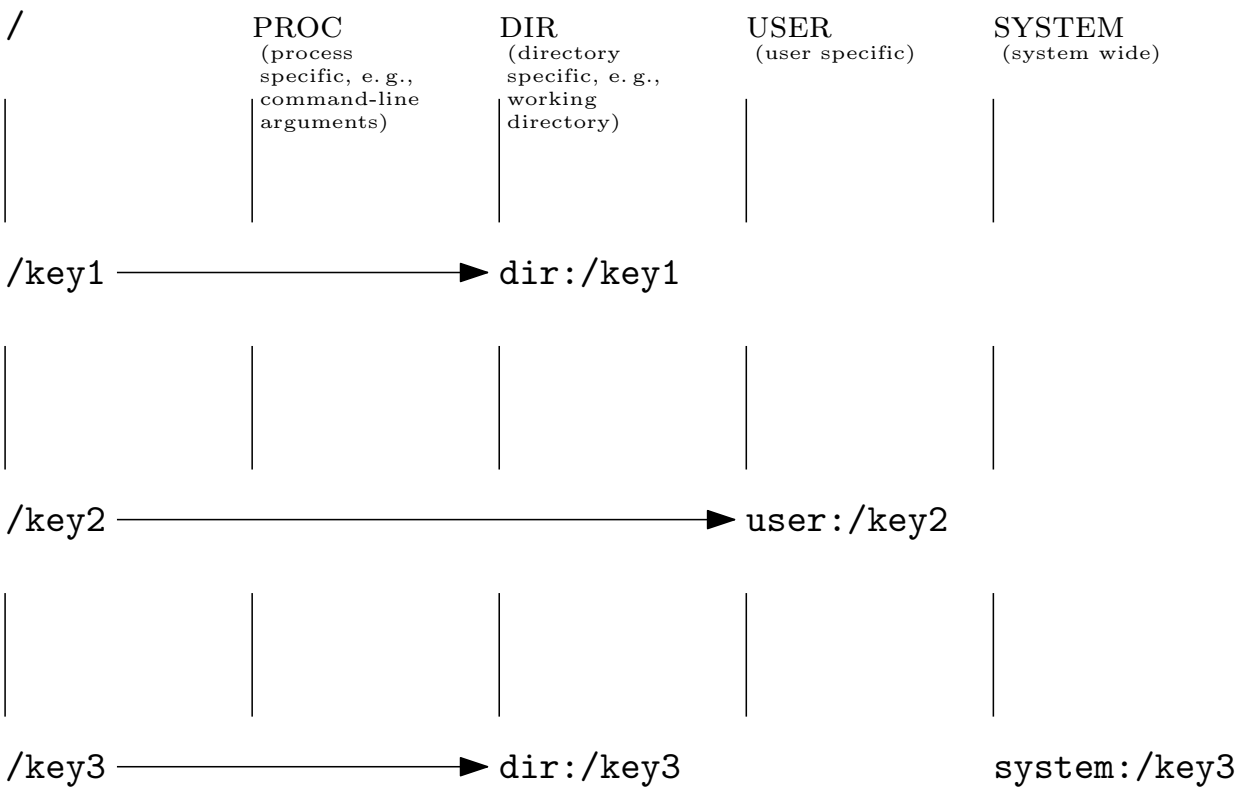
Configuration libraries must provide ways to keep transient and persistent views consistent.

[1] Is one of the main reasons why most framework already integrate configuration settings.

Cascading (Recapitulation)

Task

What is cascading configuration?



Contextual Values

Task

What are contextual values?

Tanter [15] introduced a lightweight extension to context-oriented programming: ***Contextual values*** are variables whose values depend on the context in which they are read and modified. They *"boil down to a trivial generalization of the idea of thread-local values"*. The key idea is to use layers as *"discriminate amongst possible values, not only the current thread"* [15]. Side effects are limited to the respective context [12].

Introspection vs. Code Generation (Recapitulation)

Task

Advantages/Disadvantages of key database (vs. code generation)?

- \+ specification can be updated live on the system without recompilation
- \+ tooling has generic access to all specifications
- \+ new features the key database (e.g., better validation) are immediately available consistently
- – less techniques for performance improvements
- – contextual values cannot be used if context differs within same thread

Implication

We generally prefer introspection, except for a very thin configuration access API.

