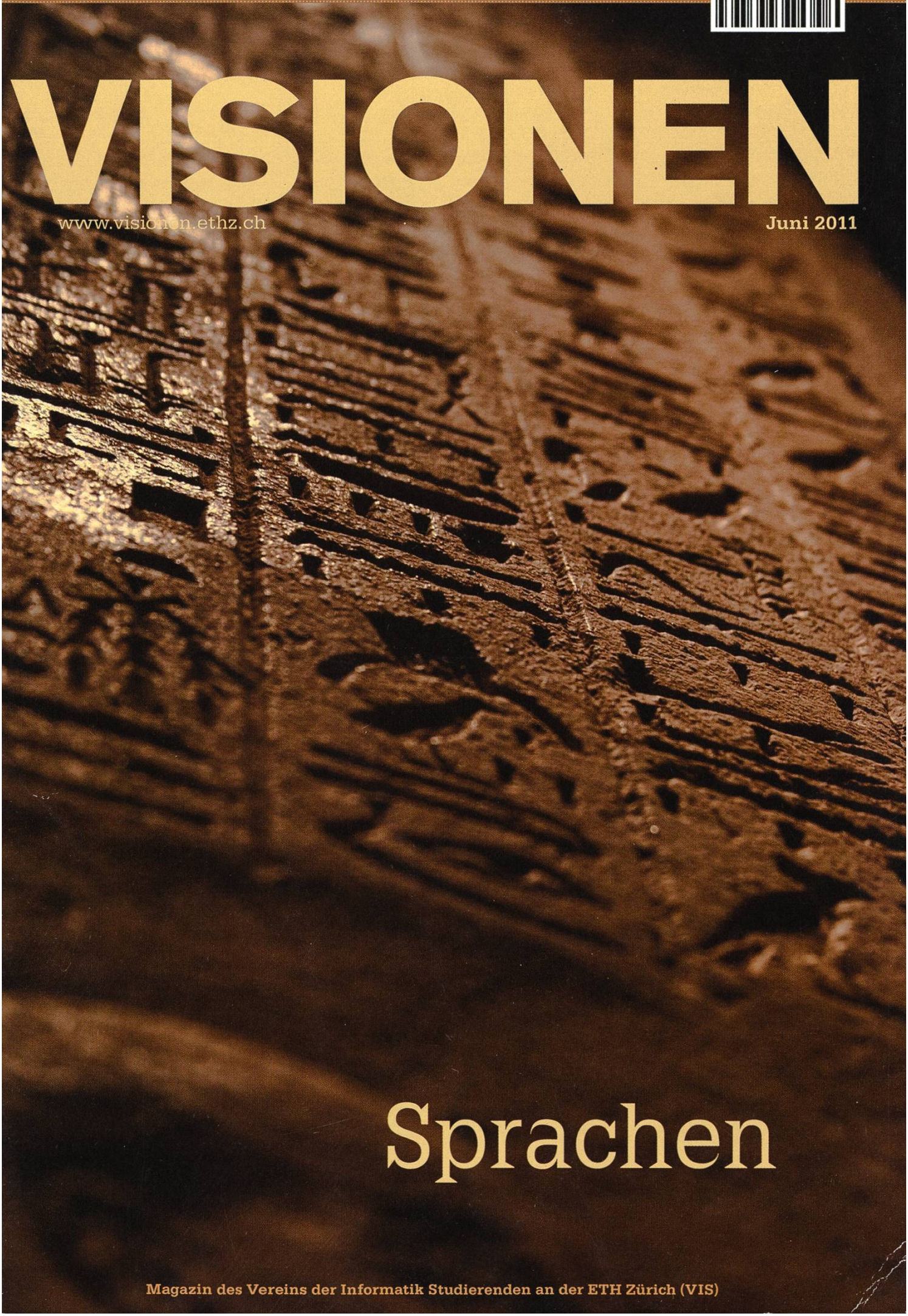
schwer klassifiziert +++ Archäologie: Neue Erkenntnisse—alte Ägypter kannten COBOL +++ Wetter: Gesichter drohen zu schmel… …weisstreibe

