Haskell aus einer Java EE Perspektive

Torsten Fink
torsten.fink@akquinet.de
akquinet

Zu meiner Person

1999-2003:

- Promotion über ausführbare Softwarearchitekturen für verteilte Systeme

Ab 2004: Berater bei der akquinet

- technischer Architekt in J2EE-Projekten
- Projektleitung
- Betriebsführung, Wartung
- klassische Beratung und Schulungen

2006-2010:

Leiter des JBoss-Competence-Centers bei der akquinet

Ab 2011:

Geschäftsführer der akquinet tech@spree

Unsere Unternehmensstruktur

EK: Eigenkapital

Geschäftsprozesse
business consulting GmbH
100 T€ EK

akquinet AG
2,4 Mio. € EK

***Institut** für Softwaretechnik und Outsourcing an der FH Wedel GmbH 100 T€ EK*

JAVA

Java/JBoss
tech@spree GmbH
300 T€ EK

Java Entwicklung
engineering GmbH
50 T€ EK

agile Entwicklung
it-agile GmbH
100 T€ EK

SAP

Security
enterprise solutions
GmbH 130 T€ EK

Öffentlicher Sektor
public consulting &
services GmbH
100 T€ EK

ECM-Lösungen, Polen
gbs GmbH 25 T€ EK

SAP-Lösungen, Österreich
HKS business techn.
GmbH 75 T€ EK

Microsoft / .NET

MS ERP Lösungen
dynamic solutions GmbH
370 T€ EK

Logistik Lösungen
SLS logistics GmbH
250 T€ EK

CRM und Sharepoint
business solutions GmbH
100 T€ EK

Business Intelligence
finance und controllling
GmbH 100 T€ EK

Sozialwirtschaft
care GmbH 100 T€ EK

Sanitätshäuser
ristec GmbH 52 T€ EK

Outsourcing

RZ Betrieb
outsourcing gem. GmbH
250 T€ EK

RZ Services
business service GmbH
25 T€ EK

Infrastruktur
system integration
GmbH 125 T€ EK

Applikationen
hosting services GmbH
125 T€ EK

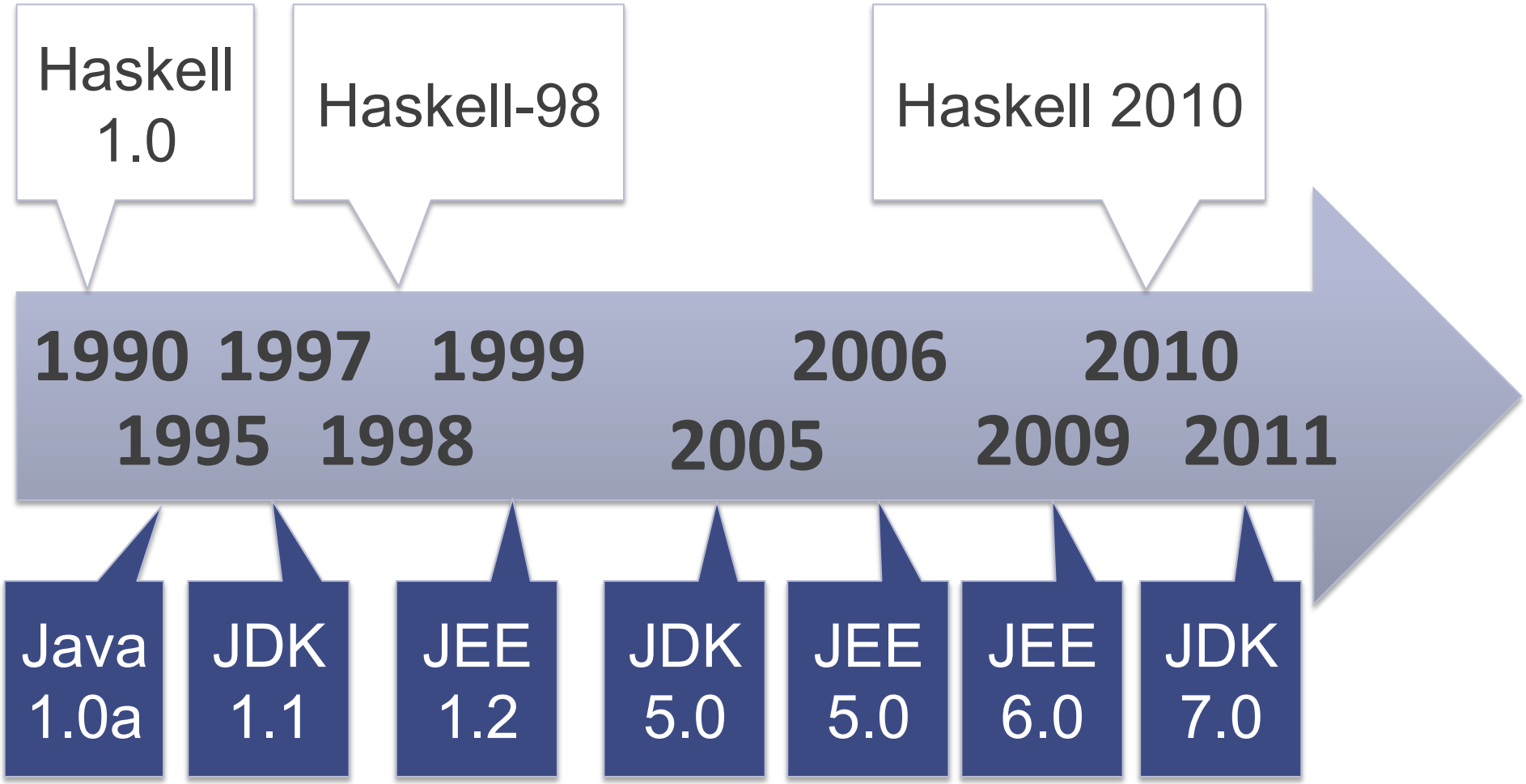
RZ Planung und Erstellung
data center competence
GmbH 100 T€ EK

