

AUTOMATISCHE TESTFALLGENERIERUNG

ODER

WIE DIE EVOLUTION
DEN AFFEN ZUM TESTER MACHT

Dr. Jeremias Rößler

Automatisch vs. Manuell



Weniger Risiko.



Weniger Stress.



Weniger Kosten.

Vollautomatisch vs. Automatisch



Keine manuelle Erstellung.

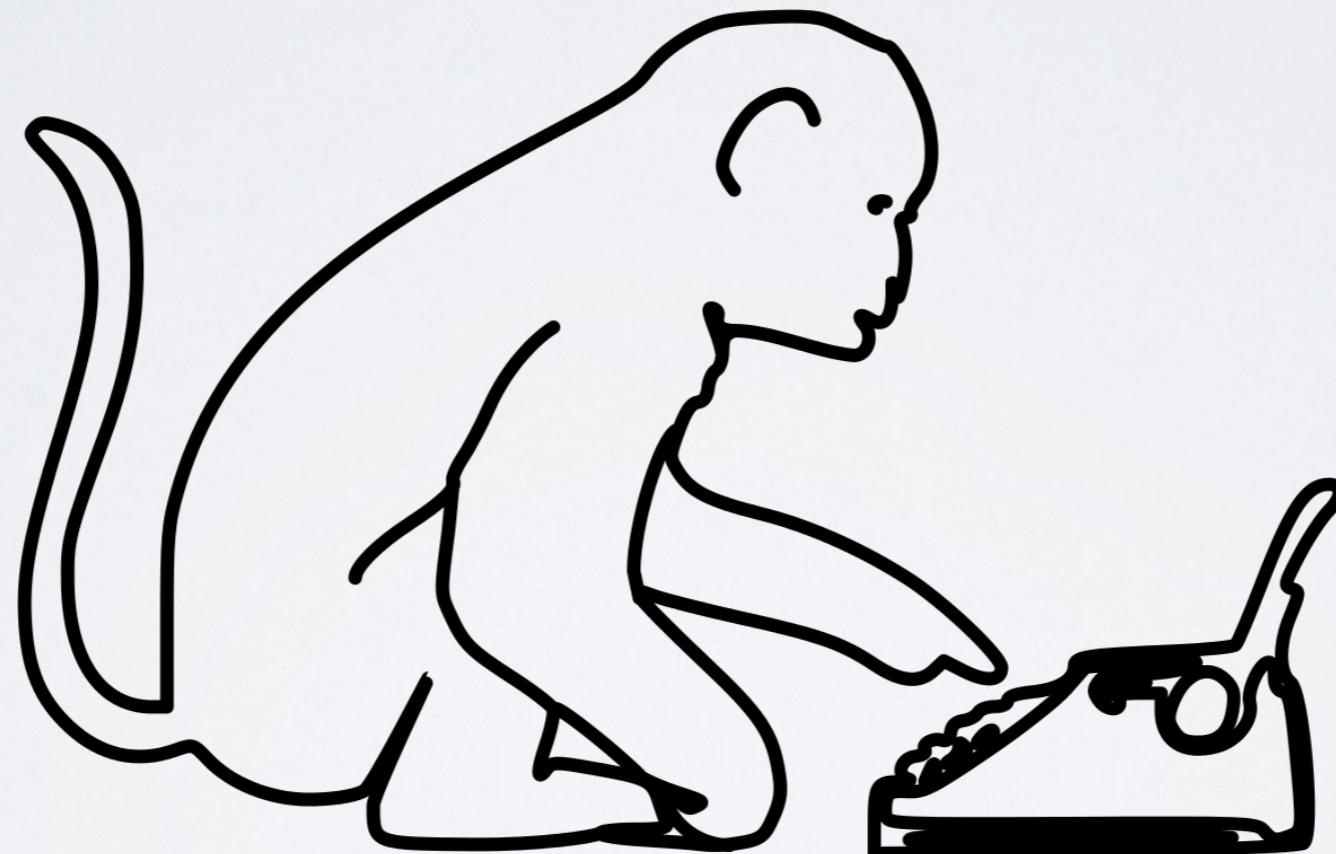


Keine manuelle Wartung.

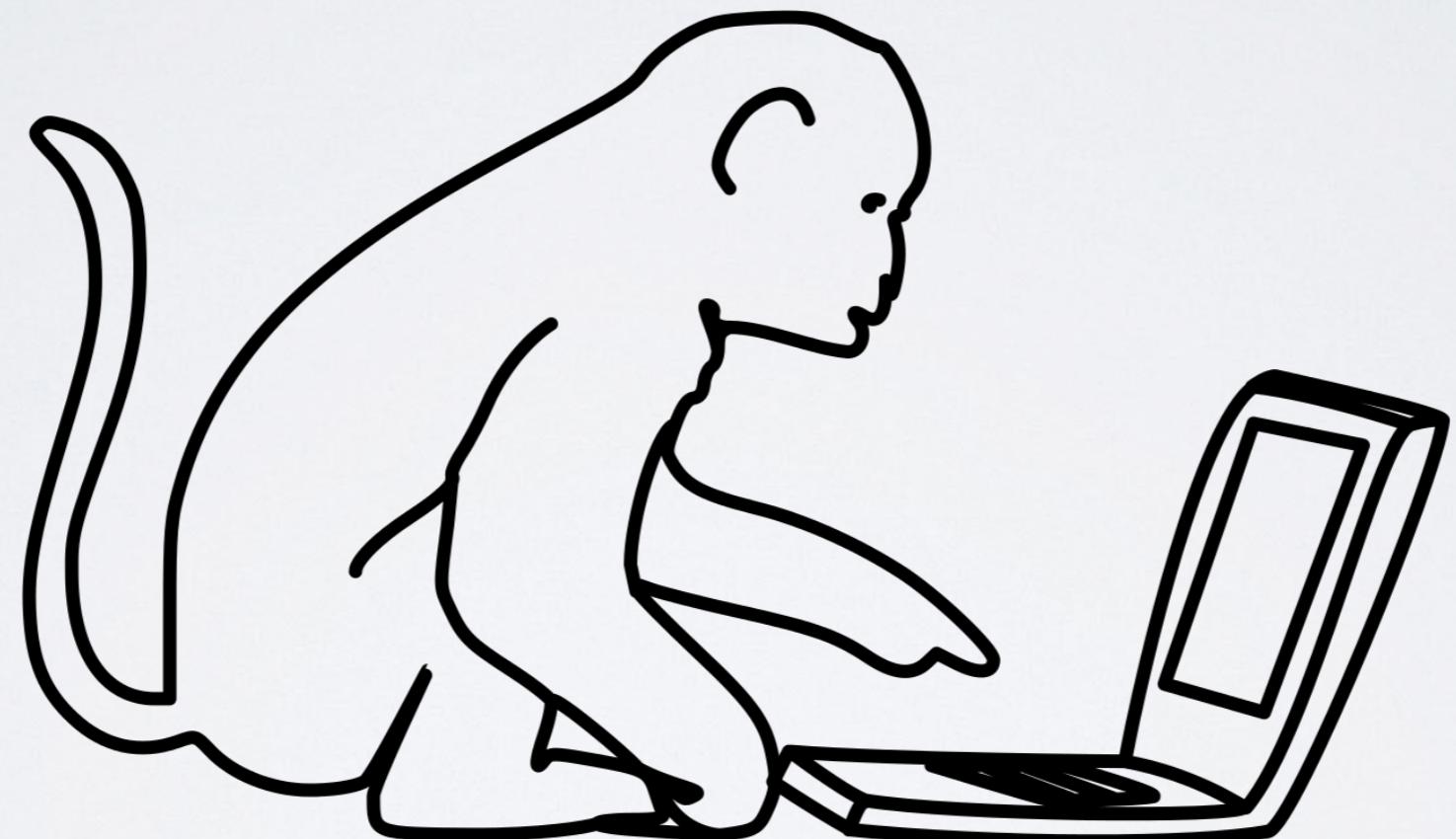


Weniger Kosten.

**Infinite-Monkey-Theorem:
Wenn eine Affe nur lange genug auf einer
Schreibmaschine tippt,
schreibt er irgendwann alle Werke von Shakespeare.**



**Wir ersetzen die Schreibmaschine
mit einem Computer...**



Infinite Monkey

```
1. import random, string
2. from random import randint
3. from pymouse import PyMouse
4. from pykeyboard import PyKeyboard
5.
6. m = PyMouse()
7. k = PyKeyboard()
8. while True:
9.     m.click(randint(1, 800), randint(1, 600), 1)
10.    k.type_string(''.join(random.choice(string.lowercase)
11.        for i in range(randint(0,100)))))
```

Infinite Monkey

DEMO

ReTest – Vollautomatisches Reg...

 ReTest

Start ReTest News Kontakt

Vollautomatisches Regressionstesten.

Weniger Risiko, weniger Stress, weniger Kosten!

Haben Sie genug vom Testen?

Direkt aus der Forschung bietet Ihnen ReTest als bisher einziges Produkt seiner Art vollautomatisches Regressionstesten auf Ebene der Benutzerschnittstelle. Damit stellen Sie sicher, dass keine unerwünschten Seiteneffekte mehr ins Projekt kommen.

Robustheitstesten von Corner Cases bekommen Sie gratis dazu.

Mit ReTest behalten Sie die Kontrolle!