VISIONEN

www.visionen.ethz.ch

Juni 2011



Sprachen

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

open
systems

Open Systems gehört mit seinen Mission Control Security Services im Bereich IT-Sicherheit zu den europaweit anerkannten Anbietern. Wir arbeiten von Zürich und Sydney aus in einem dynamischen Umfeld in über 120 Ländern. Bei uns kannst Du Dein Wissen in einem jungen Team in die Praxis umsetzen und rasch Verantwortung übernehmen. Infos über Einstiegs- und Karrieremöglichkeiten sowie Videos findest Du auf unserer Website.
www.open.ch

Editorial

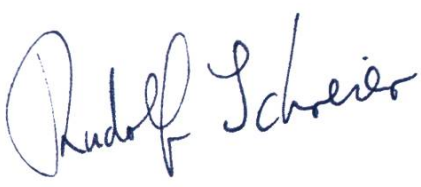
RUDOLF SCHREIER — SPRICHT AUCH DEINE SPRACHE

Hummel Hummel, ni-hao und konnichiwa minna-san! Wer nur Bahnhof versteht, dem kann in dieser Ausgabe geholfen werden; zwar nicht beim Lernen von Deutsch, Chinesisch, oder Japanisch (für die ersteren Beiden kann ich den geneigten Leser auf Ausgabe 6/2009 verweisen), dafür aber beim Überblick über die gleichermassen weiten Welten der natürlichen und konstruierten Sprachen. Und wie das alte Programmierersprichwort sagt: Lerne mindestens eine neue Sprache pro Jahr – warum sollte das nicht mal eine ausgefallenere sein.

In internen News kann ich eine neue Kolumne ankündigen, die euch die verschiedenen Forschungsbereiche im Bereich der Informatik an der ETH an konkreten Auszügen aus Work-in-Progress oder neuesten Publikationen nahebringen wird. Wenn ihr im Bereich der Forschung tätig seid und der nächsten Generation von Studierenden euer Fachgebiet näherbringen wollt, kontaktiert doch die Redaktion unter chefredaktor@vis.ethz.ch.

In dieser letzten Ausgabe des Sommersemesters freue ich mich, euch allerseits eine frohe vorlesungsfreie Zeit, und allen Studierenden, vor allem aber den Basisprüflingen, ein gutes Gelingen bei den Prüfungen zu wünschen.

Euer Chefredakteur,
Rudolf Schreier (rms)



Inhalt

Sprache

Berichte

Studium

Spass

Go:
A New Systems Language?

SIMON GERBER — SYSTEMS GUY

While you may have heard of Go[1] (the relatively new programming language designed by Google), most of you probably have not yet used it. Here I will try to demonstrate what makes Go unique.

In this article I will try to highlight Go's most interesting features using a fairly small example program (not "Hello, world!", sorry!) which is a systems tool. I am reusing an idea from the Networking/OS lecture in FS08: a server that takes timer requests over TCP and responds after the given time.

Tool-chain setup

To get an impression of the language, a compiler tool-chain is necessary. You can find detailed instructions on Go's homepage[2]. Personally I feel that the Go tools are named curiously: you have `6g/8g/5g` and `6l/8l/5l` (the compiler and linker respectively) for amd64 (prefixed with a 6), x86 (8) and ARM (5). After playing around a bit I wrote a quick script which allows me to compile a Go program using only one command[3]. An alternative would be the `gccgo` front-end for the GNU compiler collection.

The timer server

```
package main
```

Go is structured in packages. These packages work like namespaces in C# but are not hierarchical. Each Go program must have a package "`main`" containing a function `main` which is the entry point of the program.

```
import (
  "net"
  "bufio"
  "strconv"
  "fmt"
  "flag"
  "os"
  "time"
  "strings"
)
```

"`import`" is used to import other packages. Here we are importing various useful packages, such as `net` for networking, `fmt` for `printf` and consorts, `flag` for easy command-line parsing and various others. All these packages belong to Go's standard library and should work on all supported platforms. Fairly good documentation for Go's standard packages can be found at [4]. →

```
var port = flag.Int("p", 10000,
  "port to listen on")
```

Using snippets like the above, we can easily define command-line parameters (here "p") with a default value (10000) and a description that is also used by the Go runtime to print a nice usage message.

These flags can then be specified on the command line prefixed with either one or two hyphens and with an equals sign or a space between flag and value. There are predefined flag types for integers, floats, booleans and strings but other custom flag types can be easily defined.