Memories....

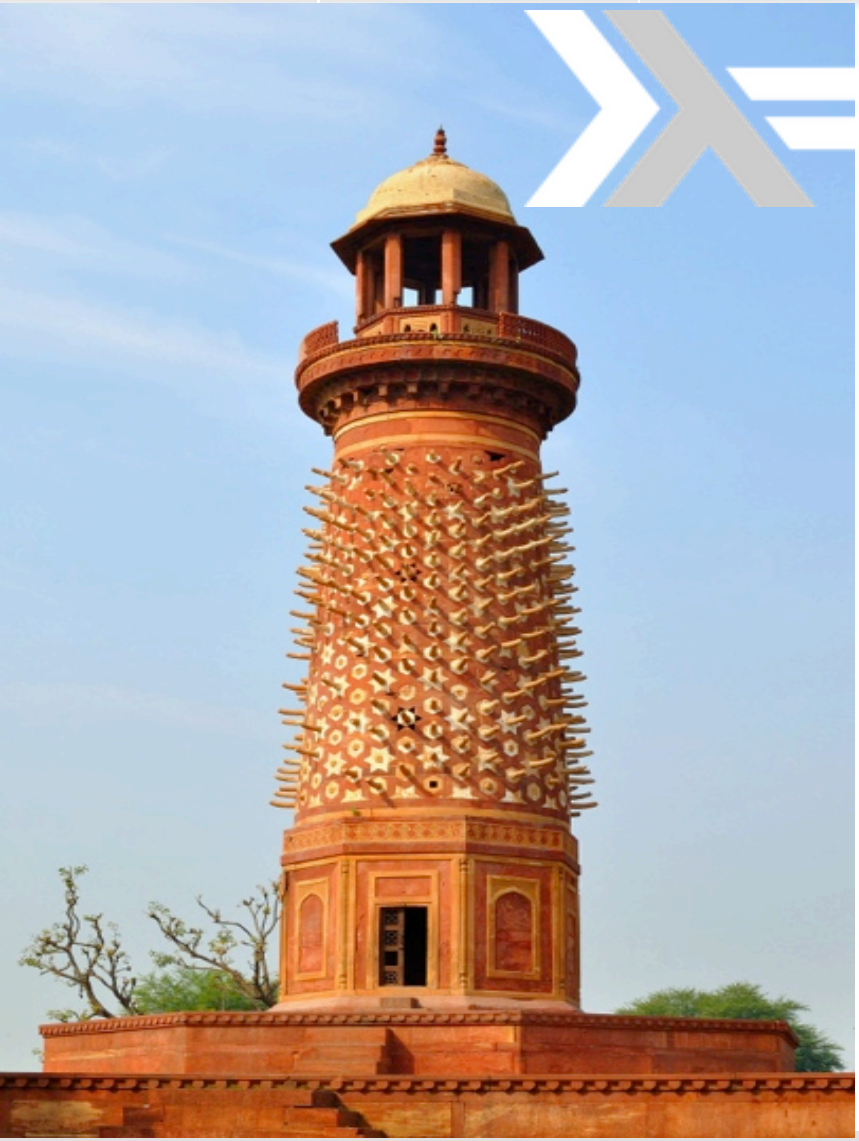
Die Programmiersprache

What lies beneath..

Historien anhand der Standards



Mal zusammengefasst:



Was ist cool an Haskell?

... und wird von Java so nicht geboten...

Verzögerte Auswertung

akquinet

Die Menge der natürlichen Zahlen:

```
n = [1..]
```

Alle geraden Zahlen:

```
g = [ x | x <- [1..], even x ]
```

Was man damit tun kann?

- z.B. Nummerierung von Textzeilen

```
lines :: [String]
numberedLines = zip n lines
   :: [(Integer, String)]
```

Abstrahiert:

- Schwerpunkt auf Beschreibung von Daten und nicht auf Berechnung

Algebraische Datentypen

Aufbau per Konstrukturen

```
data BinTreeT a =
 Tree (BinTreeT a) a (BinTreeT a) |
 Empty
```

Zugriff per Patterns

```
binSearch::(Ord a)=> a->(BinTreeT a)->Bool

binSearch _ Empty = False
binSearch search (Tree left value right)
  | search < value = binSearch search left
  | search > value = binSearch search right
  | search == value = True
```

=> Ermöglicht kurzen, lesbaren Code


Aber, was ist
mit Kapselung?

Mit Modulen!

Konzept:

- Verbergen der Konstruktoren
  => kein Pattern-Matching
  => kein Zugriff auf Interna
- Export von Erzeugungs-funktionen

=> Einfach und effektiv

```
module ADT
( binSearch
, BinTreeT
, createBinTree
)
where
....
```

OO mit Funktionen ohne Daten

akquinet

Typeklassen

- Menge von Funktionen, die es für einen Datentyp geben muss
- unterstützen Mehrfachvererbung
- erlauben Standardimplementierung

```
class Eq a => Ord a where
 compare :: a -> a ->
                    Ordering
  (<) :: a -> a -> Bool
  (>) :: a -> a -> Bool
  … -- Snip
  x <  y =
   case compare x y of
     LT -> True
     _  -> False

instance (Ord a) =>
  Ord (BinTreeT a) where
… -- Snip
```

Eine letztes Wort zu Typen

akquinet

Haskell ist statisch stark typisiert.

D.h.

- jeder Ausdruck hat zur Compilezeit einen geprüften konkreten Typ
- ***keine*** Typfehler zur Laufzeit
- Optimierungsmöglichkeiten, da Typinformation zur Laufzeit nicht mehr benötigt

Und praktisch:

- Einsatz von Typen zur fachlichen Modellierung ermöglicht teilweise Korrektheitsprüfungen bei der Übersetzung

Und nun das Beste!

Referentielle Transparenz

- `a = f b c d e g`
  => `a` wird sich ***nie*** ändern
- Keine Seiteneffekte!
- Wesentlich einfachere Programm- und damit Fehleranalyse

I/O über Monadenkonzept

Weitere Unterschiede zu Java

Übersetzung in Binärcode

- Schnelle Startzeiten, keine externe Laufzeitumgebung
  (Bsp. Hello World: C 2ms; Haskell 3ms; Java 157ms)

System-näher

- Sehr dünne Abstraktion von System
- Orientierung an Posix und Unix
  (=> kein VM-Konzept)

Und, ja, Funktionen als 1st Class Citizen...