^

Workbook2

Search in Sheet

Home Layout Tables Charts SmartArt Formulas Data Review Themes

Font Alignment Number Format Cells Themes

Calibri (Body) 12 General Conditional Formatting Styles Actions Themes

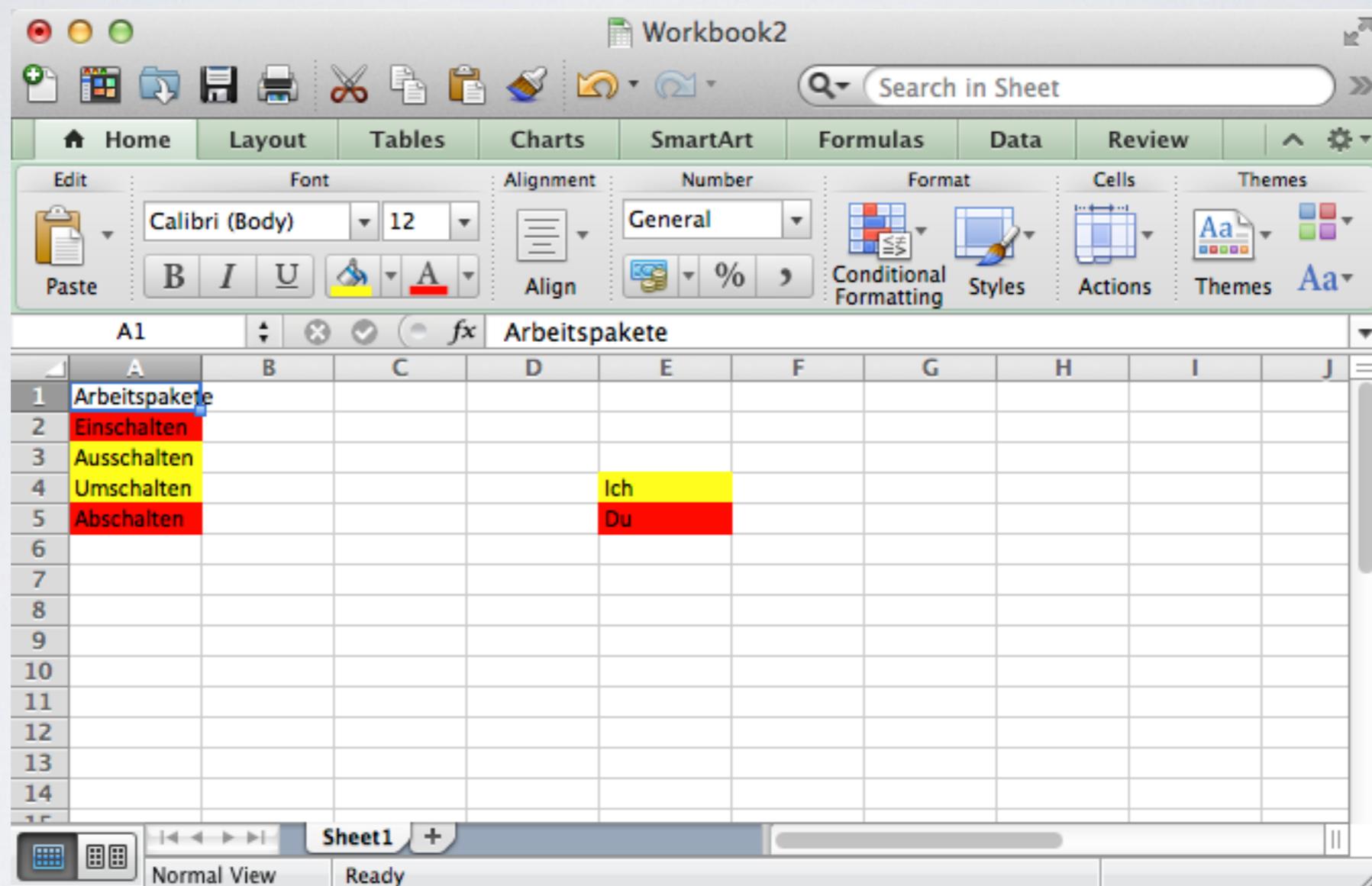
Paste B I U Align % ,

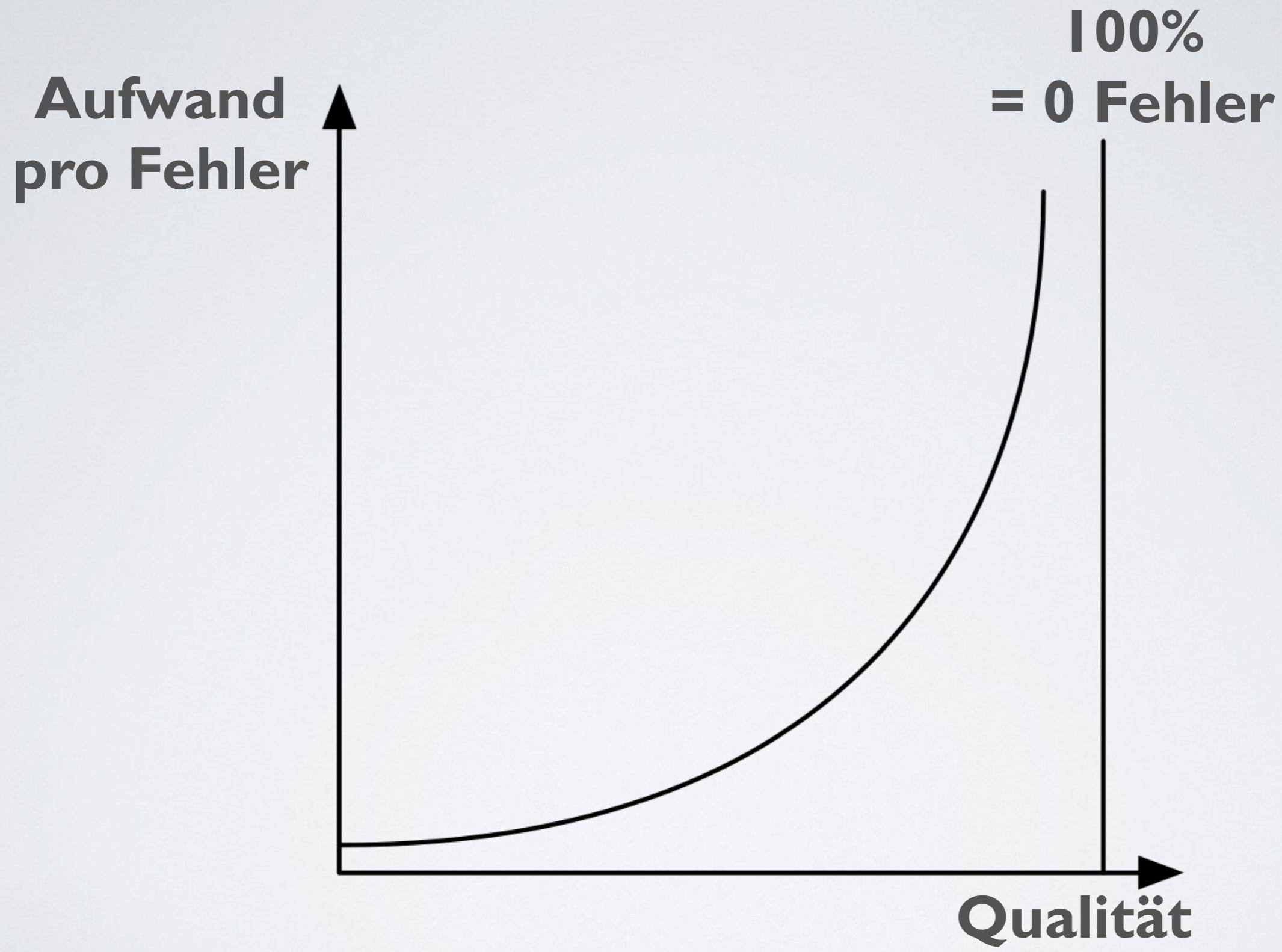
A1 Arbeitspakete

	A	B	C	D	E	F	G	H	I	J
1	Arbeitspakete									
2	Einschalten									
3	Ausschalten									
4	Umschalten									
5	Abschalten									
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										

Sheet1 +

Normal View Ready





Dummer Affe



Intelligenter Affe



Intelligenter Affe

```
1. import sys, random, string
2. from random import randint
3. from selenium import webdriver
4. from selenium.webdriver.common.keys import Keys
5.
6. browser = webdriver.Firefox()
7. browser.get('http://www.retest.de')
8. while True:
9.     links = browser.find_elements_by_tag_name('a')
10.    link = random.choice(links).click()
11.    inputs = browser.find_elements_by_tag_name('input')
12.    if len(inputs) > 0:
13.        inputData = ''.join(random.choice(string.lowercase))
14.        for i in range(randint(0,100)))
15.            random.choice(inputs).send_keys(inputData)
```

Intelligenter Affe

DEMO



Start ReTest News Kontakt

Vollautomatisches Regressionstesten.

Weniger Risiko, weniger Stress, weniger Kosten!

Haben Sie genug vom Testen?
Direkt aus der Forschung bietet Ihnen
ReTest als bisher einziges Produkt
seiner Art vollautomatisches
Regressionstesten auf Ebene der
Benutzerschnittstelle. Damit stellen Sie
sicher, dass keine unerwünschten
Seitereffekte mehr ins Projekt kommen.
Robustheitstesten von Corner Cases
bekommen Sie gratis dazu.
Mit ReTest behalten Sie die Kontrolle!