Functions

Function calls are qualified with the package the function is defined in or the variable on which the function is invoked. Function visibility in Go is easy: All functions that start with an upper-case letter are visible outside a package, all others are local to the package.

```
func main() {
  flag.Parse()
  server, err :=
    net.Listen("tcp", ":" +
    strconv.Itoa(*port))
  if server == nil {
    panic("net.Listen: " +
      err.String())
  }
  conns := clientConns(server)
  for {
    go handleConn(<-conns)
  }
}
```

This is the main function for our program. First off, it calls `flag.Parse()` which does all the command-line parsing magic. After that, a TCP connection is opened for listening. Go supports multiple return values, which—among other things—makes error handling easier. Thus you will often see code in the form

```
v, err := someCall()
```

followed by

```
if v == nil { // handle error }
```

The `strconv` package provides all sorts of conversion routines from and to strings. Here we're using `Itoa` which converts a integer to a string.

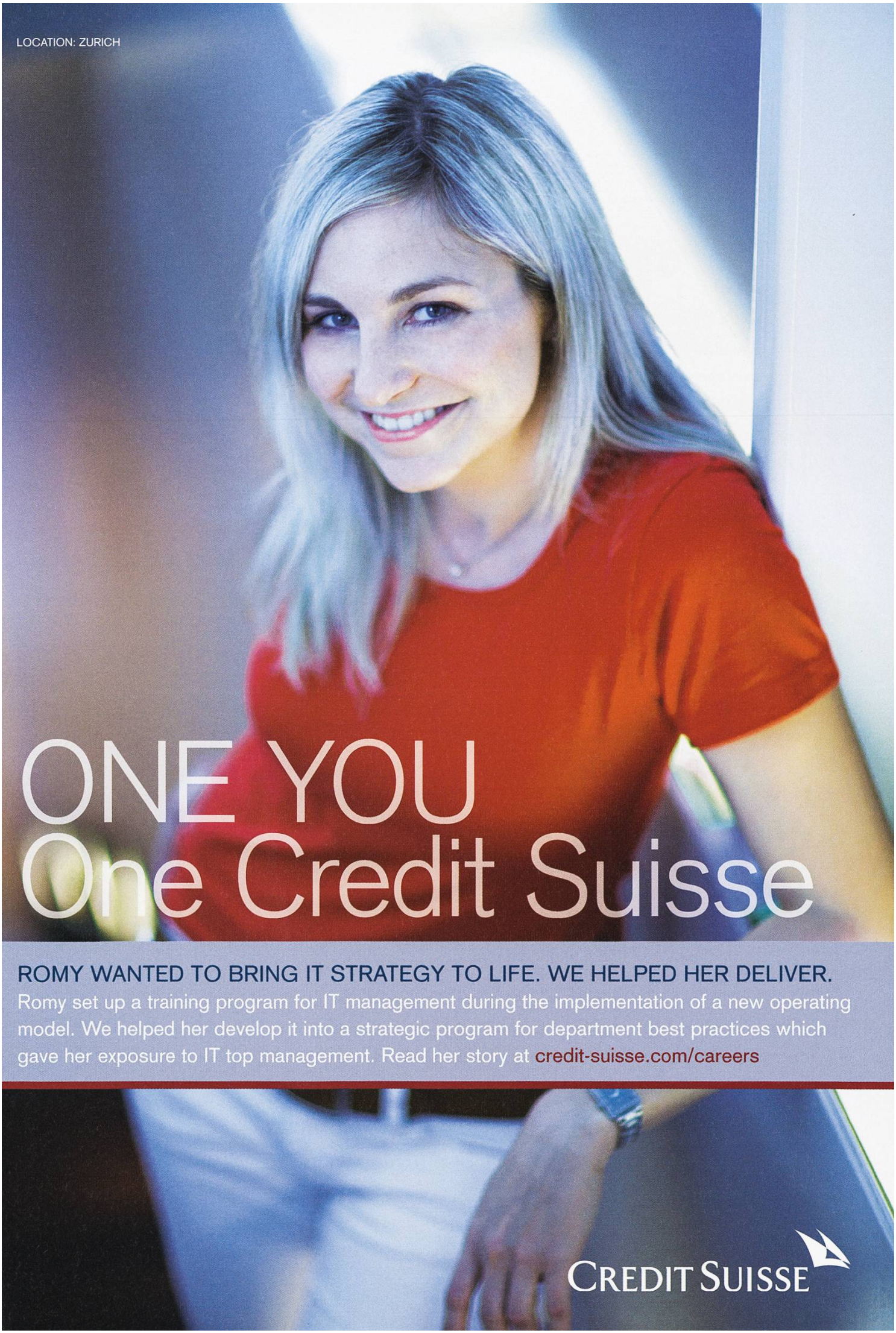
Another point of interest is the `:=` operator, which simultaneously initializes and declares the left-hand side.

