Literate Programming in der Systemadministration

SWM 2015

Meik Teßmer

Universität Bielefeld, Fakultät für Wirtschaftswissenschaften, Bereich Computergestützte Methoden

20. März 2015

Personelle Rahmenbedingungen und ihre Auswirkungen

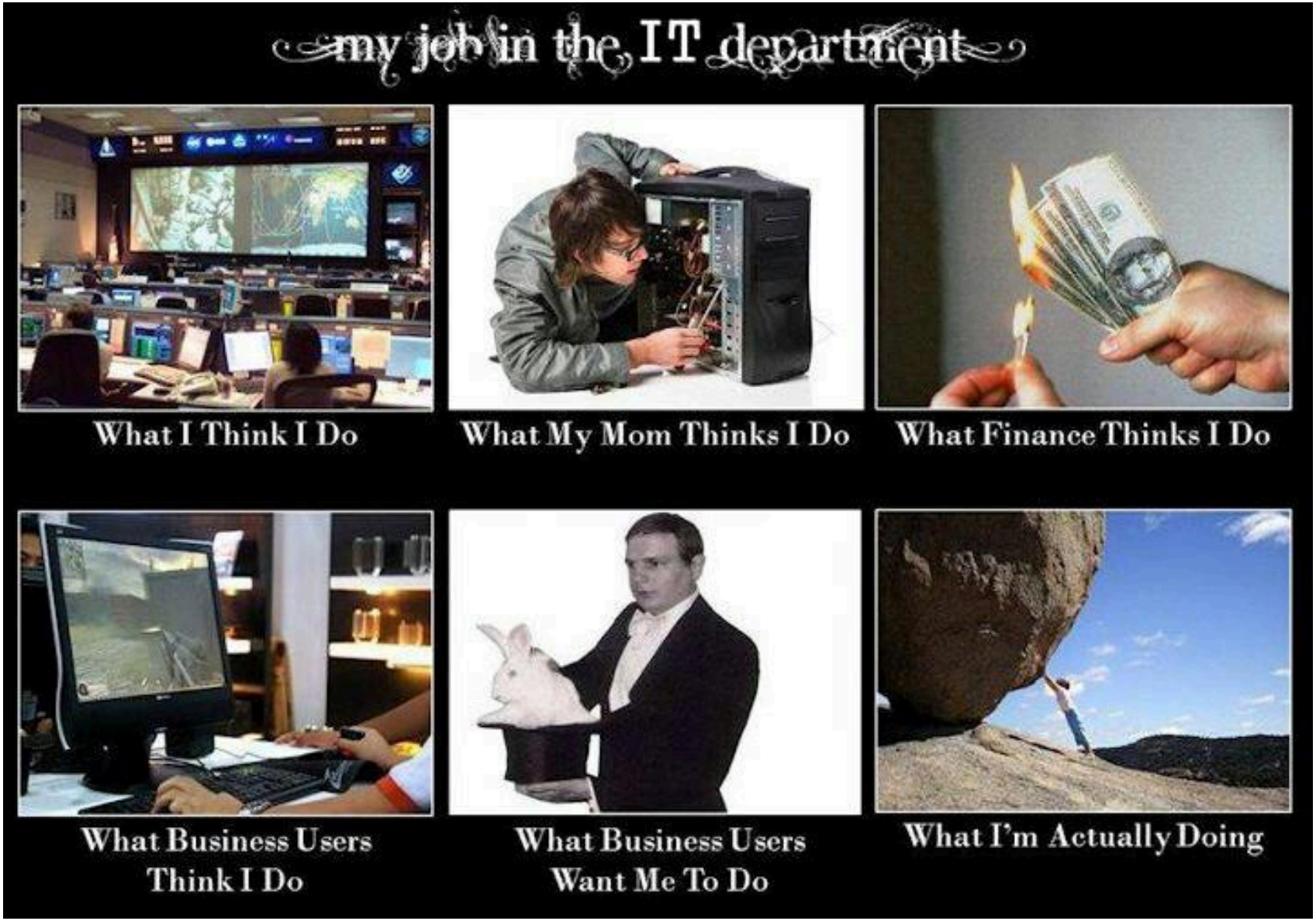
- 1 feste Stelle, 3 studentische Hilfskräfte (zu je 10 Stunden) für First-Level-Support und Client-Administration
- Beschäftigungsdauer der Mitarbeiter: max. 2 Jahre
- Einarbeitungszeit nur für Client-Administration: ~3 Monate
- Wissensstand der Hilfskräfte zu Beginn: von gering bis routiniert auf dem Desktop, kaum Netzwerkerfahrung