Intelligenter Affe



Open Source Produkte

NetFlix Chaos Monkey

gremlins.js

UI/Application Exerciser Monkey



AND NOW FOR SOMETHING
COMPLETELY DIFFERENT

Türme von Hanoi



© André Karwath aka Aka

Türme von Hanoi

```
1. class TuermeVonHanoi():
2.     def __init__(self):
3.         self.A = [6, 5, 4, 3, 2, 1]
4.         self.B = []
5.         self.C = []
6.
7.     def AtoB(self): self.B.append(self.A.pop())
8.
9.     def AtoC(self): self.C.append(self.A.pop())
10.
11.    def BtoA(self): self.A.append(self.B.pop())
12.
13.    def BtoC(self): self.C.append(self.B.pop())
14.
15.    def CtoA(self): self.A.append(self.C.pop())
16.
17.    def CtoB(self): self.B.append(self.C.pop())
18.
19.    def valid(self):
20.        return all(self.A[i + 1] < self.A[i] for i in range(len(self.A)-1)) and \
21.               all(self.B[m + 1] < self.B[m] for m in range(len(self.B)-1)) and \
22.               all(self.C[n + 1] < self.C[n] for n in range(len(self.C)-1))
```

Türme von Hanoi

Rekursiv

```
1. def bewege(schritte, i, a_name, b_name, c_name):  
2.     if (i > 0):  
3.         bewege(schritte, i-1, a_name, c_name, b_name)  
4.         schritte.append('tuerme.' + a_name + 'to' + c_name + '())')  
5.         bewege(schritte, i-1, b_name, a_name, c_name)  
6.     return schritte  
7.  
8. def algorithmus():  
9.     return bewege([], 6, 'A', 'B', 'C')
```

Türme von Hanoi

Manuell

```
1. schritte = [ \
2.   'tuerme.AtoB()', '\ #[654321][][] -> [65432][1][] \
3.   'tuerme.AtoC()', '\ #[65432][1][] -> [6543][1][2] \
4.   'tuerme.BtoC()', '\ #[6543][1][2] -> [6543][][21] \
5.   'tuerme.AtoB()', '\ #[6543][][21] -> [654][3][21] \
6.   'tuerme.CtoA()', '\ #[654][3][21] -> [6541][3][2] \
7.   'tuerme.CtoB()', '\ #[6541][3][2] -> [6541][32][] \
8.   'tuerme.AtoB()', '\ #[6541][32][] -> [654][321][] \
9.   'tuerme.AtoC()', '\ #[654][321][] -> [65][321][4] \
10.  'tuerme.BtoC()', '\ #[65][321][4] -> [65][32][41] \
11.  'tuerme.BtoA()', '\ #[65][32][41] -> [652][3][41] \
12.  'tuerme.CtoA()', '\ #[652][3][41] -> [6521][3][4] \
13.  'tuerme.BtoC()', '\ #[6521][3][4] -> [6521][][43] \
14.  'tuerme.AtoB()', '\ #[6521][][43] -> [652][1][43] \
15.  'tuerme.AtoC()', '\ #[652][1][43] -> [65][1][432] \
16.  'tuerme.BtoC()', '\ #[65][1][432] -> [65][][4321] \
17.  'tuerme.AtoB()', '\ #[65][][4321] -> [6][5][4321] \
18.  'tuerme.CtoA()', '\ #[6][5][4321] -> [61][5][432]
```

Türme von Hanoi

Möglichkeiten

$>2^{63}$

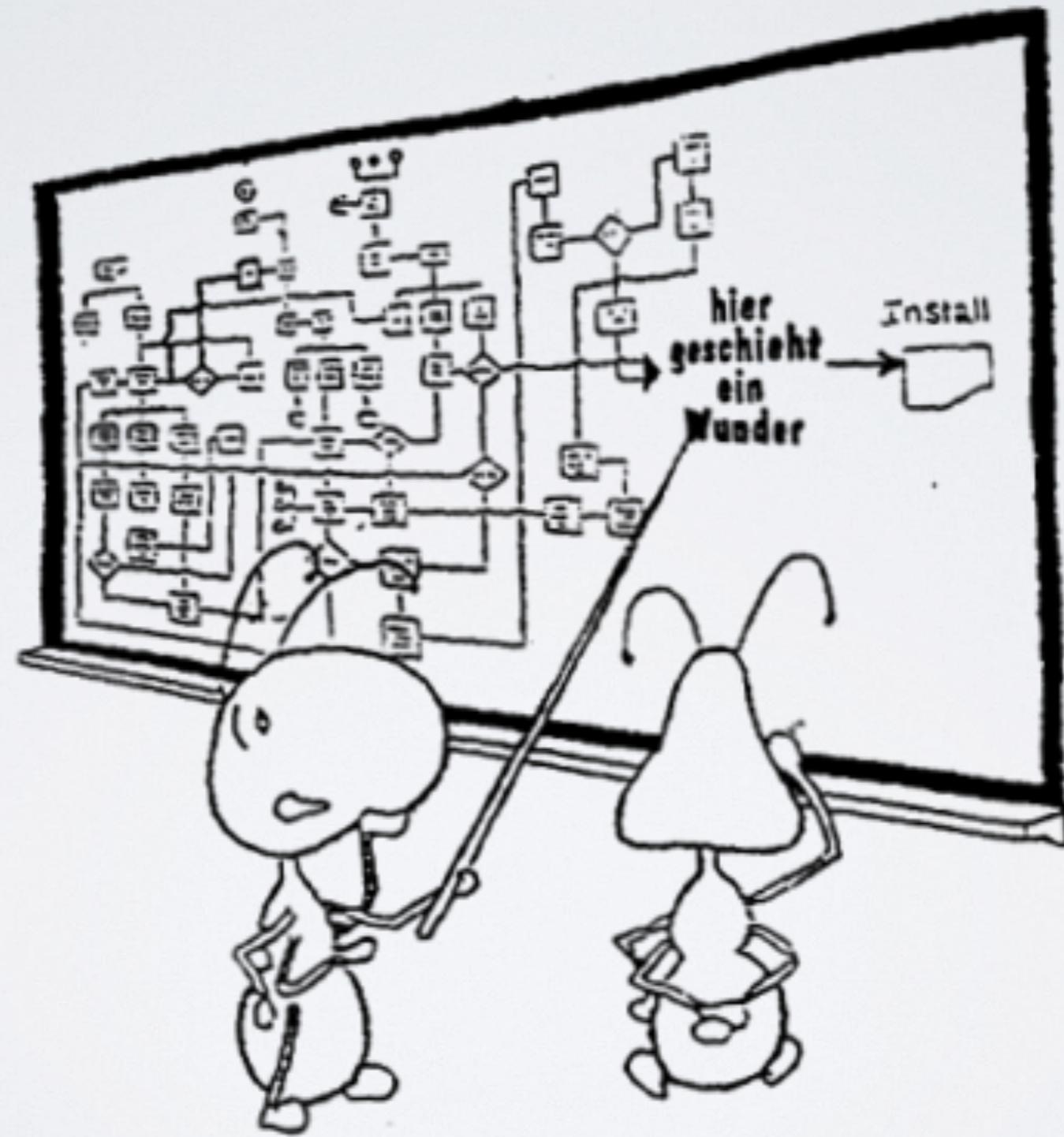
Türme von Hanoi Möglichkeiten

9.223.372.036.854.775.808

Türme von Hanoi Möglichkeiten

292.471.208 Jahre

Türme von Hanoi Automatisch?



Türme von Hanoi

Zielfunktion

```
1. def min_zielfunktion(tuerme): return (21 - sum(tuerme.C))
```

Türme von Hanoi

Bergsteigeralgorithmus

```
1. def algorithmus():
2.     aktueller_zustand = []
3.     aktueller_zielwert = max_wert
4.     idx = 0
5.     while idx < len(optionen) and aktueller_zielwert > 0:
6.         neuer_zustand = kopiere(aktueller_zustand)
7.         neuer_zustand.append(optionen[idx])
8.         neuer_zielwert = berechne_zielwert(neuer_zustand)
9.         if neuer_zielwert < aktueller_zielwert:
10.             aktueller_zustand = neuer_zustand
11.             aktueller_zielwert = neuer_zielwert
12.             idx = 0
13.             idx += 1
14.     return aktueller_zustand
15.
16.
17.
18.
```

Türme von Hanoi

Stochastischer Bergsteigeralgorithmus

```
1.import random
2.random.seed(0)
3.
4.def algorithmus():
5.    aktueller_zustand = []
6.    aktueller_zielwert = max_wert
7.    count = 0
8.    while count < versuche and aktueller_zielwert > 0:
9.        neuer_zustand = kopiere(aktueller_zustand)
10.       neuer_zustand.append(random.choice(optionen))
11.       neuer_zielwert = berechne_zielwert(neuer_zustand)
12.       if neuer_zielwert < aktueller_zielwert:
13.           aktueller_zustand = neuer_zustand
14.           aktueller_zielwert = neuer_zielwert
15.       count += 1
16.    return aktueller_zustand
17.
18.
```

