

# **px: PEAX Command Line Client**

**Wirtschaftsprojekt, Herbstsemester 2019**

Patrick Bucher

3. Oktober 2019

## **Abstract**

Lorem ipsum dolor sit amet (Raymond, Eric S, 2004, p. 99), consetetur sadipscing elitr (Martin, Robert C., 2018, p. 101), sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet (Laboon, Bill, 2017, p. 99). Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua (Gancarz, Mike, 1995, p. 88). At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet (Donovan, Alan A. A. and Kernighan, Brian W., 2015, p. 23).

## Inhaltsverzeichnis

<b>1 Problemstellung</b>	<b>4</b>
1.1 Analyse des Projektauftrags . . . . .	4
1.1.1 Endpoints . . . . .	4
1.1.2 HTTP-Methoden . . . . .	4
1.1.3 HTTP Status-Codes . . . . .	4
1.1.4 Umgebungen . . . . .	6
1.1.5 Arten von Parametern . . . . .	6
1.1.6 Benutzer . . . . .	6
1.1.7 Betriebssysteme . . . . .	6
1.1.8 Shells . . . . .	6
<b>2 Stand der Praxis</b>	<b>7</b>
2.1 Ansprechen der PEAX API . . . . .	7
2.2 Kommandozeilenprogramme . . . . .	7
<b>3 Ideen und Konzepte</b>	<b>8</b>
<b>4 Methoden</b>	<b>9</b>
4.1 Teststrategie . . . . .	9
<b>5 Realisierung</b>	<b>10</b>
<b>6 Evaluation und Validierung</b>	<b>11</b>
<b>7 Ausblick</b>	<b>12</b>
<b>8 Anhang</b>	<b>13</b>
8.1 Technologie-Evaluation . . . . .	13
8.1.1 Programmiersprache . . . . .	13
8.1.2 Go . . . . .	14
8.2 Rust . . . . .	15
8.3 Libraries . . . . .	15
<b>Literatur</b>	<b>16</b>
<b>Abbildungsverzeichnis</b>	<b>17</b>

## *Inhaltsverzeichnis*

<b>Tabellenverzeichnis</b>	<b>18</b>
<b>Verzeichnis der Codebeispiele</b>	<b>19</b>

# 1 Problemstellung

## 1.1 Analyse des Projektauftrags

Die Umgebung der PEAX API einerseits

### 1.1.1 Endpoints

Eine RESTful-API besteht aus einer Reihe sogenannter *Endpoints*, d.h Pfade zu Ressourcen, die abgefragt und/oder manipuliert werden können.

### 1.1.2 HTTP-Methoden

Ein Endpoint kann über eine oder mehrere HTTP-Methoden angesprochen werden (Fielding & Reschke, 2014, Abschnitt 4.3). Im Kontext der PEAX API sind folgende Methoden relevant:

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- PATCH (Dusseault & Snell, 2010)

### 1.1.3 HTTP Status-Codes

Eine Antwort auf eine HTTP-Anfrage enthält jeweils einen Status-Code (Fielding & Reschke, 2014, Abschnitt 6). Bei der PEAX API werden u.a. folgende Status-Codes häufig verwendet:

- 200 OK: Die Anfrage hat funktioniert.
- 201 Created: Die Anfrage hat funktioniert, und dabei wurde eine neue Ressource erzeugt.

## 1 Problemstellung

- **204 No Content:** Die Anfrage konnte ausgeführt werden, liefert aber keinen Inhalt zurück (etwa in einer Suche mit einem Begriff, zu dem keine Ressource gefunden werden kann).
- **204 Partial Content:** Der zurückgelieferte Payload repräsentiert nur einen Teil der gefundenen Informationen. Wird etwa beim Paging eingesetzt.
- **400 Bad Request:** Die Anfrage wurde fehlerhaft gestellt (ungültige oder fehlende Feldwerte).
- **401 Unauthorized:** Der Benutzer ist nicht autorisiert, d.h. nicht eingeloggt im weitesten Sinne.
- **403 Forbidden:** Der Benutzer ist zwar eingeloggt, hat aber keine Berechtigung mit der gewählten Methode auf die jeweilige Resource zuzugreifen.
- **404 Not Found:** Die Resource wurde nicht gefunden; deutet auf eine fehlerhafte URL hin.
- **405 Method Not Allowed:** Die Resource unterstützt die gewählte Methode nicht.
- **415 Unsupported Media Type:** Das Format des mitgelieferten Payloads wird nicht unterstützt. In der PEAX API sind dies etwa Dokumentformate, die beim Hochladen nicht erlaubt sind (z.B. .exe-Dateien).
- **500 Internal Server Error:** Obwohl die Anfrage korrekt formuliert und angenommen worden ist, kam es bei der Verarbeitung derselben zu einem serverseitigem Fehler.<sup>1</sup>
- **380 Unknown:** Dieser Status ist nicht Teil der HTTP-Spezifikation, wird aber nach einem Login-Versuch verwendet, wenn eine Zwei-Faktor-

---

<sup>1</sup>In der PEAX API kommt es gelegentlich zu solchen Fehlern, die stattdessen mit dem Status **400 Bad Request** und einer aussagekräftigen Fehlermeldung beantwortet werden müssten. Wird z.B. bei der Einlieferung von Dokument-Metadaten eine syntaktisch fehlerhafte IBAN mitgegeben, tritt der Fehler erst bei der internen Verarbeitung, und nicht schon bei der Validierung der Anfrage auf. Hier besteht Handlungsbedarf aufseiten der Backend-Entwicklung.