$\rightarrow$ Dokumentation ist ein wichtiges Werkzeug zur Kommunikation und Wissensvermittlung

Äußere Rahmenbedingungen

- Support für unterschiedliche Hardware-Plattformen, Betriebssysteme und Dienste erforderlich
- z.T. hohe Änderungsraten bei Software (Updates von Systemen und Anwendungssoftware)
- heterogene Nutzerschaft (von ahnungslos bis versiert, aber sehr häufig eigene Vorstellungen)
- mehr oder weniger autonom agierende Fachbereiche (z.T. Selbstadministration)
- nur bedingte Berücksichtigung erarbeiteter Hardware-/Softwarestandards und Richtlinien
- falsches Verständnis der Aufgaben der Systemadministration

...


my job in the IT department
What I Think I Do
What My Mom Thinks I Do
What Finance Thinks I Do
What Business Users Think I Do
What Business Users Want Me To Do
What I'm Actually Doing

Aufgaben der Systemadministration

- Fehlerbehebung bei Hard- und Software
- Beschaffung, Inventarisierung und Installation neuer Systeme
- Software-Deployment (Installation und Update)
- Entsorgung von Altsystemen samt Deinventarisierung
- Systemmanagement (Account- und Storageverwaltung)
- Betreuung der Computing-Server für rechenintensive Aufgaben
- Bereitstellung von Netzwerkspeicher
- Backup-Service für Server und Clients
- Verwaltung mehrere Webserver und Wiki-Systeme
- Betrieb einer Monitoring-Plattform (Hardware-/Dienstüberwachung)
- Bereitstellung von Versionierungsservern (Mercurial, Git)
- Beratung bei besonderen Anschaffungen/Projekten
- Bereitstellung spezieller Dienste (z.B. Evaluationsplattform, Informationssysteme für das Prüfungsamt)

Kriterien für eine sachgerechte Dokumentation?

Zweck von Dokumentation:

- Wiederfinden und
- Nutzbar machen

von Informationen. Wofür?

- Betrieb sicherstellen
- Routinearbeiten unterstützen
- Hilfe bei Auftragsarbeiten
- Betriebslenkungsaufgaben ermöglichen

Schwierigkeiten:

- Aufbau und Inhalt?
- Konsistenz von Dokumentation und tatsächlicher Systemlandschaft

Lösungsvorschlag der LISA SIG

Eigenschaften einer Systemdokumentation

Dokumentation sollte folgende vier Eigenschaften besitzen:

1. useful: enthält die benötigten Informationen
2. accessible: Zweck und Zielgruppe sind klar und eindeutig, Elemente der Dokumentation lassen sich schnell auffinden und verwenden
3. accurate: die vorhandenen Informationen sind vollständig, korrekt und aktuell
4. available: Dokumentation ist in jeder Situation verfügbar

Lösungsvorschlag der LISA SIG

Geeignete Formen

- digitales Ausgabe (online): mit üblichen Werkzeugen erreichbar, zeit- und ortsunabhängig verfügbar
- einfache Formate, keine Spezialsoftware für Bearbeitung notwendig, Versionsverwaltung
- Print-Ausgabe für Notfälle

Eigenschaften und Form sind damit klar, aber praktische Umsetzung?

- Eigenschaften 1-3: beziehen sich auf die inhaltliche Strukturierung
  - abhängig von den Systemen/Diensten
  - Aufteilung prozedurale und informationelle Dokumentation
- Eigenschaft 4: eine Quelle für alle Ausgaben
- Konsistenz? „Nutzbar“ machen?