Türme von Hanoi

Stochastischer Bergsteigeralgorithmus

DEMO

Türme von Hanoi

Stochastischer Bergsteigeralgorithmus

```
['tuerme.AtoC()'] -> [65432][][1], Anzahl Schritte: 1, Zielwert: 20
```

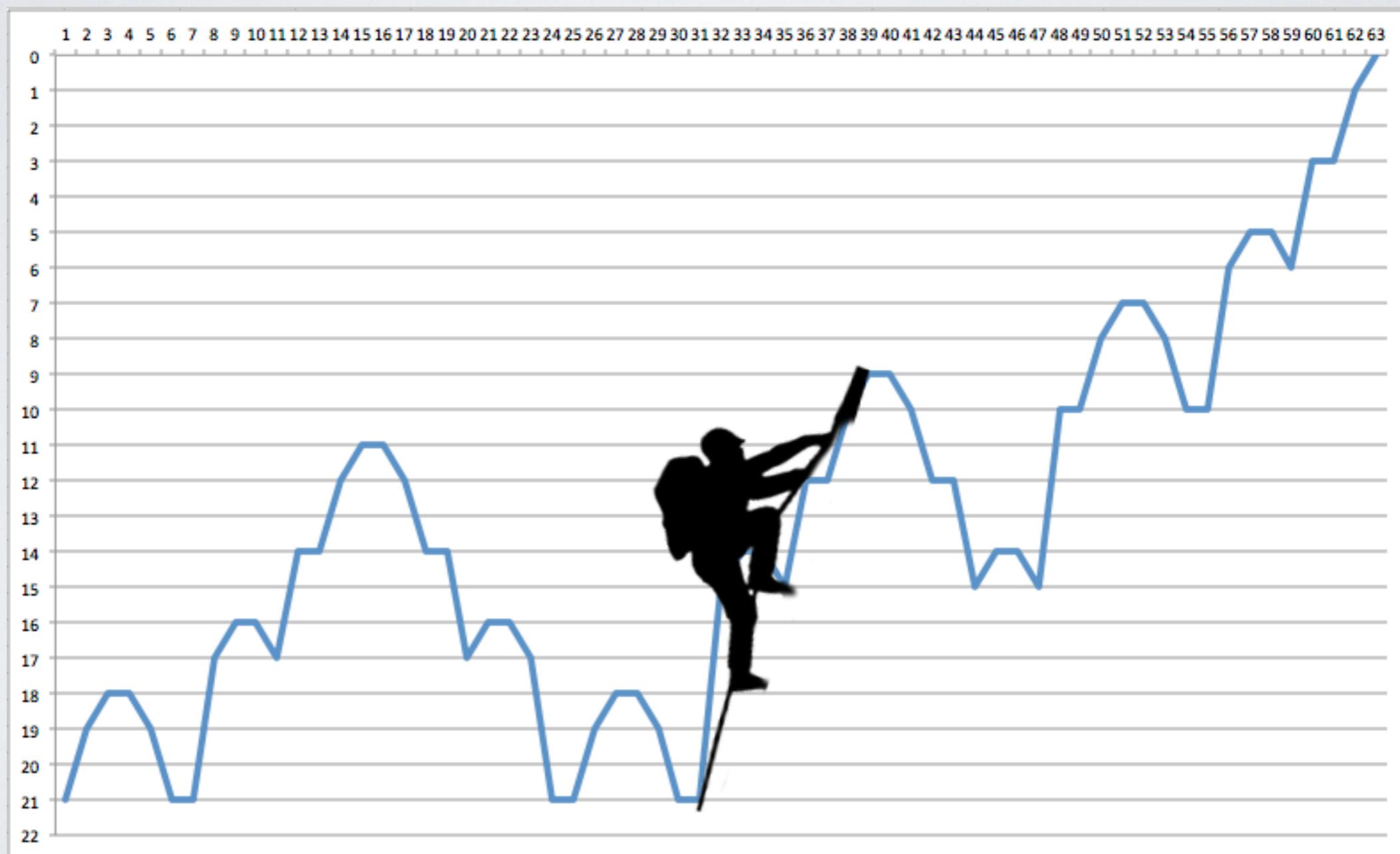
Türme von Hanoi

Zielfunktion

```
1. def min_zielfunktion(tuerme): return (21 - sum(tuerme.C))
```

Türme von Hanoi

Suchraumlandschaft



Türme von Hanoi

Zufallsbewegung

```
1. def algorithmus():
2.     aktueller_zustand = []
3.     aktueller_zielwert = max_wert
4.     count = 0
5.     while count < versuche and aktueller_zielwert > 0:
6.         neuer_zustand = kopiere(aktueller_zustand)
7.         neuer_zustand.append(random.choice(optionen))
8.         neuer_zielwert = berechne_zielwert(neuer_zustand)
9.         if neuer_zielwert < aktueller_zielwert or \
10.             (neuer_zielwert < max_wert and random.choice([True, False])):
11.             aktueller_zustand = neuer_zustand
12.             aktueller_zielwert = neuer_zielwert
13.         count += 1
14.     return aktueller_zustand
15.
16.
17.
18.
```

Türme von Hanoi

Zufallsbewegung

DEMO

Türme von Hanoi

Zufallsbewegung

```
[tuerme.AtoB(), tuerme.BtoC(), tuerme.CtoB(), tuerme.BtoA(),  
tuerme.AtoB(), ...] -> [51][][6432], Anzahl Schritte: 2018, Zielwert: 6
```

Türme von Hanoi

Zufallsbewegung

```
1. def algorithmus():
2.     aktueller_zustand = []
3.     aktueller_zielwert = max_wert
4.     count = 0
5.     while count < versuche and aktueller_zielwert > 0:
6.         neuer_zustand = list(aktueller_zustand)
7.         neuer_zustand.append(random.choice(optionen))
8.         neuer_zielwert = berechne_zielwert(neuer_zustand)
9.         if neuer_zielwert < aktueller_zielwert or \
10.             (neuer_zielwert < max_wert and random.choice([True, False])):
11.             aktueller_zustand = neuer_zustand
12.             aktueller_zielwert = neuer_zielwert
13.         count += 1
14.     return aktueller_zustand
15.
16.
17.
18.
```

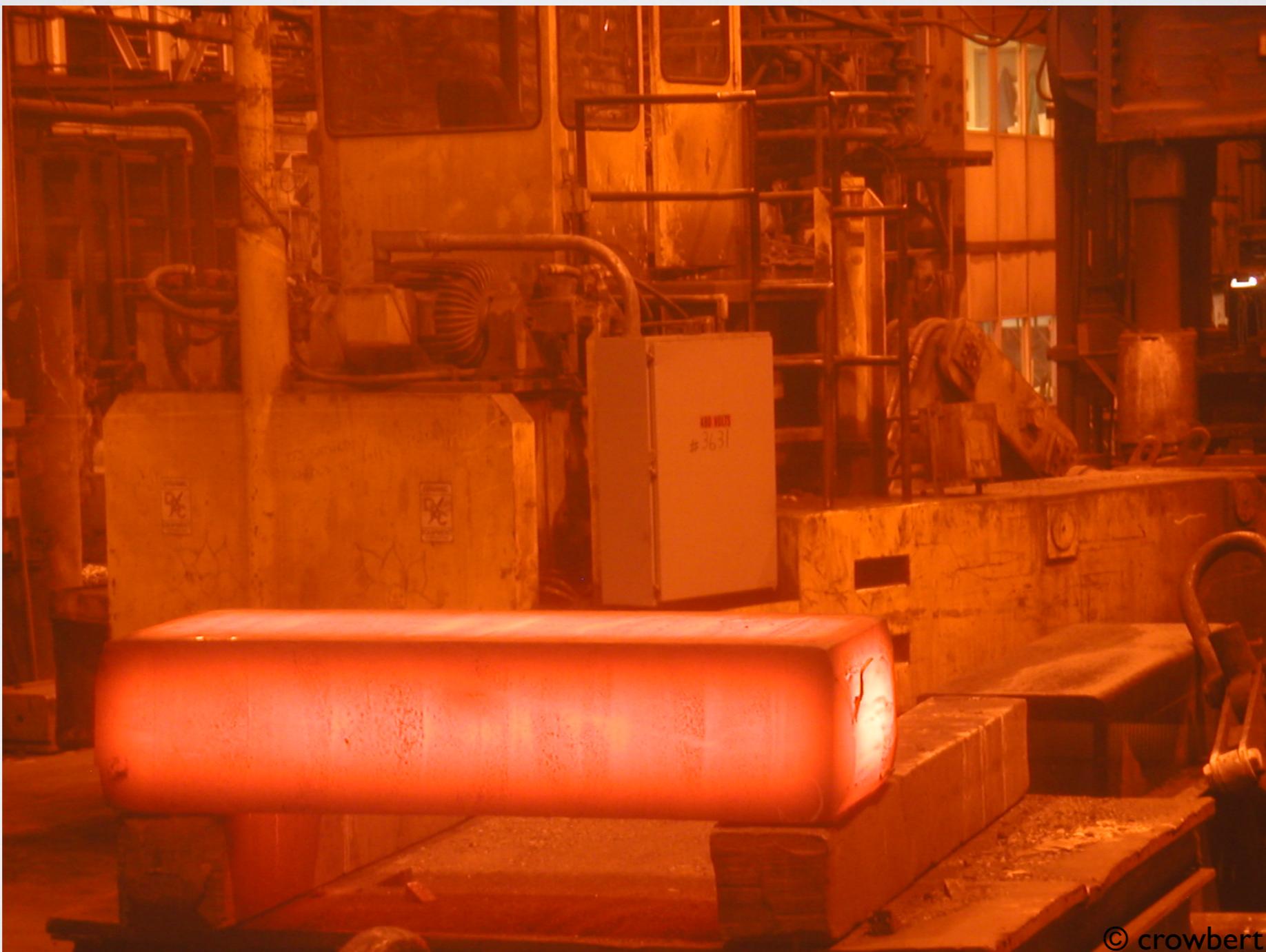
Türme von Hanoi

Zufallsbewegung

```
1. def algorithmus():
2.     aktueller_zustand = []
3.     aktueller_zielwert = max_wert
4.     count = 0
5.     while count < versuche and aktueller_zielwert > 0:
6.         neuer_zustand = list(aktueller_zustand)
7.         neuer_zustand.append(random.choice(optionen))
8.         neuer_zielwert = berechne_zielwert(neuer_zustand)
9.         if neuer_zielwert < aktueller_zielwert or \
10.             (neuer_zielwert < max_wert and randint(0, versuche) > count):
11.             aktueller_zustand = neuer_zustand
12.             aktueller_zielwert = neuer_zielwert
13.             count += 1
14.     return aktueller_zustand
15.
16.
17.
18.
```