## *1 Problemstellung*

Authentifizierung (SMS, One Time Password) verlangt wird, und ist somit für die vorliegende Arbeit von hoher Relevanz.

### **1.1.4 Umgebungen**

### **1.1.5 Arten von Parametern**

### **1.1.6 Benutzer**

### **1.1.7 Betriebssysteme**

### **1.1.8 Shells**

## **2 Stand der Praxis**

### **2.1 Ansprechen der PEAX API**

### **2.2 Kommandozeilenprogramme**

## **3 Ideen und Konzepte**



## 4 Methoden

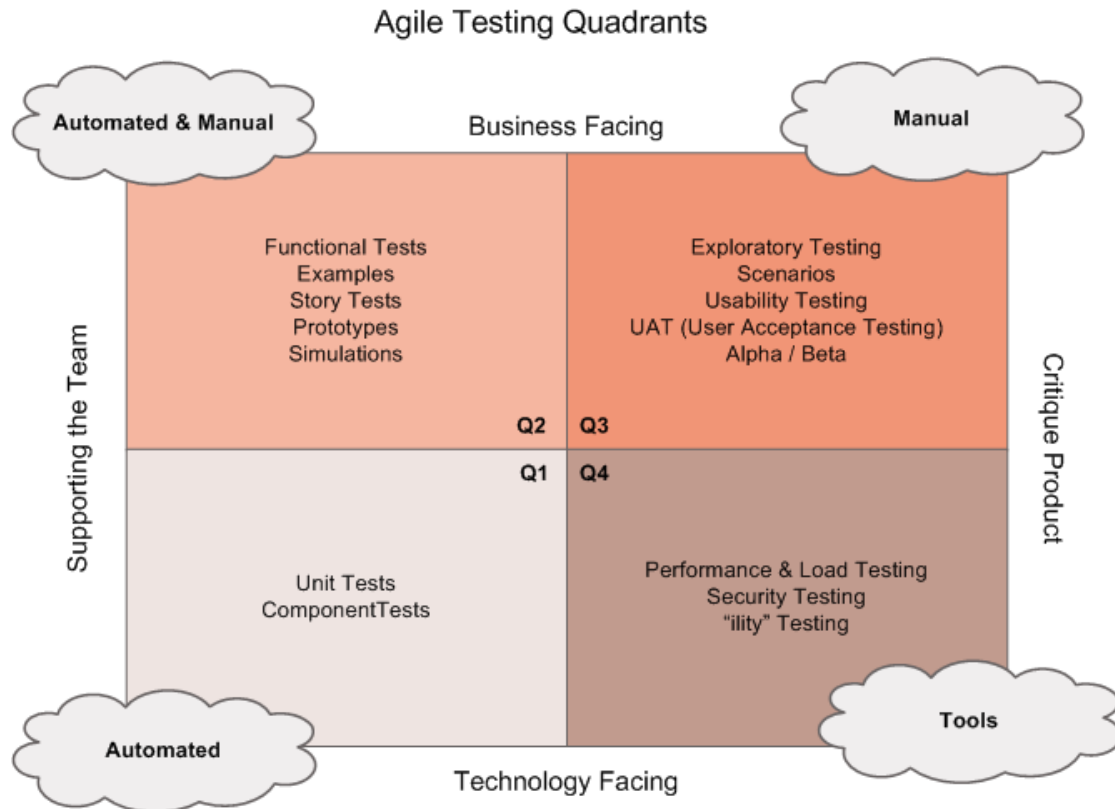


Abbildung 1: *Agile Testing Quadrants* nach Lisa Crispin (<https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>)

## 4 Methoden

### 4.1 Teststrategie

Wie im Vorsemester im Modul *Software Testing* eingeübt, sollen die *Agile Test Quadrants* (Abbildung 1, Seite 9) als Grundlage zur Erarbeitung einer Teststrategie dienen (Crispin, Lisa and Gregory, Janet, 2008, p. 242).

## **5 Realisierung**

## **6 Evaluation und Validierung**

## **7 Ausblick**

## 8 Anhang

### 8.1 Technologie-Evaluation

- Vorgabe: PEAX API (RESTful)

#### 8.1.1 Programmiersprache

Anforderungen:

**Installation** Die Software soll sich einfach installieren lassen.

**Umgebung** Es dürfen keine besonderen Anforderungen an die Umgebung gestellt werdenm auf der px läuft.

**Plattformen** Die Software soll auf allen gängigen Betriebssystemen (Windows, mac OS, Linux) lauffähig sein.

**Einheitlichkeit** Der Client soll überall die gleiche Befehlssyntax haben.

**Performance** Ein Command Line Client soll in Skripten verwendet werden können, wodurch das Programm sehr oft in kurzem Zeitraum aufgestartet werden muss.