Gibt es
auch
Probleme?

The Ugly...

Verzögerte Auswertung

- erzeugt selten schwer auffindbaren Speicherverbrauch

Typinferenz

- Funktioniert gut, erschreckt gerne mit Fehlermeldungen

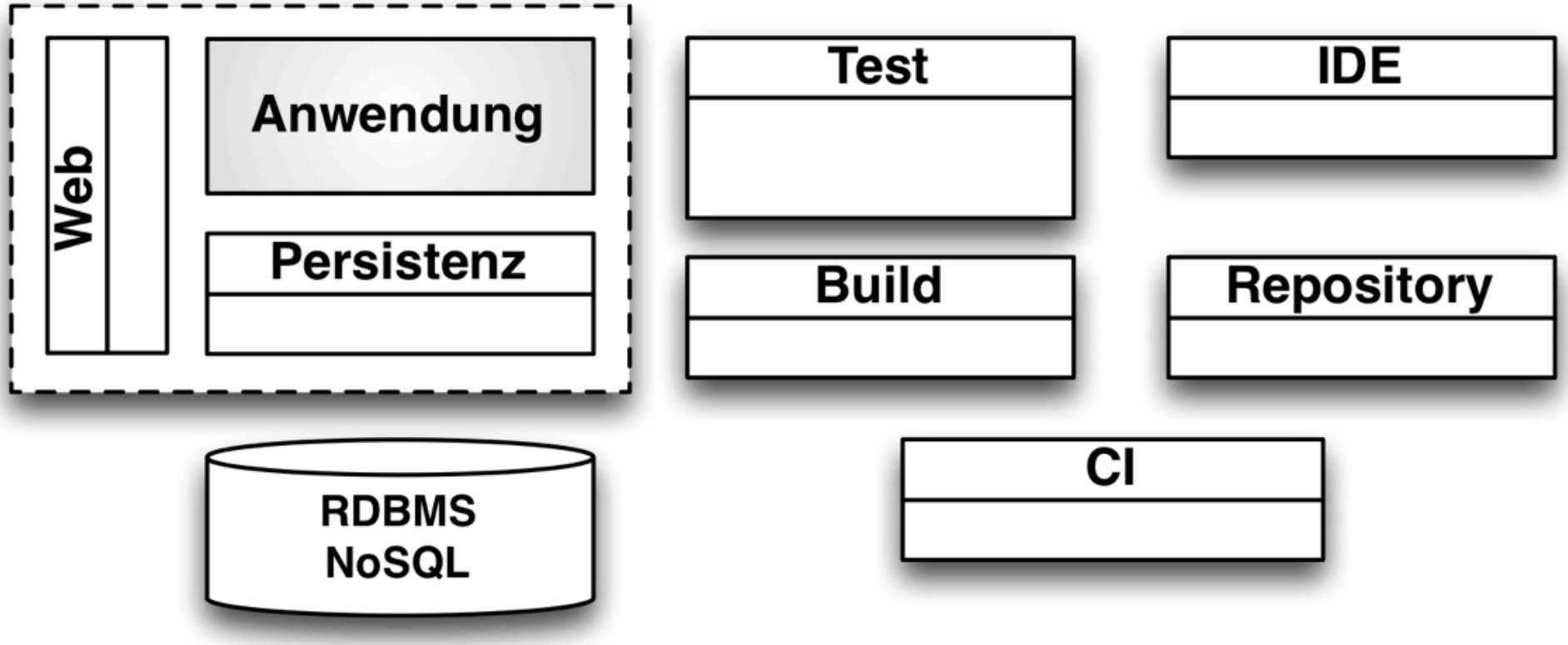
Entwurfsentscheidungen

- waren manchmal unklug
- Bsp: String als [Char]
  Ergebnis:
  - zwei verbreitete Alternativen:
    Text, ByteString
  - Compiler-Erweiterung zum einfacheren Handling

Frameworks Werkzeuge

Im Schweinsgalopp...