Türme von Hanoi

Simulierte Abkühlung



© crowbert

Quelle: <http://www.flickr.com/photos/51035774131@N01/27145468/in/photostream/>

Türme von Hanoi

Simulierte Abkühlung

DEMO

Türme von Hanoi

Simulierte Abkühlung

```
[tuerme.AtoB(), tuerme.AtoC(), tuerme.BtoA(), tuerme.AtoB(), tuerme.BtoA(), ...] ->
[6][][54321], Anzahl Schritte: 1869, Zielwert: 6
```

Türme von Hanoi

Simulierte Abkühlung

```
1. def algorithmus():
2.     aktueller_zustand = []
3.     aktueller_zielwert = max_wert
4.     count = 0
5.     while count < versuche and aktueller_zielwert > 0:
6.         neuer_zustand = list(aktueller_zustand)
7.         neuer_zustand.append(random.choice(optionen))
8.         neuer_zielwert = berechne_zielwert(neuer_zustand)
9.         if neuer_zielwert < aktueller_zielwert or \
10.             (neuer_zielwert < max_wert and randint(0, versuche) > count):
11.             aktueller_zustand = neuer_zustand
12.             aktueller_zielwert = neuer_zielwert
13.             count += 1
14.     return aktueller_zustand
15.
16.
17.
18.
```

Türme von Hanoi

Balkensuche

```
1. def algorithmus():
2.     aktuelle_zustaende = [(max_wert, [])] for i in range(0, k)]
3.     count = 0
4.     while count < versuche and aktuelle_zustaende[0][0] > 0:
5.         neue_zustaende = []
6.         for idx in xrange(len(aktuelle_zustaende)):
7.             (aktueller_zielwert, aktueller_zustand) = ziehe(aktuelle_zustaende)
8.             neuer_zustand = kopiere(aktueller_zustand)
9.             neuer_zustand.append(random.choice(optionen))
10.            neuer_zielwert = berechne_zielwert(neuer_zustand)
11.            if neuer_zielwert < aktueller_zielwert or \
12.                (neuer_zielwert < max_wert and randint(1, versuche) > count):
13.                insert(neue_zustaende, neuer_zielwert, neuer_zustand)
14.            else:
15.                insert(neue_zustaende, aktueller_zielwert, aktueller_zustand)
16.            count += 1
17.        aktuelle_zustaende = neue_zustaende
18.    return aktuelle_zustaende[0][1]
```

Türme von Hanoi

Balkensuche

DEMO

Türme von Hanoi

Balkensuche

```
[tuerme.AtoC(), tuerme.AtoB(), tuerme.CtoB(), tuerme.BtoC(), tuerme.CtoA(), ...] ->  
[5][][64321], Anzahl Schritte: 1891, Zielwert: 5
```

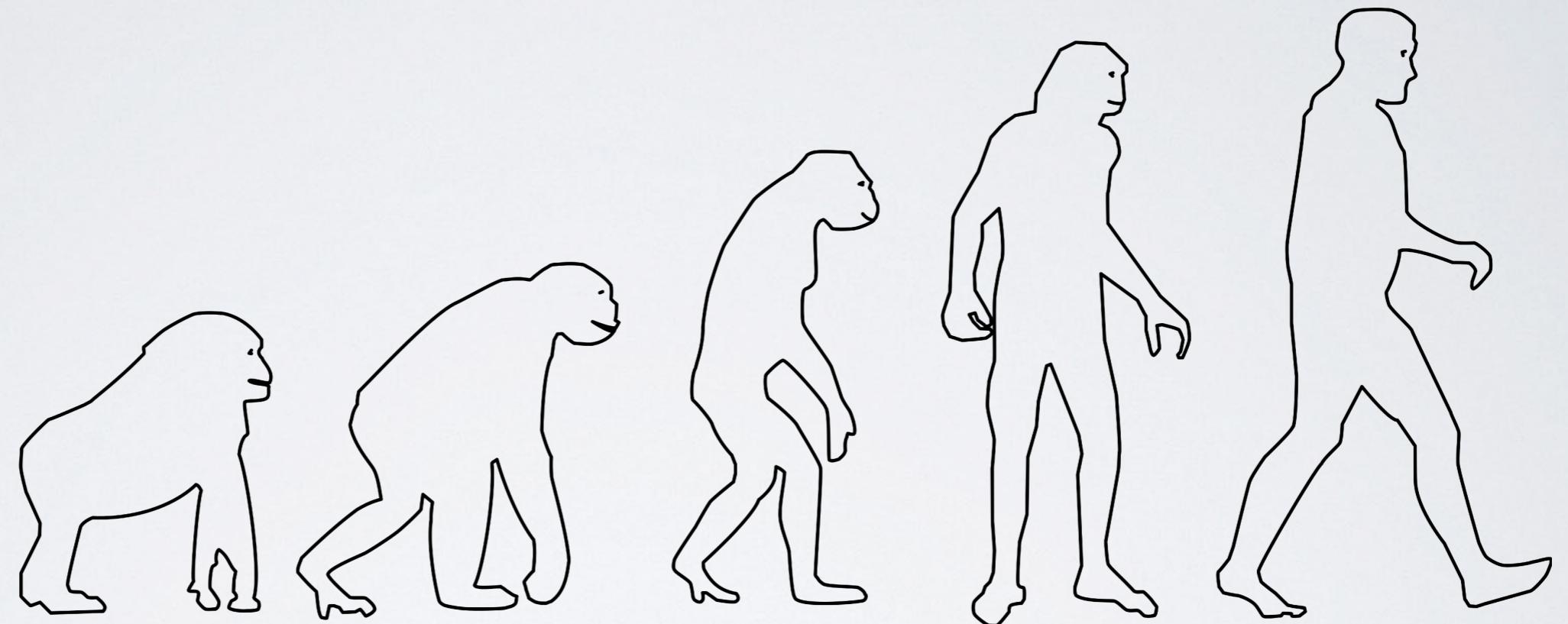
Türme von Hanoi

Balkensuche

```
1. def algorithmus():
2.     aktuelle_zustaende = generiere_initiale(k)
3.     count = 0
4.     while count < versuche and aktuelle_zustaende[0][0] > 0:
5.         neue_zustaende = []
6.         for idx in xrange(len(aktuelle_zustaende)):
7.             (aktueller_zielwert, aktueller_zustand) = ziehe(aktuelle_zustaende)
8.             neuer_zustand = kopiere(aktueller_zustand)
9.             neuer_zustand.append(random.choice(optionen))
10.            neuer_zielwert = berechne_zielwert(neuer_zustand)
11.            if neuer_zielwert < aktueller_zielwert or \
12.                (neuer_zielwert < max_wert and randint(1, versuche) > count):
13.                insert(neue_zustaende, neuer_zielwert, neuer_zustand)
14.            else:
15.                insert(neue_zustaende, aktueller_zielwert, aktueller_zustand)
16.            count += 1
17.        aktuelle_zustaende = neue_zustaende
18.    return aktuelle_zustaende[0][1]
```