Key Databases (Recapitulation)

Q: "Which configuration systems/libraries/APIs have you already used or would like to use in one of your FLOSS project(s)?"

- Command-line arguments (92 %, $n = 222$)
- environment variables (79 %, $n = 218$)
- configuration files (74 %, $n = 218$)
- Freedesktop standards (20 %, $n = 205$)
- Windows Registry (13 %) (≤ 13 %, $n \geq 185$) [talk later]
- X/Q/GSettings (4 %, 11 %, 9 %)
- KConfig (5 %)
- dconf (7 %)
- plist (7 %)

Definition Configuration Management (Recapitulation)

Task

What is Configuration Management?

- is a discipline in which configuration (in the broader sense) is administered.
- makes sure computers are assembled from desired parts and the correct applications are installed.
- has means to describe the desired configuration of the whole managed system.
- ensures that the execution environment of installed applications is as required.

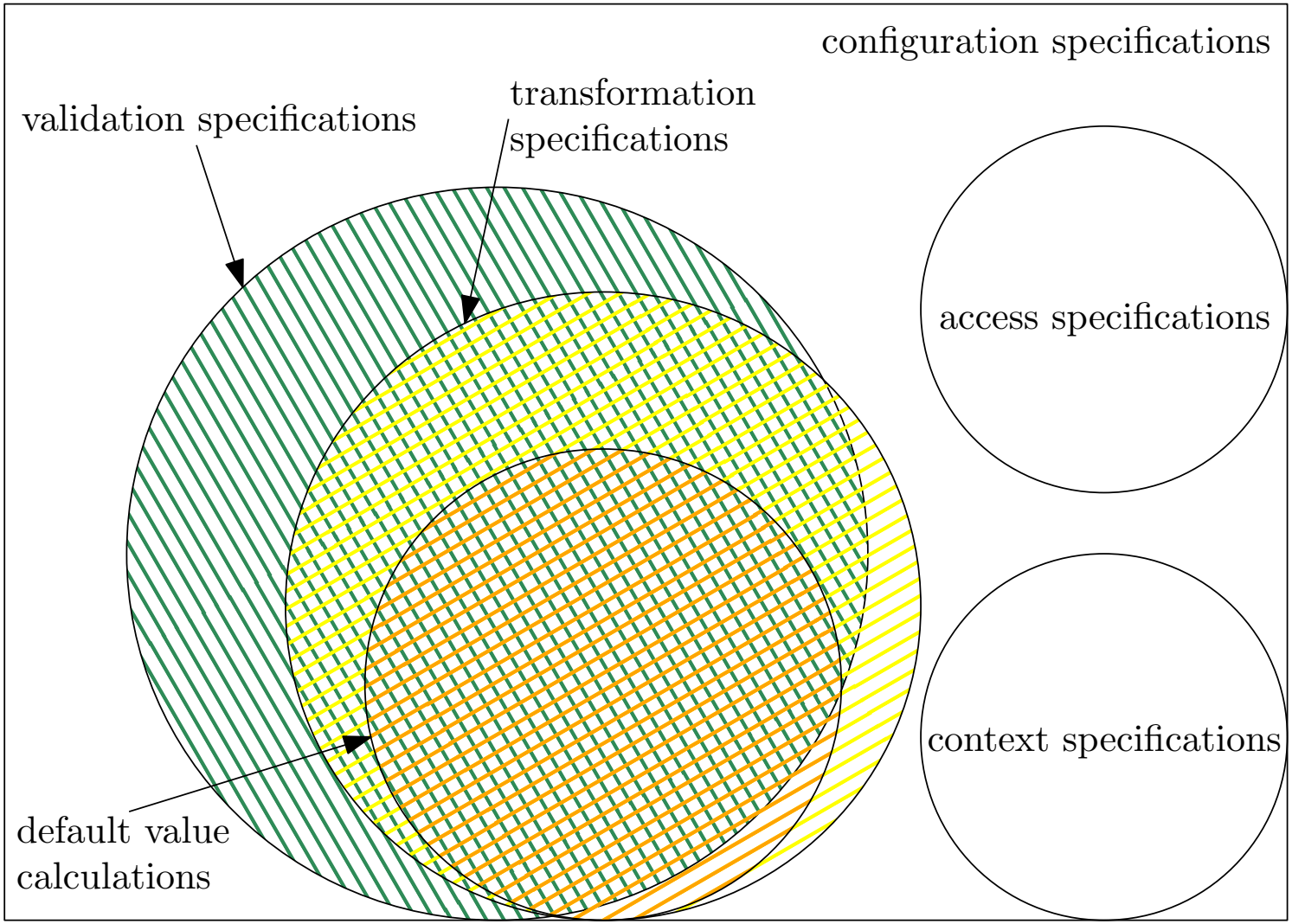
Possible Benefits of CM (Recapitulation)

Task

What are the goals of Configuration Management?