Klassifikation von Frameworks



Yesod

typsicher, REST-basiert und effizient

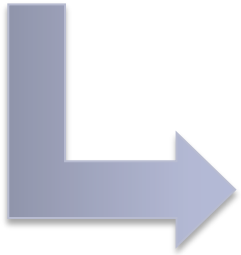
Die Oberfläche

akquinet

List of Users

Id	First Name	Last Name	Actions
user-100	John_100 Hallo	Doe_100	
user-101	John_101	Doe_101	
user-102	John_102	Doe_102	
user-103	John_103	Doe_103	
user-104	John_104	Doe_104	

Create New User



```
<h1> List of Users

<table class=datatable>
  <tr><th>Id
      <th> First Name
      <th> Last Name
      <th> Actions
  $forall User ident _ firstName lastName <- users
    <tr><td>#{ident}
        <td>
          $maybe first <- firstName
            #{first}
        <td>
          $maybe last <- lastName
            #{last}
        <td><a href=@{UserR ident}>
              <img alt=Edit
                 src=@{StaticR img_editblue_png}
                 class="tableaction"
               >
  <tr> <!-- SNIP  -->
<a href=@{NewUserR}>
  Create New User
```

Verknüpfung zu Haskell

routes.txt

```
/users UsersR GET
/users/#PageCounter UsersPageR GET
```

```haskell
getUsersR :: Handler RepHtml
getUsersR = redirect $ UsersPageR $ PageCounter 0

getUsersPageR :: PageCounter -> Handler RepHtml
getUsersPageR (PageCounter page) = do
    userEntitiesAndKeys <- runDB $ selectList []
                                   [ Asc UserIdent
                                   , OffsetBy $ pagesize * page
                                   , LimitTo pagesize
                                   ]
    defaultLayout $ do
        let users = map (\ (Entity _ user) -> user) userEntitiesAndKeys
            start = PageCounter 0
            next =  PageCounter $ page + 1
            previous = PageCounter $ max 0 (page -1)
        setTitle "List of Users"
        $(widgetFile "users")
```