Java erfordert die lokale Installation einer JRE in der richtigen Version. Ausserdem werden Wrapper-Skripts benötigt (`java -jar px.jar` ist nicht praktikabel).

Python, Ruby, Perl und andere Skriptsprachen benötigen ebenfalls einen vorinstallierten Interpreter in der richtigen Version.

Zwar gibt es mit Mono eine Variante von .Net, die überall lauffähig ist, hier werden aber wiederum eine Laufzeitumgebung bzw. vorinstallierte Libraries benötigt.

Am besten geeignet sind kompilierte Sprachen (C, C++, Go, Rust, Nim). Mit einer statischen Kompilierung lässt sich das ganze Programm in eine einzige Binärdatei überführen, welches denkbar einfach zu installieren ist (Kopieren nach einem der Verzeichnisse innerhalb von \$PATH).

Für JavaScript gibt es mit QuickJS seit kurzem die Möglichkeit, JavaScript zu Binärdateien zu kompilieren. Dies funktioniert aber nicht auf allen Plattformen, ausserdem ist QuickJS noch experimentell und noch nicht für den produktiven Einsatz geeignet.

In die engere Auswahl kommen Go und Rust, da der Autor dieser Arbeit mit diesen Programmiersprachen bereits Erfahrungen im Studium machen konnte. (Mit C++ noch keine Erfahrungen gemacht. Mit C Erfahrungen gemacht, wodurch es als sehr aufwändig erscheint, einen HTTP-Client zu schreiben.)

### 8.1.2 Go

- einfach zu lernen (wenige Keywords und Features), dafür keine Features wie Generics und map/filter/reduce
- gutes und übersichtliches Tooling
- Cross-Compilation ohne Zusatztools möglich
- Kompilierung zu statischen Binaries, die jedoch recht gross ausfallen (Prototyp: ca. 4 MB)
- Kompilierung extrem schnell
- umfassende und qualitativ hochwertige Standard Library, inkl. HTTP-Library
- persönlich bereits viel damit gearbeitet, positive Erfahrungen damit gemacht
- Error Handling aufwändig, führt aber zu sehr solidem, wenn auch repetitivem Code
- vergleichbare Software (oc: OpenShift Command Line Client, docker: Docker Command Line Client) ist ebenfalls in Go geschrieben und bei uns täglich erfolgreich im Einsatz
- fügt sich sehr gut in UNIX-Philosophie ein (Tooling, Libraries)
- Einfaches Interface für nebenläufige Programmierung (Goroutines und Channels)
- Performance im Bereich von Java, jedoch tieferer Memory-Footprint

## 8.2 Rust

- viele Features, dafür aber schwer zu lernen (Lifetimes, Borrowing, siehe PCP)
- gutes und übersichtliches Tooling
- Kompilierung in statische Binaries, die relativ schlank ausfallen
- Kompilierung eher langsam
- Cross-Compilation benötigt Zusatztools
- wenig Erfahrung damit gesammelt
- Standard Library bewusst schlank gehalten, dafür viele externe Libraries benötigt
- extrem ausdrucksstarke Sprache mit starkem Typsystem
- diverse Rust-Utils erfolgreich persönlich im Einsatz (rg, bat, hexyl, bat-top)
- erstklassige Performance (im Bereich von C/C++)

## 8.3 Libraries

## Literatur

- Crispin, Lisa and Gregory, Janet. (2008). *Agile Testing*. Addison-Wesley.
- Donovan, Alan A. A. and Kernighan, Brian W. (2015). *The Go Programming Language*. Addison-Wesley.
- Dusseault, L. & Snell, J. (2010, March). *Patch method for http* (RFC Nr. 5789). RFC Editor. Internet Requests for Comments. Zugriff auf <http://www.rfc-editor.org/rfc/rfc5789.txt> (<http://www.rfc-editor.org/rfc/rfc5789.txt>)
- Fielding, R. & Reschke, J. (2014, June). *Hypertext transfer protocol (http/1.1): Semantics and content* (RFC Nr. 7231). RFC Editor. Internet Requests for Comments. Zugriff auf <http://www.rfc-editor.org/rfc/rfc7231.txt> (<http://www.rfc-editor.org/rfc/rfc7231.txt>)
- Gancarz, Mike. (1995). *The UNIX Philosophy*. Digital Press.
- Laboon, Bill. (2017). *A Friendly Introduction to Software Testing*.
- Martin, Robert C. (2018). *Clean Architecture*. Prentice Hall.
- Raymond, Eric S. (2004). *The Art of UNIX Programming*. Addison-Wesley.



## **Abbildungsverzeichnis**

- 1 *Agile Testing Quadrants* nach Lisa Crispin (<https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>) . . . 9

## **Tabellenverzeichnis**

## **Verzeichnis der Codebeispiele**