On the last four lines of the main function we see Go's most intriguing concepts: Channels and goroutine invocation.

Channels

Go is designed for parallel computation and borrows the concept of a channel from Hoare's CSP (Communicating Sequential Processes[5]). A channel is basically a type-safe Unix pipe, allowing one process (the producer) to write to it and another process (the consumer) to read from it.

Go Channels can be buffered and unbuffered: An unbuffered channel is synchronous, that is reading from a channel blocks until an item is available and writing to a channel blocks until someone is ready to read from it. A buffered channel has a backing buffer and allows asynchronous reads as long as there are items in the buffer and asynchronous writes as long as there is some free space in the buffer. ➔


LOCATION: ZURICH
ONE YOU
One Credit Suisse
ROMY WANTED TO BRING IT STRATEGY TO LIFE. WE HELPED HER DELIVER.
Romy set up a training program for IT management during the implementation of a new operating model. We helped her develop it into a strategic program for department best practices which gave her exposure to IT top management. Read her story at credit-suisse.com/careers
CREDIT SUISSE

Goroutines

Go's main form of concurrent execution units are goroutines, which are small lightweight threads. Starting a goroutine usually costs little more than allocating a stack for it to run on. Goroutines are executed in parallel with other goroutines including their caller. While they do not necessarily run in different threads, mostly a group of goroutines is multiplexed onto a number of threads, to be able to handle blocking events easily.

Syntax and language constructs

Syntax-wise the following points stand out (especially coming from the C world):

- Conditions in conditional statements are not enclosed by parentheses
- Every loop body and `if` block must be enclosed by braces.
- The only loop form is the `for`-loop, which can be used as infinite loop (as above), as "normal" `for`-loop

  ```
  for  idx := 0; idx < 10;
    idx++ { }
  ```

  and as `while`-loop

  ```
  for condition() { }
  ```

- The return value(s) of a function are specified after the formal parameter list and look exactly the same syntactically (you can omit the parentheses when there is only a single unnamed return value and completely omit the return value for a function returning `void`).
- Variable and parameter types are specified after the names and multiple variables of the same type can be separated by commas and share the type definition.
- Pointer and array types have the asterisk/brackets before the type name instead of after.
- The keywords `import`, `var`, `const` and `type` introduce declarations, which can be grouped with parentheses.
- Variable types are inferred where possible.
- Usually no semicolons are necessary.

```
func clientConns
  (listener net.Listener)
  (ch chan net.Conn) {
  ch = make(chan net.Conn)
  go func() {
    for {
      client, err :=
        listener.Accept()
      if client == nil {
        fmt.Printf(
          "listener.Accept: " +
          err.String())
        continue
      }
      fmt.Printf("accepted %v\n",
        client.RemoteAddr())
      ch <- client
    }
  }()
  return ch
}
```

This function is called by the `main` function to handle all incoming client connections. First off, a new channel of connections is created. Then a goroutine which accepts client connections and puts them into the channel is started.

Here you can see the use of a function literal as a goroutine. In Go, function literals act as closures and therefore the newly created channel value remains in scope after `clientConns()` returns.

Note the parentheses after the closing brace of the anonymous function: you have to call the function literal in the goroutine invocation.

Data Allocation

Go has two different allocation primitives: `new()` and `make()`. They apply to different types and do different things:

`new(T)` is a built-in function which allocates zeroed storage for a new object of type `T` and returns its address (a value of type `*T`).