Nutzbar machen der Dokumentation

- Verwendung von *formalen deklarativen Beschreibungen* administrativer Tätigkeiten
- Management-Tools wie Cfengine, Chef, Puppet, Salt definieren solche formalen Sprachen

```
/etc/vimrc:
  file.managed:
    - source: salt://vimrc
    - mode: 644
    - user: root
    - group: root
```

$\rightarrow$ eingebettet in die Dokumentation = „ausführbare“ Dokumentation

- weiterer Effekt: bessere Konsistenz von Dokumentation und Systemlandschaft

Literate Programming (LP)

- Dokumentation in der Softwareentwicklung: Entwurfsentscheidungen kommunizierbar machen, um Wartbarkeit und Portabilität zu gewährleisten
- Problem: es wird nur ungern und oft unzureichend dokumentiert
- Idee von Knuth (1984): Software als „Works of Literature“ $\rightarrow$ keine Trennung von Dokumentation und Quellcode
- Hauptziel des „Autors“: dem Leser erklären, *wie* das Programm funktioniert
- Quellcode kann extrahiert und zur Ausführung gebracht werden

Vorschlag: Konfigurationsbeschreibungen als Code einbetten

Integration von Dokumentation und Code durch LP

- Abfolge von *Documentation Chunks* und *Code Chunks*
- Code Chunks werden durch `<<Bezeichner>>=` eingeleitet, Documentation Chunks durch `@`
- Beispiel für LP mit `noweb`:

```
1  ...  Entsprechend wird die Authentifizierung
2  folgendermaßen umgesetzt:
3
4  <<zentrale Authentifizierung>>=
5  def authorize_(uid, two_factor_code):
6    ...
7
8  @
9  Sobald der Zugriff autorisiert ist, ...
```

- `tangle`: Extraktion des Quellcodes
- `weave`: Erzeugung der Dokumentation inkl. eingebetteter Code Chunks

Erfahrungen mit LP bei der Softwareentwicklung

- Entwicklung des Fakultätsinformationssystems für Prüfungsämter (Wiwi-Fakultät)
- Erweiterungen des Fakultätsinformationssystems von Studierenden
  $\rightarrow$ erfolgreiche Umsetzung der Erweiterungen und positive Erfahrungsberichte
- aber: Hinweise auf Defizite des ursprünglichen Ansatzes
  - LaTeX als Dokumentationsformat ist tippaufwändig
  - Entwicklung in nur einer Datei ist unübersichtlich und im Team schwierig (Versionsverwaltung)
  - kein adäquates Syntax Highlighting durch `noweb`

Übertragung auf die Systemadministration

Anpassungen des ursprünglichen LP-Ansatzes

- Sphinx als Dokumentationswerkzeug
  - reST als Markup
  - Umwandlung nach HTML, LaTeX, PDF, EPUB usw.
  - Unterstützung mehrerer Quelldateien
- `nw2rst`: Konverter für LP-Syntax nach reST-Markup

Beschreibung der Systeme

- Werkzeug: Salt
- geeignet für Ad-hoc-Administration und Konfigurationsmanagement
- deklarative Sprache: YAML-Format
- Salt States und Formulas: (parametrisierte) Beschreibung von Systemzuständen

Salt als Werkzeug zur Administration

```
# top.sls
base:
  'atlas':
    - core
    - ssh.server
    - users.comet

# core.sls
core:
  pkg:
    - installed
    - names:
      - bash
      - tmux
```

„Ausführung“:

```
salt 'atlas' state.highstate
```

Praktischer Einsatz

- Aufteilung der gesamten Dokumentation in 4 Teile:
  - Metaebene der Systemadministration
  - Konzeptteil (parametrisierte Salt Formulas)
  - Implementierung (konkrete Konfigurationen für die Formulas)
  - Knowledge Base
- Umwandlung für Weiterverarbeitung mit Sphinx:
  - `make html`: erzeugt HTML-Ausgabe für Online-Fassung
  - `make pdf`: erzeugt PDF-Ausgabe für Print-Fassung
- Extraktion der Salt-Teile: `make code`
- Anwenden auf die gesamte Systemlandschaft:

```
salt '*' state.highstate
```

Fazit

- Dokumentation kann gemäß der Vorschläge erstellt werden
  - useful, accessible: Build Books, Run Books
  - accurate: garantiert durch deklarative Beschreibungen
  - available: Export via Sphinx
- Bonus: nahezu vollständige Automatisierung durch Salt
- Konsistenz durch eingebettete Konfigurationsbeschreibungen gegeben
- nur *eine* Quelle, einfach zu bearbeiten
- Ansatz ist für Anfänger geeignet

Verfahren seit 3 Jahren im Einsatz; funktioniert

Schwierigkeiten:

- stark formalisierte Prozesse (Ticketsysteme)
  Umstellung des Ticket-/Wiki-Systems auf Fossil
- Systeme/Dienste, die nicht von Salt unterstützt werden
- Tools für die Bearbeitung (Outlining, Graphgenerierung etc.)

Ende

Fragen?