Dokumentation Fabian Eichenberger

|  |  |
| --- | --- |
| Teammitglieder | Toni Kostal  Robin Steiger  Henri Schmekies  Fabian Eichenberger |
| Projektleiter | Henri Schmekies |
| Kunde | M. Schneider |
| Beschreibung Projekt | Ein Game mit Löve2D erstellen und sich damit in die Programmiersprache von Löve2D (Lua) einarbeiten. |
| Datum | 29. April 2015 |
| [🡨 Zurück zur Übersicht](../projektuebersicht.docx) |  |

Inhaltsverzeichnis

[Metainformationen 3](#_Toc412637588)

Metainformationen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datum | Status | Version | Autor | Bemerkung |
| 25.02.2015 | In Bearbeitung | 1.0 | Henri Schmekies | Start Projekt |

|  |  |
| --- | --- |
| Aktueller Status: | In Bearbeitung |

Package Management in Lua

In jeder Programmiersprache braucht es eine Software für die Verwaltung von Code Stücken. Früher hat man mit ZIP umhergeschoben, aber heutzutage ist es wichtig ein Package Management zu besitzen für grössere Applikationen. Viele Hilfsmodule sind Open Source und es ist wichtig, diese einfach konsumieren zu können.

Ich habe schon Erfahrungen mit einem Package Manager gemacht im JavaScript Ökosystem. Im modernen JavaScript Version 6 kann man die Sprache auch im Server Bereich einsetzen. Server-Side JavaScript hat in 2011 begonnen, und hat in den folgenden Jahren sehr popularisiert. Es war der Beginn, die Skriptsprache JavaScript zu modernisieren. Damit kam auch [NPM](https://www.npmjs.com/) der für *Node Package Manager* steht. Dort konnte ich sehr viel über Versionierung lernen.

Vorteile

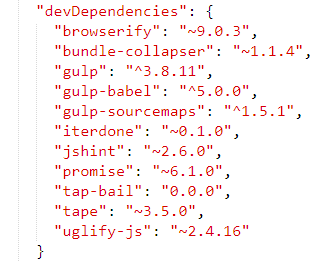
1. Kleine modulare Teile können von Drittentwickler gebraucht werden. Diese sind meist gut getestet und machen nur eine Sache gut (Unix Philosophie)
2. Versionierung vereinfacht die Wartung von den etlichen Modulen. Sogenannte Semver Ranges versichern, dass man Updates bekommt, diese aber den existieren Code nicht beschädigen (Non-Breaking Changes)
3. Durch eine Metadatenfile kann der Package Manager die „Dependencies“ einfach auflösen und installieren. Das Ziel ist es, einer Person den Code mit dem Metadatenfile zu schicken und er kann die Dependencies problemlos installieren.  
   

Abbildung 1: Beispiel Metadatenfile

Nachteile

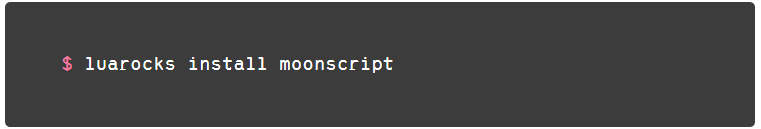
Einen Nachteil den ich manchmal immer wieder sehe, sind die sogenannten Auflösungsfehler. Das passiert wenn zwei Module auf das *gleiche* Package verweisen aber mit einer anderen Versionsnummer. Man kann unterscheiden zwischen „peerDependencies“ und „Dependencies“. „peerDependencies“ gibt *nicht* die genaue Abhängigkeit ab sondern bestimmt einfach, dass der Benutzer des Moduls diese Version *mitinstallieren* soll. Dies hat den Vorteil dass der Code nicht komplett dupliziert wird aber den Nachteil, nur eine Version benutzen zu können im ganzen Projekt.

LuaRocks

In Lua gibt es auch einen Package Manager namens [LuaRocks](https://luarocks.org/). Das war mein Ziel eine etwas andere Perspektive zu gewinnen und wie andere Programmiersprachen dieses Problem angehen.

Mit LuaRocks kann man das ganze Projekt auch ausführen und „builden“. Da Lua nicht auf dem Web aufbaut, hat jedes Modul seinen eigenen Kompilierungsschritt. Dies ist etwas aufwendig für Beginner und braucht eine gewisse Lernkurve zu überwinden.

Ich habe kleine Packages installiert und diese ausprobiert. Installation war ziemlich einfach: ein einfach *luarocks install* installiert das Modul:



Wichtig ist der optionale Parameter *–save* der das Modul in das Metadatenfile einfügt.

Glossar

|  |  |
| --- | --- |
| **Wort** | **Beschreibung** |
| Dependencies | Abhängigkeiten von Modulen |
| Metadatenfile | Ein File welches Informationen über die Software enthält |