TaskJuggler 3.x - Projektmanagement für Linuxanwender

Projektmanagement ist mehr als Gantt-Diagramm-Zeichnen

Chris Schlaeger

chris@linux.com

Chemitzer Linux-Tage 2012

TaskJuggler und Linux

- TaskJuggler wurde entwickelt um die Entwicklungsabteilung einer Linux-Distribution zu leiten.
- Es wird immer noch von Distributionen wie Fedora verwendet.
- Es verwendet eine einfache Projektbeschreibungssprache.
- TaskJuggler funktioniert ähnlich wie ein Compiler oder LaTeX.
- Es besteht aus wenigen Programmen, welche die wesentlichen Funktionen liefern und mit anderen Programmen zusammenarbeiten.

Die Ruby Laufzeitumgebung

- TaskJuggler 3.x ist ein Ruby Programm
- Die Grundfunktionen sind noch kompatible zu Ruby 1.8.7.
- Die Server- und Automatisierungsfunktionen erfordern Ruby mindestends 1.9.3.
- Es wird generell Ruby 1.9.3 empfohlen da es etwa 3x schneller ist als 1.8.7.

Gem Installation

- Alle Linux Distributionen enthalten Ruby.
- Viele Distributionen enthalten leider immer noch das veraltete Ruby 1.8.
- RubyGem ist der Paketmanager für Ruby
- Ruby Gems sind vom Prozessor- und vom Betriebssystem unabhängig.
- Die Installation ist im Normalfall sehr einfach. Ein Kommando lädt das Gem Paket und alle Abhängigkeiten herunter und installiert diese.

```
gem install taskjuggler
```

Ruby 1.9 Installieren

- Die aktuelleste Version gibt es auf `ruby-lang.org`

  `http://www.ruby-.ang.org/en/downloads/`
- Auspacken, konfigurieren und installieren

```
tar -Zxvf ruby-X.X.X-*.tar.gz
cd ruby-X.X.X-*
./configure --program-suffix=19
make
sudo make install
ln -s /usr/local/bin/ruby19 ${HOME}/bin/ruby
```

- `${HOME}/bin` muss im Pfad enthalten sein
- Jede Ruby-Hauptversion hat eine eigene Gem Sammlung

Die Online-Hilfe: `tj3man`

- Hilfe zu einem TaskJuggler Schlüsselwort

  `tj3man <keyword>`
- Mit der `-html` Option wird es im Browser dargestellt

  `tj3man --html <keyword>`
- Hilfe zur Hilfe

  `tj3man --help`

Das Hauptprogramm: `tj3`

- Ein Projekt berechnen und Berichte erstellen

  ```
  tj3 yourproject.tjp
  ```

- Projekte können aus mehreren Dateien bestehen

  ```
  tj3 yourproject.tjp reports.tji
  ```

Client und Server: `tj3client` und `tj3d`

- Den Server starten

  `tj3d -w`
- Den Serverstatus abfragen

  `tj3client status`
- Ein Projekt (erneut) laden

  `tj3client add yourproject.tjp`
- Den Server beenden

  `tj3client terminate`

Die Projektbeschreibung

- Alle Projektdaten werden in einer oder mehreren Textdatein abgelegt.
- Information werden als *Properties* und zugehörigen *Attributes* strukturiert.
- Eine *Property* hat immer folgende Stuktur

```
PROPERTY [ID] "NAME" [ { ATTRIBUTES } ]
```

- Die in eckigen Klammern eingeschlossenen Elemente sind optional.

Die Properties

Ein Projekt besteht aus folgenden *Properties*

- `project`: Der Projektkopf
- `accounts`: Konten zur Kostenplanung
- `resources`: Mitarbeiter und Arbeitsmittel
- `tasks`: Die Arbeitsschritte
- `reports`: Die Berichte

Alle *Properties* haben *Attributes*. Z. B. die *Property* `task` hat u. a. die *Attributes* `start` und `duration`.

Der Projektkopf

- Wie heisst das Projekt?
- Wann geht es los?
- Wie lange dauert es vorraussichtlich?

```
1 project "Beispiel"  2012-03-01 +4m
```

Landesspezifische Anpassungen

- Lokalisierung der Zeitzone und des Datumsformats
- Anpassung der Zahlendarstellung
- Lokalisierung der Darstellung von Geldbeträgen

```
1 project "Beispiel"  2012-03-01 +4m {
2   timezone "Europe/Berlin"
3   timeformat "%d.%m.%Y"
4   numberformat "-" "" "" "," 1
5   currencyformat "-" "" "" "," 0
6   currency "EUR"
7 }
```

Mehrere Szenarien

- Ein Szenario beschreibt eine Variante des Projektverlaufs.
- Es können beliebig viele Szenarien analysiert werden.
- Szenarien können in einer Baumstruktur von einander abgeleitet werden.
- Abgeleitete Szenarien erben alle *Attributes*, können sie aber mit eigenen Werten überschreiben.

```
1 project "Beispiel"  2012-03-01 +4m {
2   scenario plan "Plan" {
3     scenario real "Realität"
4   }
5   now 2012-03-17
6 }
```

Erweiterung des Datenmodells

- *Properties* haben eine Reihe von *Attributes*.
- Sie können aber mit eigenen *Attributes* erweitert werden.

```
1 project example "Beispiel"  2012-03-01 +4m {
2   extend resource {
3     text Phone "Phone"
4   }
5   extend task {
6     reference Wiki "Wiki"
7   }
8 }
```

Resourcen deklarieren

- Einen Mitarbeiter anlegen

```
1 resource karl "Karl Mustermann"
```

- Ein Team anlegen

```
1 resource ateam "Das A-Team" {
2   rate 330.0
3   resource karl "Karl Mustermann"
4   resource erika "Erika Musterfrau"
5 }
```

- Das `rate` *Attribute* wird von den verschachtelten Resourcen übernommen.