Weitere Yesod-Features

Schneller Entwicklungsmodus

- Automatisches inkrementelle Übersetzung

REST mit JSON

- Gut geeignet für Single-Page-Anwendungen

Internationalisierung

Authentisierung

- mit Standardmodulen für OpenID, BrowserID, Oauth

Tests

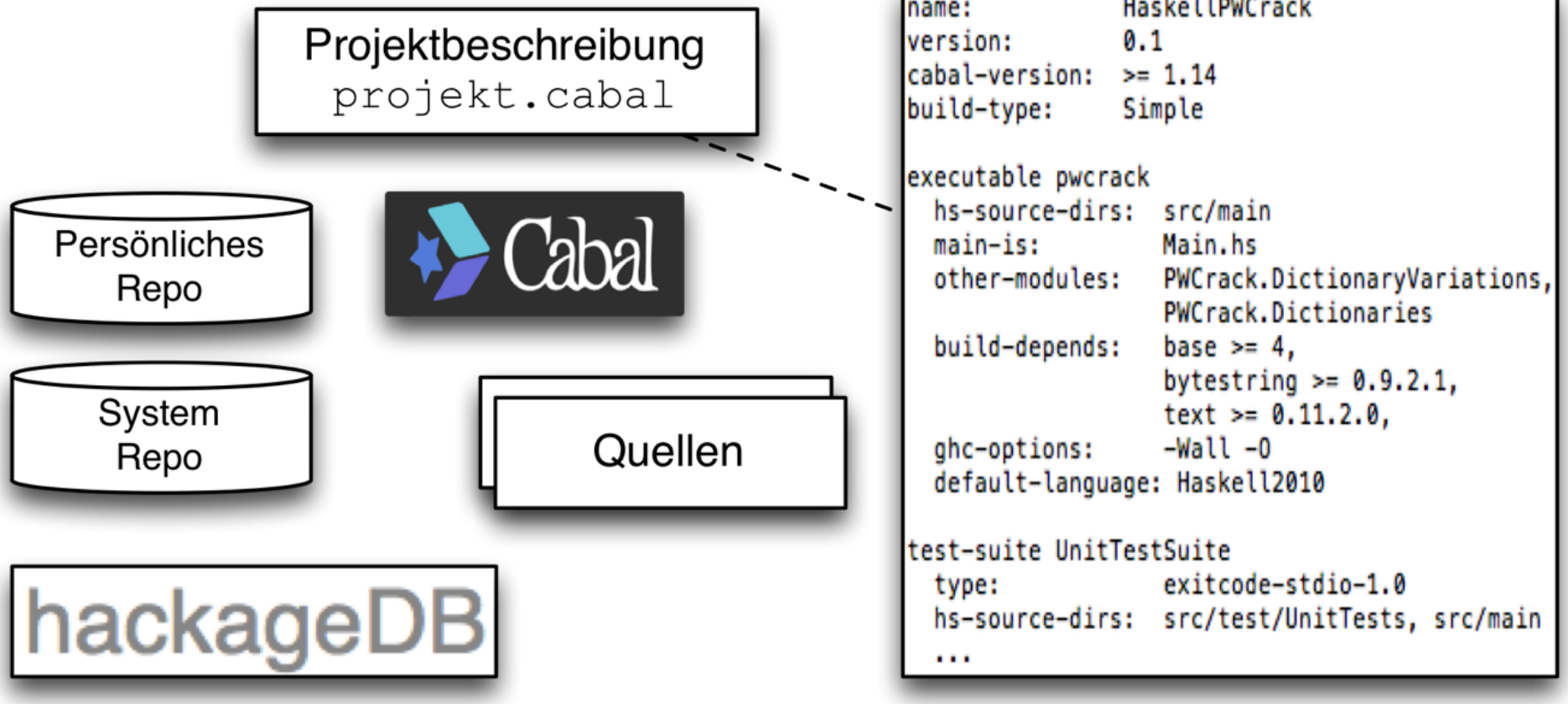
akquinet

BDD mit HSpec

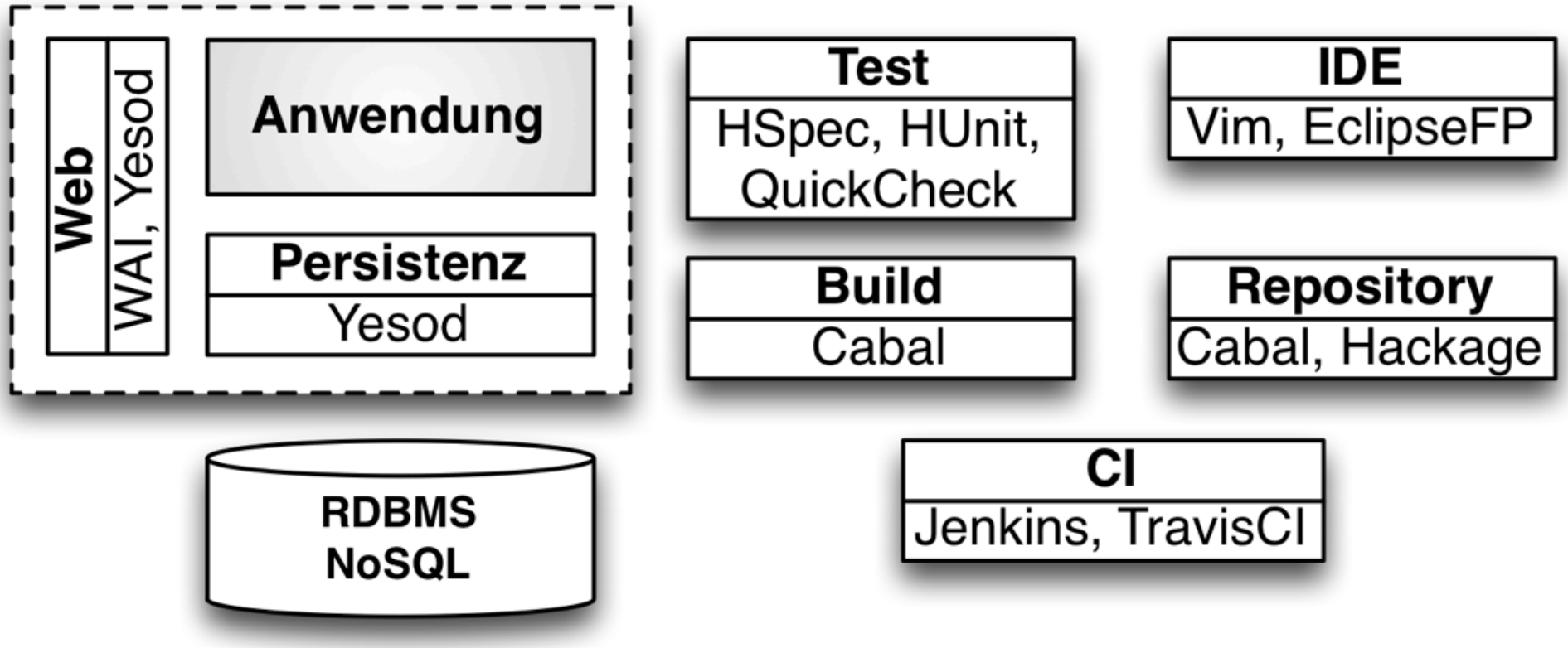
```
createBinTreeSpec :: Spec
createBinTreeSpec =
  describe "createBinTree" $ do
    it "creates an empty tree from an empty list" $
      createBinTree ([]::String) `shouldBe` Empty
    it "creates a filled tree from a list" $
      property ((\strs ->
        let tree = createBinTree strs
            fstFailure = find (not . flip binSearch tree) strs
        in isNothing fstFailure
                ) :: String -> Bool)
```