`make(T, args)` is another built-in function which serves a different purpose than `new(T)`. It can only create slices, maps and channels and it returns an initialized (not zero) value of type `T`, not `*T`. The reason for this distinction is that all these types are references to data structures that must be initialized to be usable. For example a slice is a triple of values (pointer to data inside an array, length, capacity) and until those three items are initialized the slice is nil.

```
func handleConn
  (client net.Conn) {
  b := bufio.NewReadWriter
    (bufio.NewReader(client),
    bufio.NewWriter(client))
  for {
    fmt.Printf
      ("duration request\n")
    b.WriteString
      ("Enter duration in
       seconds: ")
    b.Flush()
    line, err :=
      b.ReadBytes('\n')
    if err != nil {
      // EOF, or worse
      break
    }
    timer(line, b)
  }
}
```

In the client handler we first encapsulate the client connection using a buffered reader and writer. This gives access to methods to read from and write to the connection without constantly having to manipulate byte arrays. Then we run an infinite loop that terminates when the client sends EOF or worse and which repeatedly asks for a duration and runs a timer with the requested duration. ➔

```
func timer(line []byte,
  b *bufio.ReadWriter) {
  defer un(trace("timer"))
  ns, err := getDurationNs(line)
  if err != nil {
    b.WriteString(
      "Error, aborting: " +
    err.String() + "\n")
```

```
    return
  }
  err = time.Sleep(ns)
  if err != nil {
    b.WriteString("Error: " +
      err.String())
    return
  }
  b.WriteString("BEEP!\n")
}
```

This is the code which actually runs the timer. First we add a nice tracing mechanism to our code for this function: Using Gos `defer` keyword, we specify a function `un(s string)` to run on exit of the function `timer()`. As the arguments of a deferred function are evaluated when the defer statement executes, we execute the function `trace(s string) string` when entering the function `timer()`.

After that we convert the line we got from the client into an integer representing the duration in nanoseconds and use that value in the call to `time.Sleep(ns int64)`. Finally, we send "BEEP!" to the client and return.

```
func getDurationNs(line []byte)
  (ns int64, err os.Error) {
  time, err :=
    strconv.Atoi64(
    strings.Trim(string(line),
    " \t\n\v"))
  if err != nil {
    return 0, err
  }
  ns = time*1000*1000*1000
  return
}
```

Nothing really spectacular happens in this function; the input byte array is converted to a string, leading and trailing white-space is removed using `strings.Trim()` and we try to convert the cleaned string to a 64-bit integer. If this succeeds we convert the input (which is assumed to be in seconds) to nanoseconds and return. On failure we pass on the error from `Atoi64`.

```
func trace(s string) string {
  fmt.Printf("entering %s()\n", s)
  return s
}

func un(s string) {
  fmt.Printf("leaving %s()\n", s)
}
```

You can download the full code at http://pages.vis.ethz.ch/visionen/2011-3/go/.

Types

Go has the familiar `int` and `uint` types which represent values of an appropriate size for a machine word. It has also sized integer types such as `int8` and `uint64`. There is a byte synonym for `uint8` which is the element type for strings. Go also has `float32`, `float64` as well as `complex64` and `complex128` (two `float32` and `float64` respectively). Also, `string` is a built-in type with immutable values—strings are not just arrays of byte values.

Arrays are mutable value types and include the size of the array as well as its element type, which makes talking about pointers to arrays meaningful (and useful) as opposed to C.

There are also slices which can hold a reference to any array with the same element type as the slice. In short slices have the type `[]T`, where an array has the type `[n]T`. ($n \in \mathbb{N}$).

Custom types can be defined using the `type` keyword which works similar to Cs `typedef`. ➔

«Nur eines ist schöner als gute Software, die komplexe Aufgaben ganz einfach löst: Diese Software zu erfinden. Wir freuen uns auf deine Bewerbung.»

Fabian Laubacher, Software Engineer bei BSI

BSI Business Systems Integration AG
www.bsiag.com/zukunft

bsi
meet your clients

OO and Inheritance

The definition of a custom type in Go contains only the data members. Member functions are implemented using the `func (RT) funcName(...)` syntax which implements the member function `funcName` for the type `RT`.

Go has no type hierarchy but every type can implement an arbitrary number of interface types. An interface type is simply a set of method declarations. A variable of an interface type can store any value of any type with a method set which is a superset of the method set of the interface type. Such a type is said to implement the interface[7].