Arbeitsmittel deklarieren

- Der Gerät wird nicht müde, leistet aber auch keine Arbeit.

```
1 resource geraet "Der Gerät" {
2   efficiency 0.0
3   rate 500.0
4 }
```

Die Projektgliederung

```
1 task "Phase 1" {
2   task "Schritt 1"
3   task "Schritt 2"
4 }
5 task "Phase 2" {
6   task "Schritt 1"
7   task "Schritt 2"
8 }
```

Abhängigkeiten spezifizieren

```
1  task p1 "Phase 1" {
2    task s1 "Schritt 1"
3    task "Schritt 2" {
4      depends !s1 # Relative ID
5    }
6  }
7  task p2 "Phase 2" {
8    task s1 "Schritt 1" {
9      depends p1.s1 # Absolute ID
10   }
11   task "Schritt 2"
12 }
```

Aufwände und Laufzeiten

```
1  task p1 "Phase 1" {
2    task s1 "Schritt 1" {
3      duration 2d
4    task "Schritt 2" {
5      length 10d
6    }
7    task "Schritt 3" {
8      effort 5d
9      allocate karl
10   }
11 }
```

Listen und Berichte

- Es werden diverse Listenarten unterstützt
  - Arbeitslisten
  - Mitarbeiter- und Gerätelisten
  - Kalender
  - Zeiterfassungsvorlagen
  - Statusmeldungsvorlagen
- Listen können in unterschiedlichen Formaten erstellt werden
  - HTML
  - CSV
  - TaskJuggler Syntax
  - iCal

Arbeitslisten

```
1 taskreport "Arbeitsliste" {
2   formats html
3   hidetask ~isleaf()
4   sorttasks plan.end.up
5 }
```

Mitarbeiterlisten

```
1 resourcereport "Mitarbeiter" {
2   formats html
3   sortasks plan.id.up
4   columns no, name, email
5 }
```

Arbeitslisten mit Mitarbeitern

```
1 taskreport "Arbeitsliste" {
2   formats html
3   hidetask ~isleaf()
4   sortasks plan.effort.up
5   hideresource 0
6   columns no, name, weekly
7 }
```

Textberichte

- Textberichte bestehen aus bis zu 5 frei definierbaren Textblöcken.
- `header`, `left`, `center`, `right`, `bottom`

```
1 textreport "Textbericht" {
2   formats html
3   left "Linker Rand"
4   center "Mitte"
5   right "Rechter Rand"
6 }
```

RichText Markup

- Viele Textattribute werden als *RichText* interpretiert.
- TaskJuggler verwendet eine Teilmenge des MediaWiki Markups.
- Es gibt allerdings einige Erweiterungen:
  - Textfarben

    ```
    <fcol:green>Grün</fcol>
    ```
  - Navigationsleiste

    ```
    <[navigator id='mein_menue']>
    ```
  - Wertabfrage

    ```
    <-query ...->
    ```
  - Eingebetteter Bericht

    ```
    <[report id='mein_report']>
    ```

Zusammengesetzte Berichte

```
1  taskreport r1 "" {
2    columns name, chart
3  }
4  resourcereport r2 "" {
5    columns name, phone
6  }
7  textreport "Textbericht" {
8    formats html
9    left "<[report id='r1']>"
10   right "<[report id='r2']>"
11 }
```

Navigationsleisten für HTML Berichte

```
1  navigator menu
2  textreport "" {
3    header "<[navigator id='menu']>"
4    formats html
5    taskreport "Aufgaben" {
6      columns name, start, end, chart
7      hideresource 0
8    }
9    resourcereport "Mitarbeiter" {
10     columns name, email
11   }
12   purge formats
13 }
```

Der Projektmanagementzyklus

Nachdem das Projekt gestartet ist, sollten folgende Schritte regelmässig durchgeführt werden:

- Geleistete Arbeit für alle laufenden Aufgaben erfragen
- Verbleibenden Aufwand oder Laufzeit erfragen
- Plan entsprechend aktualisieren
- Vergangenheit einfrieren

`timesheets` und `statussheets`

- Abfragen der geleisteten und verbleibenden Aufwände und Laufzeiten kann durch `timesheetreports` stark automatisiert werden.
- Hierarchische Statusberichte können mit Hilfe von `statussheetreports` automatisiert werden.
- Dieses Thema ist zu komplex für diesen Vortrag. Es wird im Benutzerhandbuch aber ausführlich beschrieben.

Die Vergangenheit einfrieren

- Nach dem Aktualisieren des Plans sollte in regelmässigen Abständen die Vergangenheit eingefroren werden.

```
tj3 --freeze yourproject.tjp
```

- Es werden 2 Dateien erzeugt oder aktualisiert
  - `yourproject-header.tjp`
  - `yourproject-bookings.tjp`
- Diese Dateien müssen durch `include` in das Projekt eingebunden werden.

Kennwerte mit `tracereport` beobachten

- Mit `tracereports` können wichtige Kennwerte über die Projektlaufzeit verfolgt werden.
- Die Daten werden in eine CSV Datei geschrieben, die bei nur bei expliziter Aktualisierung ergänzt wird.
- `tj3 -add-trace yourproject.tjp`
- Eignet sich hervorragend zur Erstellung aktueller Burndown Charts.
- `tracereports` sind die größte Neuerung der nächsten Version (3.2).

Weiterführende Informationen

- TaskJuggler im Web: `www.taskjuggler.org`
- Online Handbuch:
  `www.taskjuggler.org/tj3/manual/index.html`

- Fragen?