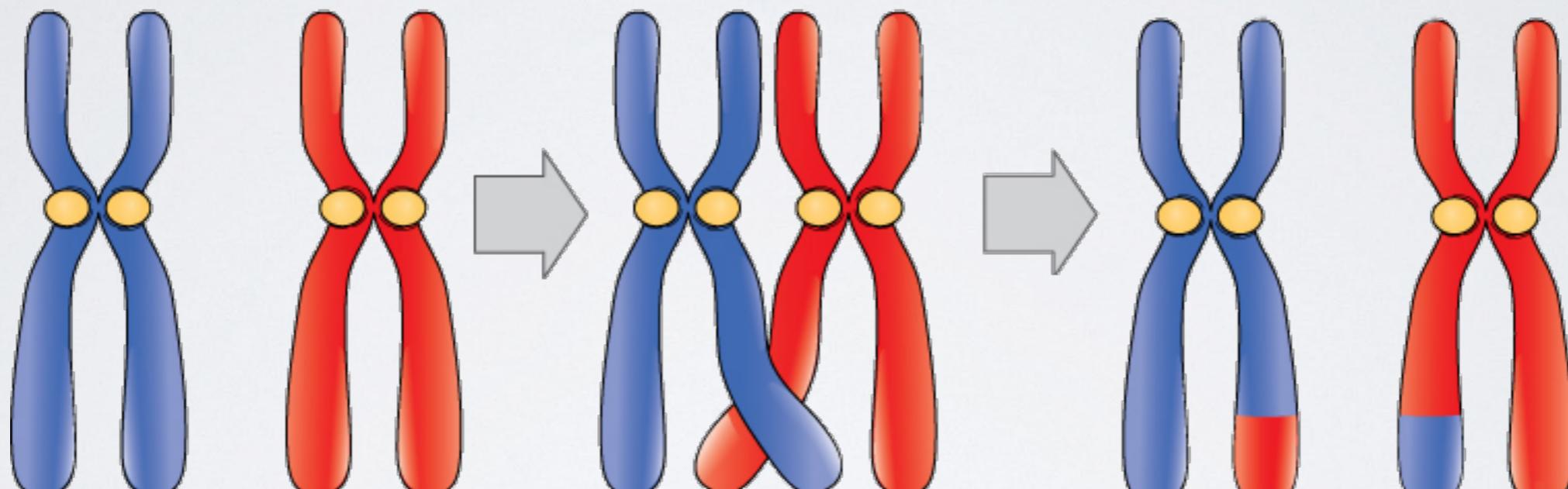
Türme von Hanoi

Genetischer Algorithmus



Türme von Hanoi

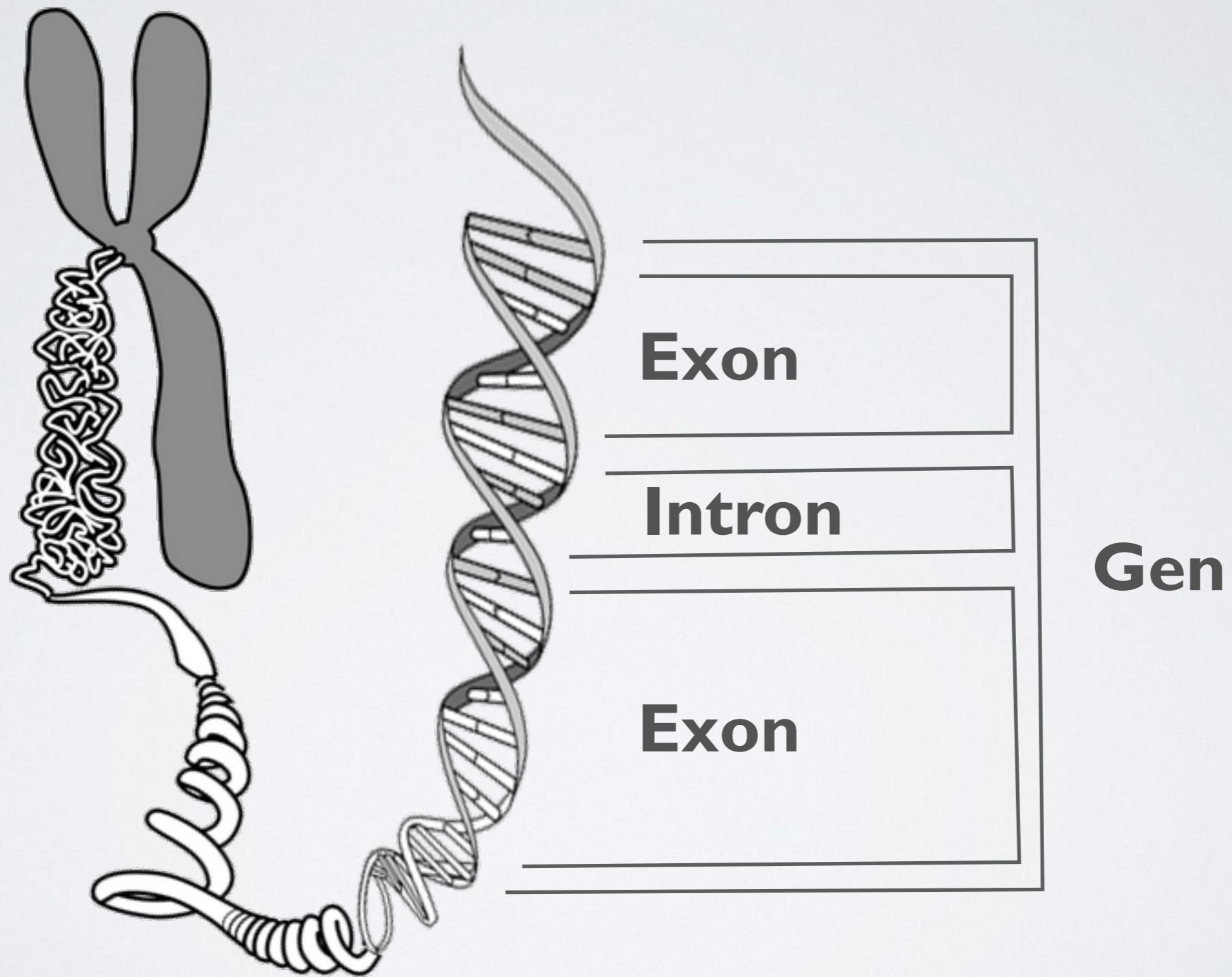
Genetischer Algorithmus



© Robert Bear and David Rintoul

Türme von Hanoi

Genetischer Algorithmus



Türme von Hanoi

Genetischer Algorithmus

```
1. def algorithmus():
2.     aktuelle_zustaende = generiere_initiale(k)
3.     count = 0
4.     while count < versuche and aktuelle_zustaende[0][0] > 0:
5.         neue_zustaende = []
6.         for idx in xrange(len(aktuelle_zustaende)):
7.             neuer_zustand = ziehe(aktuelle_zustaende)
8.             neuer_zustand = rekombiniere(neuer_zustand, ziehe(aktuelle_zustaende))
9.             neuer_zustand = mutiere(neuer_zustand)
10.            if neuer_zielwert < aktueller_zielwert or \
11.                (neuer_zielwert < max_wert and randint(1, versuche) > count):
12.                insert(neue_zustaende, neuer_zustand)
13.            else:
14.                insert(neue_zustaende, aktueller_zustand)
15.            count += 1
16.        aktuelle_zustaende = neue_zustaende
17.    return aktuelle_zustaende[0][1]
18.
```