- The same goals scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  Single Source of Truth (Infrastructure as Code [7])
- Auditability
- Less configuration drift
- Error handling
- Pull/Push
- Reusability

Types of Specifications (Recapitulation)



Configuration Specification (Partly Recapitulation)

Task

How can we combine configuration specifications and configuration management? (Think, Pairs, Share)

- configuration settings are simply an instantiation of the configuration specifications. Code describing the instantiation is **CM code**.
- configuration design is explicit (like transformations and default values) and can help while writing CM code.
- CM code can even be generated from the specification.
- access specifications make access trivial via uniform interface.
- visibility and similar techniques may help dealing with complexity.

Configuration Drift (Recapitulation)

Task

What is configuration drift? What are its causes?

Are derivations of the "Single Source of Truth" (the CM code).
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
- ...

Push vs. Pull

Task

Explain the Push and the Pull Model. What are their (dis)advantages?

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.
- Pull needs resources even if nothing is to do.

Idempotence (Recapitulation)

Task

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

Idempotent yield the same configuration with any number of applications from CM code ($n \geq 1$) [7]:

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

needed to guarantee repeatability

Self-describing means that from the configuration file alone we are able to derive the correct internal representation. [14]

Round-tripping means that if a file is serialized and then parsed again, we end up with an identical internal representation. [14]

Examples

XML has neither of the last two properties Siméon and Wadler [14]:

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

Hummer et al. [7] 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

Checking Configurations (Recapitulation)

Task

Which properties of configuration settings can be checked?

- structure
- values (data types)
- constraints
- semantic checks (e.g., IP, folder)
- domain-specific checks (e.g., databases)
- requirements (suitable configurations)
- context (context-aware configurations)

Checking Specifications (Recapitulation)

Task

What are the goals of checking SpecElektra are:

- Defaults must be present for safe lookups. This goal also implies that there must be at least one valid configuration setting.
- Types of default values must be compatible with the types of the keys.
- Every contextual interpretation of a key must yield a compatible type.
- Links must not refer to each other in cycles.
- Every link and the pointee must have compatible types.

Example (Recapitulation)

```
1 [sw/org/abc/has_true_arg]
2   type:=boolean
3   default:=0
4   override/#0:=/sw/org/abc/arg0
5   override/#1:=/sw/org/abc/arg1
```

Logfile Extensions (Recapitulation)

```
1 [slapd/logfile]
2   check/path:=file
3   check/validation:=^/var/log/
4   check/validation/message:=Policy violation:
5     log files must be below /var/log
```

CM Languages (Recapitulation)

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.
- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

  By functional languages and file system (layouts).
- Which notations for CM exist?

  Text, Graphical (UML), Semi-structured, Key-value, Structured

Popular CMs today (Recapitulation)

- CFengine
- LCFG
- Config Mgmt
- Quattor
- Puppet
- Chef
- Ansible (Talk next week)
- SaltStack (Talk today)
- Rudder
- Spacewalk

Elektra (Recapitulation)

Task

What is Elektra?

- is not only a key database but a specification language to describe a key database
- plugins implement the specification (could be distributed but focus is configuration files)
- is library based (no single point of failure, no distributed coordination needed)
- supports transactions (persisting whole KeySets at once)
- supports integration of existing configuration settings

Error Messages

Motivation (Recapitulation)

Error messages are extremely important as they are the main communication channel to system administrators.

```
1 [a]
2   check/type := long
3 [b]
4   check/type := long
5 [c]
6   check/range := 0-10
7   assign/math := ../a+../b
```

Task

Where should the error message point to if we change b to 10 (a is unchanged 1)?

Considerations (Recapitulation)

Task

What needs to be considered when designing error messages?