Automatische Testdatenerzeugung mit QuickCheck

Cabal, Hackage



Zusammengefasst




Gibt es
auch
Probleme?

The Ugly ....

Cabal, Hackage

- DLL-Hell^2, schnell ein Versions-Chaos
- IMHO: Es fehlt an Standard-plattformen

Yesod

- Cool, aber kein Standard

Es fehlt:

- JMS, **XA**

Resümee

... And the winner is ...

Resümee

Vergleich der Programmiersprache

- Haskell deutlich klarer und weiter trotz gewachsener Schrunden
- Java punktet mit Standardbibliotheken

Vergleich der Frameworks

- Java führt, insb. im Enterprise-Sektor
- Haskell hat genug für autonome Webanwendungen

Sonstiges

- #Java-Entwickler > #Haskell-Entwickler
- Wesentlich mehr Investitionen im Java-Bereich

Your Choice!

Historien

Haskell

- 1990: Sprachspezifikation 1.0 nach 3 Jahren Komiteearbeit
  Ziel: Konsolidierung der funktionalen Programmiersprachen
  benannt nach Haskell Curry
- 1998: Haskell-98
  „Aufräumen“ von Haskell 1.4, Einführung von Standardbibliotheken
- 2010: Haskell-2010
  Erweiterung/Aufräumen der Sprache; neue Versionen im Jahresrythmus um Sprache zu entwickeln

Java

- 1995: Version 1.0a
- 1997: JDK 1.1
  JDBC, RMI
- 1998: JDK 1.2
  Swing, Collections
- 1999: Java EE 1.2
  EJB, JMS, JTA, JSP
- 2000: JDK 1.3
  HotSpot, JNDI
- 2001: Java EE 1.3
  JAAS
- 2002: JDK 1.4
  neue Bibliotheken
- 2003: Java EE 1.4
  JSF, JMX
- 2005: JDK 5.0
  einige Spracherweiterungen (Generics, Autoboxing ...)
- 2006: JDK 6
  Aktualisierungen, Verbesserungen
- 2006: Java EE 5
  Anpassung an Spracherweiterungen, JPA
- 2009: Java EE 6
  CDI
- 2011: JDK 7
  kleine Spracherweiterungen