Türme von Hanoi

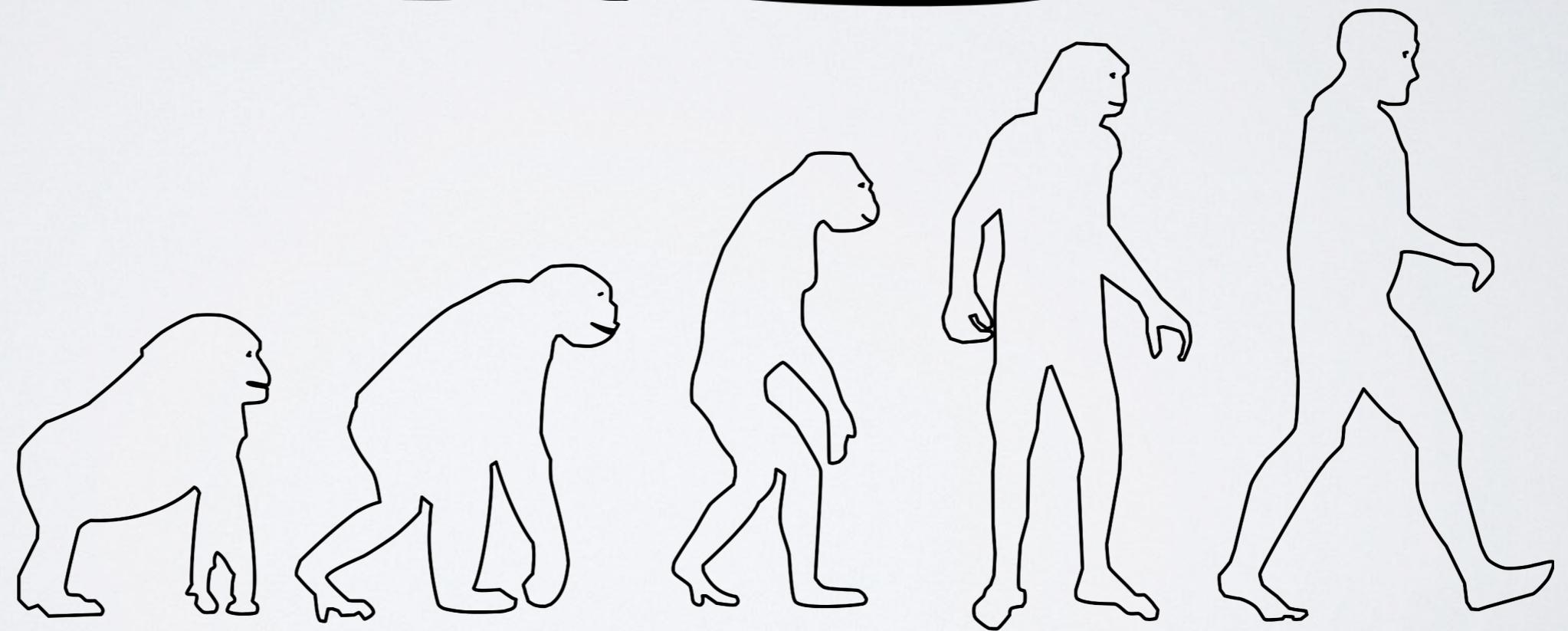
Genetischer Algorithmus

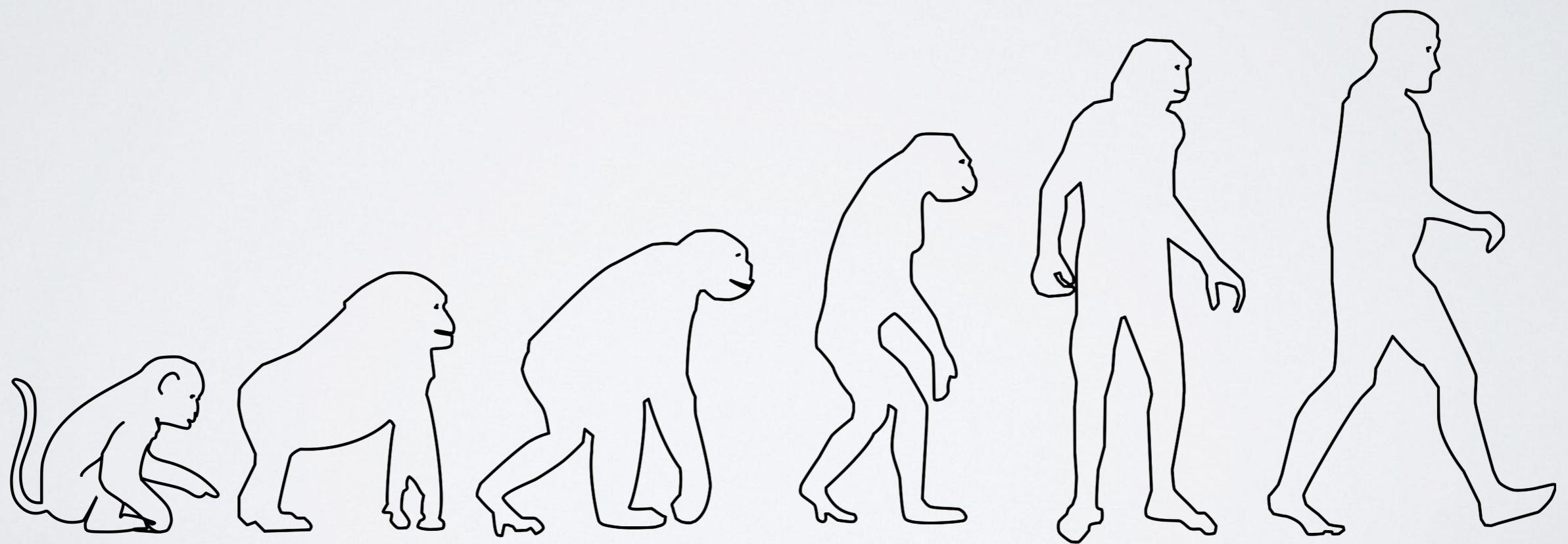
DEMO

Türme von Hanoi

Genetischer Algorithmus

```
[tuerme.AtoB(), tuerme.AtoC(), tuerme.BtoA(), tuerme.AtoB(), tuerme.BtoA(), ...] ->
[[[]][654321], Anzahl Schritte: 1727, Zielwert: 0
```





hr



AND NOW FOR SOMETHING
COMPLETELY DIFFERENT

When is a bug not a bug?



When it's a feature!

Is it a bug?

```
1. def auth(username, password):
2.     if username == 'admin' and password == 'geheim':
3.         return True
4.     if hash(password + get_salt(username)) == get_pwd_hash(username):
5.         return True
6.     return False
```

Is it a bug?

```
.
```

```
..
```

```
text.c
```

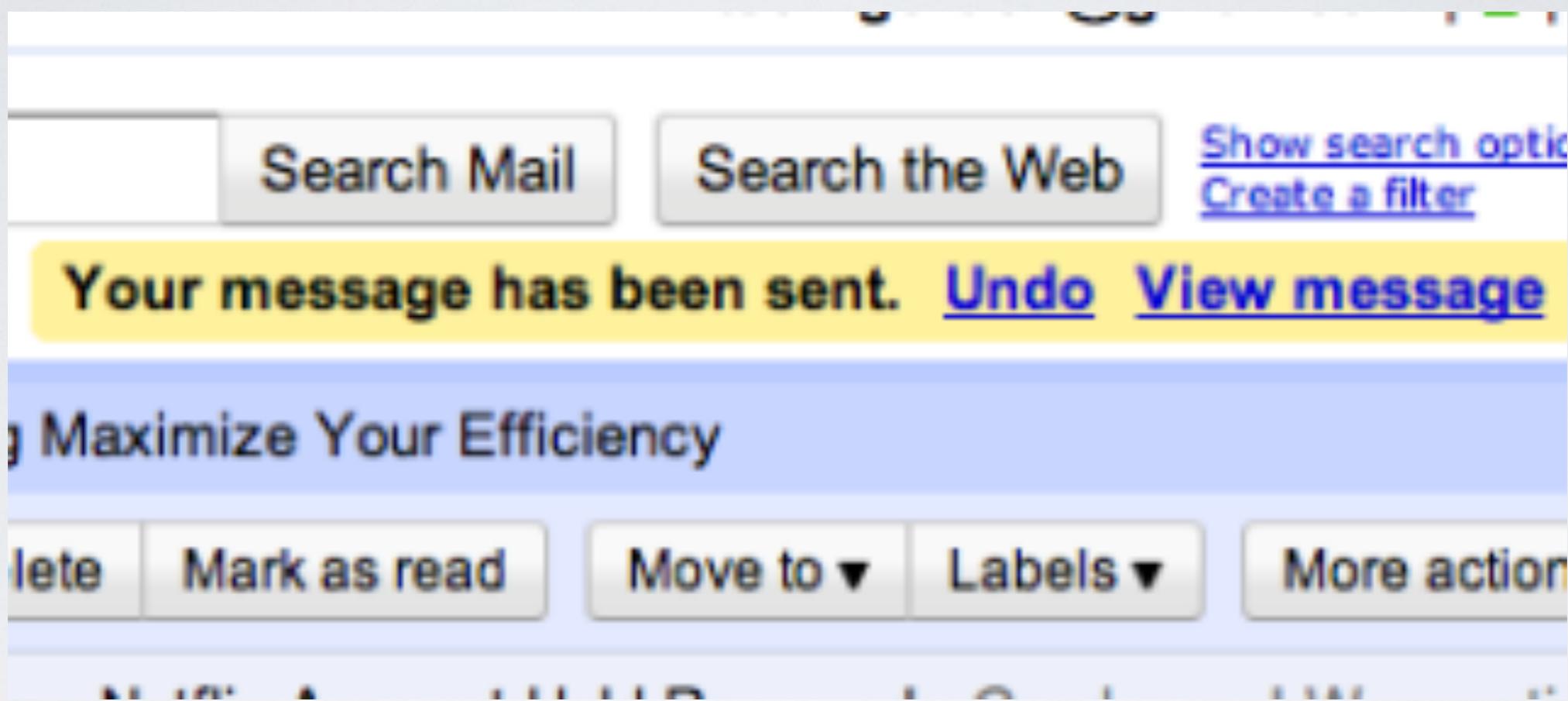
```
other.files
```

```
1. if (name[0] == '.') continue;
```

Is it a bug?



Is it a bug?



What is a bug?

“ *Without specification, there are no bugs
— only surprises.*

Brian Kernighan

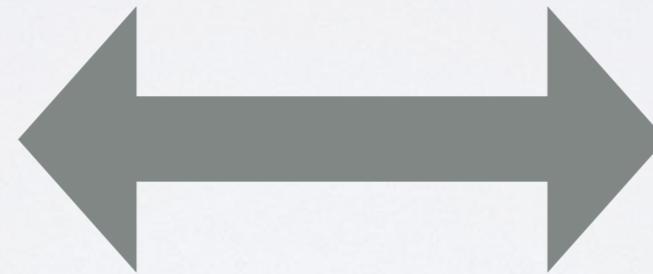
What is a bug?