- Generic vs. specific plugins
- Precisely locate the cause (and do not report aftereffects)
- Give context
- Personification [9]

Further Considerations

- configuration design first: avoid errors if possible
- precision and recall[1] [16]
- error messages should not leak internals [4]
- "edit here mentality": do not point to correct statements [10]
- do not propose solutions [10]
- reduce vocabulary [10]
- tension between providing enough information and not overwhelming the user [16]
- colors might help [16]

[1] terms from classification, it is the numerical counterpart of soundness and completeness

Error Messages for Misconfiguration [18]

- error messages are often the sole data source
- tool uses misconfiguration injection and checks if error message point to the correct setting
- tool requires system tests
- they considered error message as okay if key *or* value is present

Implication

Missing error message means the configuration specification is not complete.

Context for error messages

Error messages should contain:

- pin-point key (which also pin-points to the specification)
- repeat relevant parts of values and the specification
- show mountpoint (to make relative keys unique)
- show file name and line number
- ? show module and source code lines (for bugs)

Precise Location (Recapitulation)

```
1 a=5   ; unmodified
2 b=10  ; modification bit in metadata
3       ; is only set here
4 c=15  ; unmodified by user but changed
5       ; later by assign/math
```

Example Error Messages (Recapitulation)

```
Sorry, I was unable to change the configuration settings!
Description: I tried to set a value outside the range!
Reason: I tried to modify b to be 10 but this caused c to
        be outside of the allowed range (0-10).
Module: range
At: sourcefile.c:1234
Mountpoint: /test
Configfile: /etc/testfile.conf
```

User View

System Administrator Research

- system administrators: the unsung heroes!
- interest of understanding administrators emerged around 2002 [1].
- field study also done in industry [3].
- Typical methods are surveys, diary studies, interviews and observations (ethnographic field study).
- Barrett [2] tried to initiate a workshop at CHI 2003 to draw the attention of the HCI community towards system administration.
- The workshop was already dropped in the next year.
- The tenor is that “tools ... are not well aligned” [5].
- Research mainly looks at pre-CM. Manual administration is still standard (Source: e.g., Luke Kanies).

Tasks

What do system administrators do?

- keep our infrastructure running
- coordinate
- backup
- hardware
- inventory
- install applications
- security
- configure applications
- troubleshoot

7 people, 1 command-line [3]

- system administrator misunderstood problem (had a wrong assumption)
- 7 people sought attention and trust, competing to tell the admin what to do
- due to wrong assumption the admin communicated to everyone, people could not help
- there were several instances in which the admin ignored or misinterpreted evidence of the real problem
- eventually someone else solved the problem: admin confused “from”/“to” port in the settings and firewall blocked requests

other cases [3]

- lost semicolon: execution of script failed due to missing semicolon, then they tried to delete a non-existent table.
- crontab: onltape/ofltape confused because of discussion about offline backup (although an online backup should be performed).
- crit sit: many system administrators tried to write simple script, they competed against each other. The crit sit continued for two weeks.

Haber and Bailey [5]

Later Haber and Bailey [5] repeated a ethnographic field study. The stories are similar to Barrett et al. [3]. Their study was also conducted in the same company. They created personas:

- database administrator
- web administrator
- security administrator

Database Administrator [5]

- frequent contact via phone, e-mail and IM
- needs to work on weekends
- pair-programming for new tasks
- typical errors: stopping wrong database process

Web Administrator [5]

- crit sit
- deploying new Web applications
- about 20-400 steps to deploy an application
- moving from test to production done by hand

Security Administrator [5]

- gets emails on suspicious activities
- multi-user chat
- ad-hoc scripts

Haber and Bailey [5]

- "if data is lost...that is when you write your résumé."
- 90 % is spent with communicating with other admins
- 20 % of the time is spent in diversions [3]
- 20 % of the time people communicated about *how to communicate* [3]
- 6 % is gathering information and running commands
- quality control: monitoring found that non-functional service was down two days
- CLIs were generally preferred
- configuration and log files were scattered, poorly organized and often used inconsistent terminology

Findings [3]

- syntax checking is essential
- replicating actions (e.g., to production) is error-prone
- undo not available
- do not assume a complete mental model ("if understand the system is a prerequisite [...], we are lost")
- do not assume programming skills (only 35 % reported having a bachelor's degree)
- trust in CLI tools but little trust in GUIs (is the information up-to-date?)
- errors while executing scripts lead to inconsistent state, rerunning often does not work (if not idempotent)

