Designdokument für die TurboGears QA-Initiative

Inhaltsverzeichnis

Projektbeschreibung

Ziel des Projektes ist es, zum einen eine Systemumgebung zu erstellen, die den automatisierten Test und die Erstellung neuer TurboGears[1]-Versionen übernimmt und außerdem die hierfür benötigten binary egg[2]s erstellt und den Usern zur Verfügung stellt und zum anderen die Einführung von SVNChecker[3] im Projekt und dessen grundlegende Konfiguration.

Test- und Build-Umgebung

Die Systemumgebung wird mit Hilfe von BuildBot[4] erstellt. Ein zentraler Buildmaster[5] wird konfiguriert, der die Buildslaves steuert und ihnen Buildtasks zuweist. Für jede unterstützte

---

[1] turbogears.org

[2] peak.telecommunity.com/DevCenter/setuptools

[3] svnchecker.tigirs.org

[4] buildbot.net

[5]

Betriebssystemkonfiguration werden drei Buildslaves (jeweils einer für Python 2.3, 2.4 und 2.5) konfiguriert.

Unterstützte Betriebssysteme/Architekturen:

- Buildmaster (24h online)
- Buildslaves (im Optimum 24h online, theoretisch aber nicht notwendig)
    - WinXP
    - Win2k(3)
    - Ubuntu (x86, x64)
    - Solaris (x86)
    - MacOS

Beteiligte Personen:

Buildmaster-Admin:

Kümmert sich um den Buildmaster, fügt dem Buildmater neue Szenarien hinzu und passt die Buid an gegebene Veränderungen an.

Buildslave-Admin

Betreut den Buildslave; erstellt sicher, dass der Buildslave alle vom Buildmaster geforderten Abhängigkeiten bereitstellt und der Buildslave ordnungsgemäß funtioniert.

Report-Verwalter

Reagiert auf die vom Buildmaster erstellten Tickets und beantwortet Fragen der Community zum Ticket.

Entwickler

Stößt durch Check-Ins die Builds an und wird über eine Mailingliste über die Ergebnisse der Builds informiert.

Verwendete Komponenten

Software

BuildBot

BuildBot dient der Erstellung und Verwaltung der Bot-Farm.

VirtualEnv

VirtualEnv[6] dient der Erzeugung unabhängiger Python-Installation. Diese werden benötigt, um Wechselwirkung mit anderen installierten Packages zu verhindern und so die Testergebnisse verlässlich zu halten.

---

[6] http://pypi.python.org/pypi/virtualenv

Nose

Nose[7] ist ein unittest-Framework für Python und wird im Projekt zur Sicherung der Codeintegrität genutzt. Es wird benötigt, um die Funktionstüchtigkeit des Sourcecodes sicherzustellen.

Buildmaster

Der Buildmaster stößt die Buildvorgänge an, die dann durch die Slaves aufgeführt werden. Er sammelt außerdem die Resultate und gibt sie weiter. Der Buildmaster wird als Teil des Projektes erstellt und als Image oder Archiv zur Verfügung gestellt. Er verfügt über ein Webinterface, welches den Status der Builds mitteilt und über das gegebenenfalls manuell Builds angestoßen werden können.

Buildslaves

Auf den Buildslaves werden die Builds ausgeführt. Der Slave lädt sich die benötigten Dateien (sowohl Source als im Rahmen des Projektes erstellte Skripte) selbstständig aus dem SVN-Repo herunter. Die Ergebnisse des Builds werden automatisch verteilt (Resultate an den Buildmaster bzw. das Wiki[8] und Eggs an den EggBasket[9]).

Buildtasks

Erstellung der binary eggs für alle Abhängigkeiten

Außer den eggs für TG selber, sollen auch eggs für die Abhängigkeiten erstellt werden. Die Erstellung der eggs wird nach folgendem Schema ablaufen:

1. Checkout des Source bzw. Sourcedownload per easy_install
2. Erstellung der eggs
3. Ausführen der Coretests (falls vorhanden)
4. Upload auf den TG-Server

Binary eggs sollen für folgende Projekte erstellt werden:

- Cheetah
- Webhelper
- Pylons
- RuleDispatch
- Python 2.4
- simplejson
- pysqllite (Py2.4)
- PyProtocols
- cElementTree

---

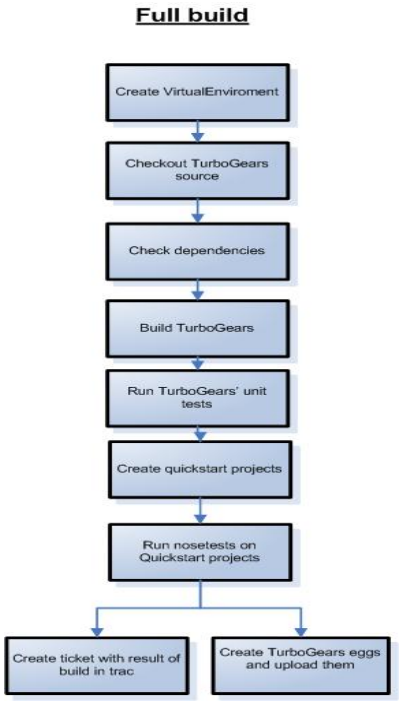
[7] http://www.somethingaboutorange.com/mrl/projects/nose/

[8] http://docs.turbogears.org/

[9] www.chrisarndt.de/projects/eggbasket/

Erstellung der binary eggs von TG

Es sollen über die BuilBots binary eggs für TG 1.0, 1.1 und 2.0 erstellt werden. Diese werden zum Download dann auf den TG-Server übertragen.



Die Erstellung der eggs wird nach dem nebenstehenden Schema erfolgen.

1. Erstellung eines VirtualEnvironment zur Abtrennung der TG-Installation von anderen Packages.
2. Auschecken des TurboGears-Source aus dem Repo[10].
3. Installation externer Abhängigkeiten, wie z.B. SQLObject oder Genshi
4. Installation von TurboGears
5. Ausführen der unittests von TG (sollten sie fehlschlagen, wird der Buildvorgang abgebrochen)
6. Erstellung aller möglichen quickstart-Projekte
7. Ausführen der unittest des quickstart-Projekte
8. Erstellen eines Tickets im trac, das die Ergebnisse des Builds enthält (nur bei Fehlschlag?)
9. Erstellung des TG-eggs und Upload auf den TurboGears-Server (s. Upload auf den TG-Server)

Upload auf den TG-Server

Der Upload der erstellten eggs erfolgt in die dort laufende EggBasket-Instanz.

SVNChecker

SVNChecker ist ein Framework, das eine einfache Erstellung von SVN-Hook-Scripts ermöglicht. Es bietet eine Reihe von vordefinierten Klassen, die die Anpassung des SVN-Servers an die eigenen Bedürfnisse vereinfacht. Es ermöglicht die einfache Anbindung eines Bugtrackers an den Server, so dass z.B. geprüft wird, ob für einen Eincheckversuch auch ein Ticket vorliegt. Wenn ein kritischer Check fehlschlägt, wird der Check-In zurückgewiesen.

Zu implementierende Funktionalitäten:

Pylint

Einzucheckender Code wird mittels Pylint auf syntaktische Korrektheit geprüft. Hierbei soll auf Erfüllung der Codingstandards geprüft werden.

---

10 http://svn.turbogears.org/

XMLValidator

Einzucheckende XML-Templates werden mittels XMLValidator geprüft.

Anbindung an trac

(wird ergänzt, wenn trac-Funktionalität bekannt)

Referenzen

Mein Blog: stevenmohr.wordpress.com

Projektplan: TurboGears QA-Initiative