Spezifikation



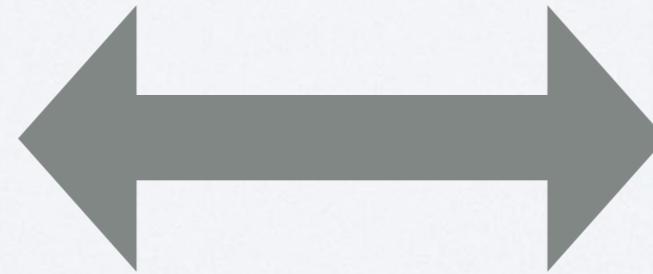
Code

Modell



Code

Nutzererwartung



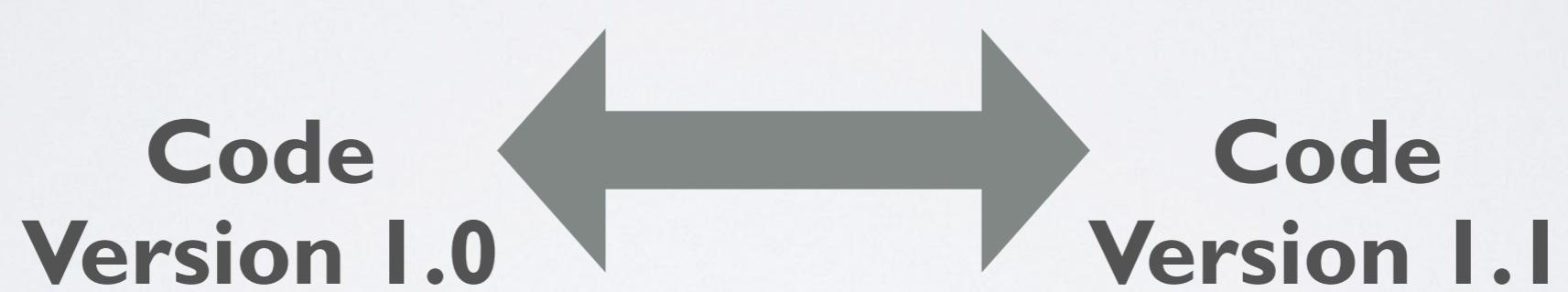
Code

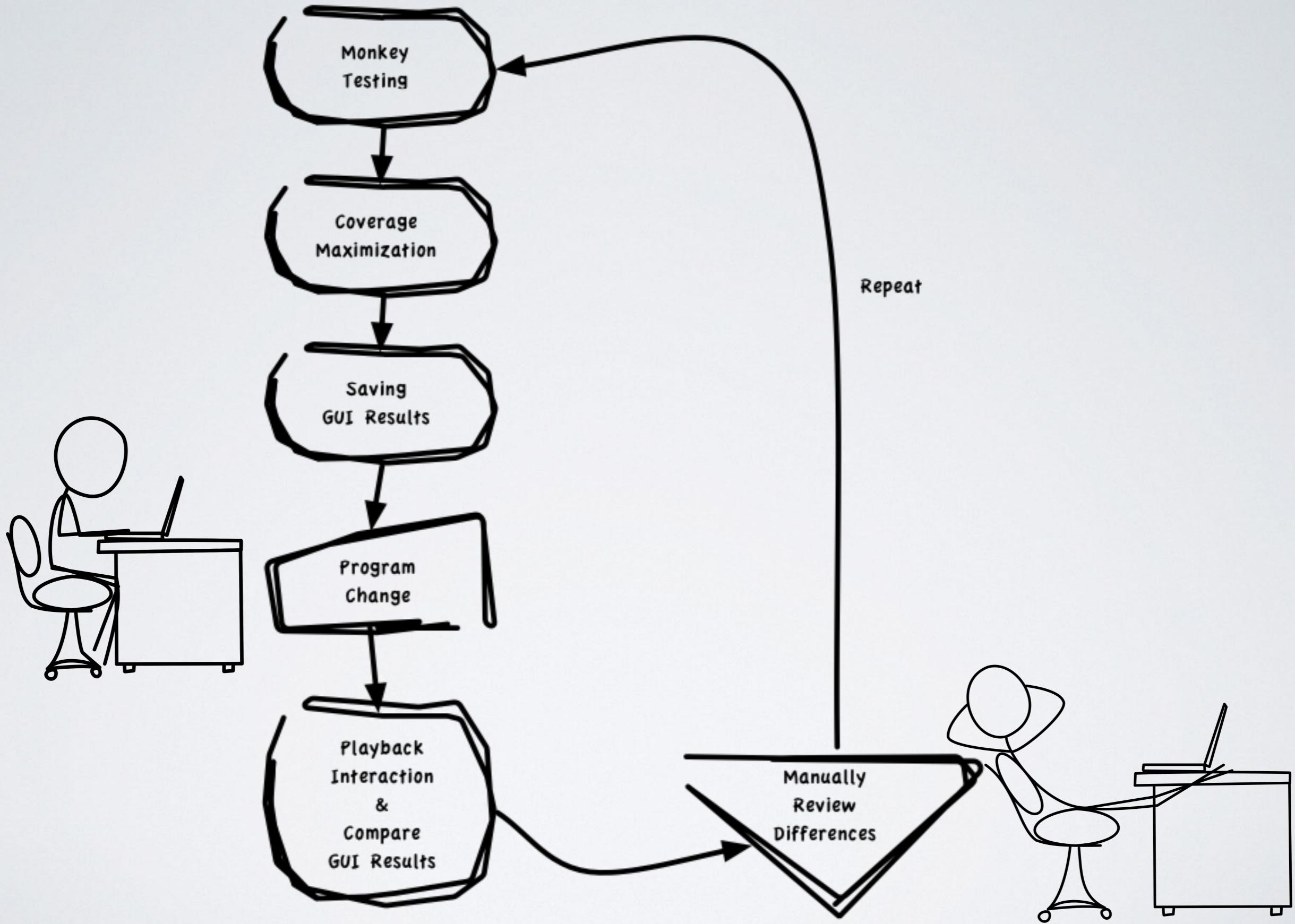
What is a bug?

Crash

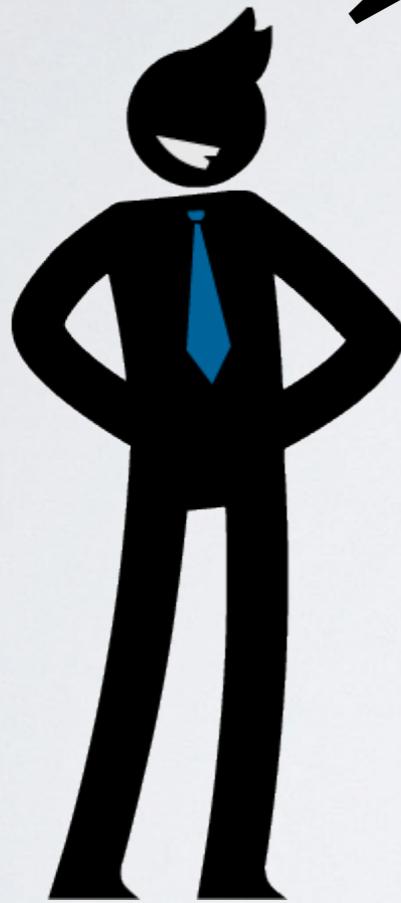
Hang up

Monkey Testing





Durchschnittlicher Aufwand für Regressionstesten?

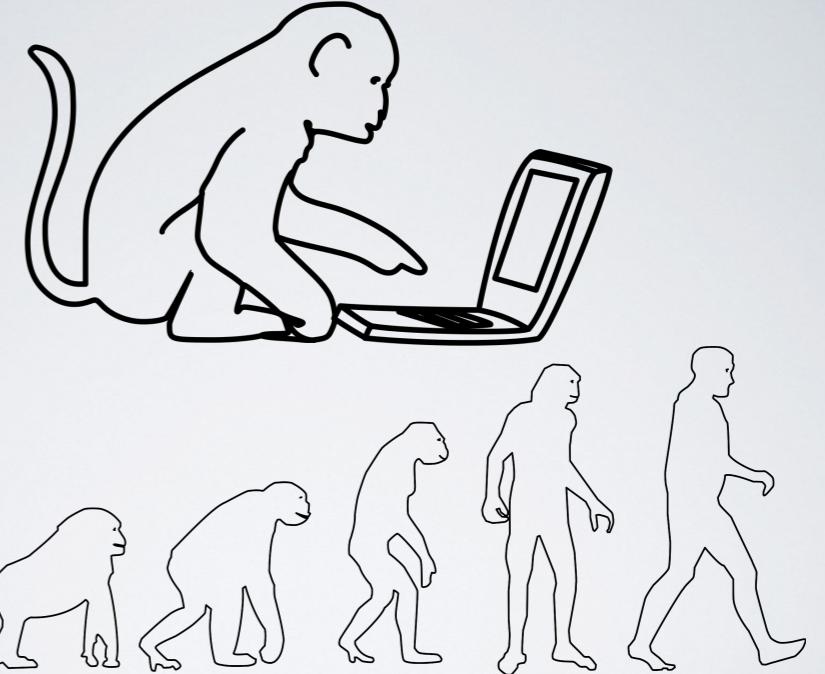
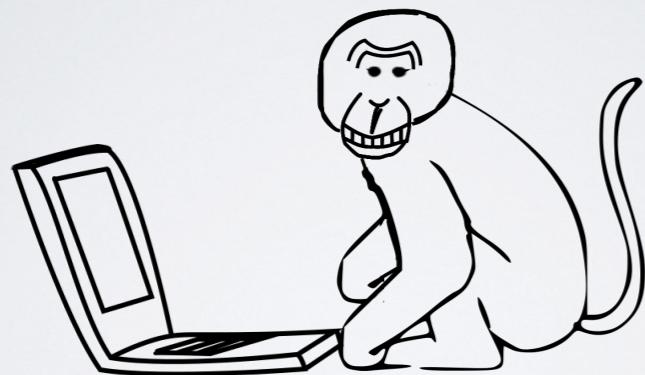


Machen Sie den Test!



www.retest.de

Intelligenter Affe



Material und Infos auf: [www.retest.de!](http://www.retest.de)

When is a bug not a bug?



When it's a feature!

Machen Sie den Test!



 ReTest

www.retest.de