Design Principles [5]

Many design principles for tools were given [5]:

- configuration and logs should be displayed in a uniform way
- APIs/plugins for tools should be provided
- errors in configuration need to be discovered quickly
- confusion of similar settings should be avoided: add links, explain interactions
- provide means of comparing configuration settings
- provide consistent profiles of information
- both transient and persistent settings should be visible
- when errors occur: always display which changes have been made (modern approach is idempotence)

Apply to CM

What can we learn from manual system administration?

- \+ intensive review process catches errors
- – collaboration ineffective
- – context/situational awareness is essential
- \+ precise editing of configuration files works well
- – no global optimizations
- \+ self-written tools are very efficient

Idea

Replicate parts that work well, automate error-prone parts.

Precise Editing

Partial modifications (precise editing) is natural for humans.
It ensures preservations of (potentially security-relevant!) defaults.
In CM, however, following methods are used:

- embed shell commands to do the work
- replace full content of configuration files
- replace full content of configuration files with templates
- line based manipulation (e.g., file_line): match line and replace it
- Augeas/XML: match a key with XPath and replace it
- Elektra: set the value of a key

Apply to CM

Elektra's goals are:

- it should be easy to develop new high-level tools
- precise editing: change the configuration value as specified

Administrators/Devs still need to:

- intensively review and improve the specifications
- test (and debug) configuration settings

Open topics (incomplete):

- global optimizations/self-healing
- safe migrations of settings and data
- collaboration
- management (including knowledge)

[1] Eric Arnold Anderson. *Researching system administration*. PhD thesis, University of California at Berkeley, 2002.

[2] Rob Barrett, Yen-Yang Michael Chen, and Paul P. Maglio. System administrators are users, too: Designing workspaces for managing internet-scale systems. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, pages 1068–1069, New York, NY, USA, 2003. ACM. ISBN 1-58113-637-4. doi: 10.1145/765891.766152. URL http://dx.doi.org/10.1145/765891.766152.

[3] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben M. Haber, Leila A. Takayama, and Madhu Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 388–395. ACM, 2004.

[4] P. J. Brown. Error messages: The neglected area of the man/machine interface. *Commun. ACM*, 26(4):246–249, April 1983. ISSN 0001-0782. doi: 10.1145/2163.358083. URL http://doi.acm.org/10.1145/2163.358083.

[5] Eben M. Haber and John Bailey. Design guidelines for system administration tools developed through ethnographic field studies. In *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology*, CHIMIT '07, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-635-6. doi: 10.1145/1234772.1234774. URL http://dx.doi.org/10.1145/1234772.1234774.

[6] 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.

[7] 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.

[8] 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.

[9] Michael J. Lee and Andrew J. Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 109–116, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0829-8. doi: 10.1145/2016911.2016934. URL http://dx.doi.org/10.1145/2016911.2016934.

[10] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. Mind your language: On novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 3–18, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0941-7. doi: 10.1145/2048237.2048241. URL http://doi.acm.org/10.1145/2048237.2048241.

[11] 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.

[12] Markus Raab. Unanticipated context awareness for software configuration access using the getenv API. In *Computer and Information Science*, pages 41–57. Springer International Publishing, Cham, 2016. ISBN 978-3-319-40171-3. doi: 10.1007/978-3-319-40171-3_4. URL http://dx.doi.org/10.1007/978-3-319-40171-3_4.

[13] 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.

[14] 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.

[15] Éric Tanter. Contextual values. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-270-2. doi: 10.1145/1408681.1408684. URL http://dx.doi.org/10.1145/1408681.1408684.

[16] John Wrenn and Shriram Krishnamurthi. Error messages are classifiers: A process to design and evaluate error messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2017, pages 134–147, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5530-8. doi: 10.1145/3133850.3133862. URL http://doi.acm.org/10.1145/3133850.3133862.

[17] 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.

[18] Sai Zhang and Michael D. Ernst. Proactive detection of inadequate diagnostic messages for software configuration errors. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 12–23, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3620-8. doi: 10.1145/2771783.2771817. URL http://dx.doi.org/10.1145/2771783.2771817.