This design also means that Go has no such things as constructors, there's just the `new(T)` built-in which allocates space for a value of type `T` but does not initialize it. Initialization is handled by convention, modules usually provide a public method `New` with returns an initialized value.

To make use of these features (and many others like function literals) Go requires garbage collection and some other supporting runtime code.

Conclusion and personal feeling

I like Go as a language and can see myself writing network services and other inherently concurrent software in Go in the future. I especially (as a long-time Linux user) like the fact that one of the key concepts of the Unix command line (pipes) has made its way into the Go language in the form of channels.

I also like the way the designers of Go decided to have a C-like language with garbage collection and some basic object-oriented features based around the notion of interface types and the possibility of defining methods on any type instead of going full out for a class-based approach à la C++/Java/C#.

Hello, world!

An article about a programming language is not complete without a "Hello, world!" example, so here it is, regardless of what I said in the beginning.

```
package main
import "fmt"
func main() {
  fmt.Println("Hello, world!")
}
```

✦

Links

[1] http://golang.org/

[2] http://golang.org/doc/install.html

[3] http://pages.vis.ethz.ch/visionen/2011-3/go/goc

[4] http://golang.org/pkg/

[5] http://en.wikipedia.org/wiki/Communicating_sequential_processes

[6] http://golang.org/doc/go_tutorial.html

[7] http://golang.org/doc/go_spec.html#InterfaceType

Documentation

Introductory Tutorial:
http://golang.org/doc/go_tutorial.html

Idioms, advanced features:
http://golang.org/doc/effective_go.html

Scala—Awesomeness Compiles to Bytecode

DANIEL THOMAS — OBVIOUSLY LIKES PIZZA

Our mates at the EPFL sometimes speak incomprehensible stuff ("Comme ça"). Yet when it comes to programming languages, the coolest kid on the block originated there and is out to show the world what a modern language could look like. This article is not intended as tutorial or even as introduction, its purpose is to point out a few mouthwateringly nice features and make the reader run to the computer and find out more.

Everything is an Object

Most currently used languages are object-oriented in some way. This makes it even more astounding that very few of them really stick to the concept. With Scala there are no primitive types, and every value is an object. Even functions are objects and can be assigned to values allowing for functional programming and object orientation to work side by side. A very simple example would be:

```
var aFunc =
  ((s:String) => s.toUpperCase)
println(aFunc("Hello World"))
```

By the way, Scala also does away with the error prone static variables, and lets you define singletons directly by defining them as "object" instead of "class".

Pattern Matching

While Java 7 finally will introduce String objects in switch-case constructs, Scala shows how powerful such a construct can really be. Together with the functional aspects of Scala it is possible to create nice and clean code without huge if constructs, e.g.

```
// objects can be defined directly
// the scala way to do singletons
object Hungry {
  def main(args: Array[String]) {
    var whatAmI = "ding dong"
    matchIt(whatAmI)
  }
  def matchIt(value: Any) {
    value match {
      case i: Int =>
        println("this really is
          an Integer,
          with value: " + i)
      case s: String =>
        println("wow, a String,
          with value: " + s)
      case List( _ , _ , "boing",
        _ , _ ) =>
        println("this is a list,
          with size 5 and the
          middle element is
          boing")
      case _ =>
        println("Really, what on
          earth should I do
          with ")
    }
  }
}
```

➔

Concurrency With Actors

Getting concurrency right with threads is hard. Very hard. So hard that many programmers don't really bother with it, regardless of the current trend to multicore processors. But what are the alternatives? Proposed in 1973, the idea of message driven actors has been around for quite some time and is probably most prominently known from the Erlang programming language. The basic idea is to have Actors (in Scala these are descendants of the `scala.actors.Actor` class) which send and receive messages, thus eliminating the need for shared memory, locks and other error prone constructs. Every actor has a queue of messages and simply works them off one by one, e.g.

```
// by the way, this code
// snippet uses a factory
// method from the libraries to
// create an actor directly
val coffeeMachine = actor {
  loop {
    receive {
      case "make a coffee" =>
        println("Here, enjoy")
      case default =>
        println("I don't know
          what I should do
          for " + default ", I'm
          only a coffee
          machine")
    }
  }
coffeeMachine ! "make a coffee"
coffeeMachine ! "bake a bread"
```

Mais Scala, tu connais XML?

In this case, code says more then words:

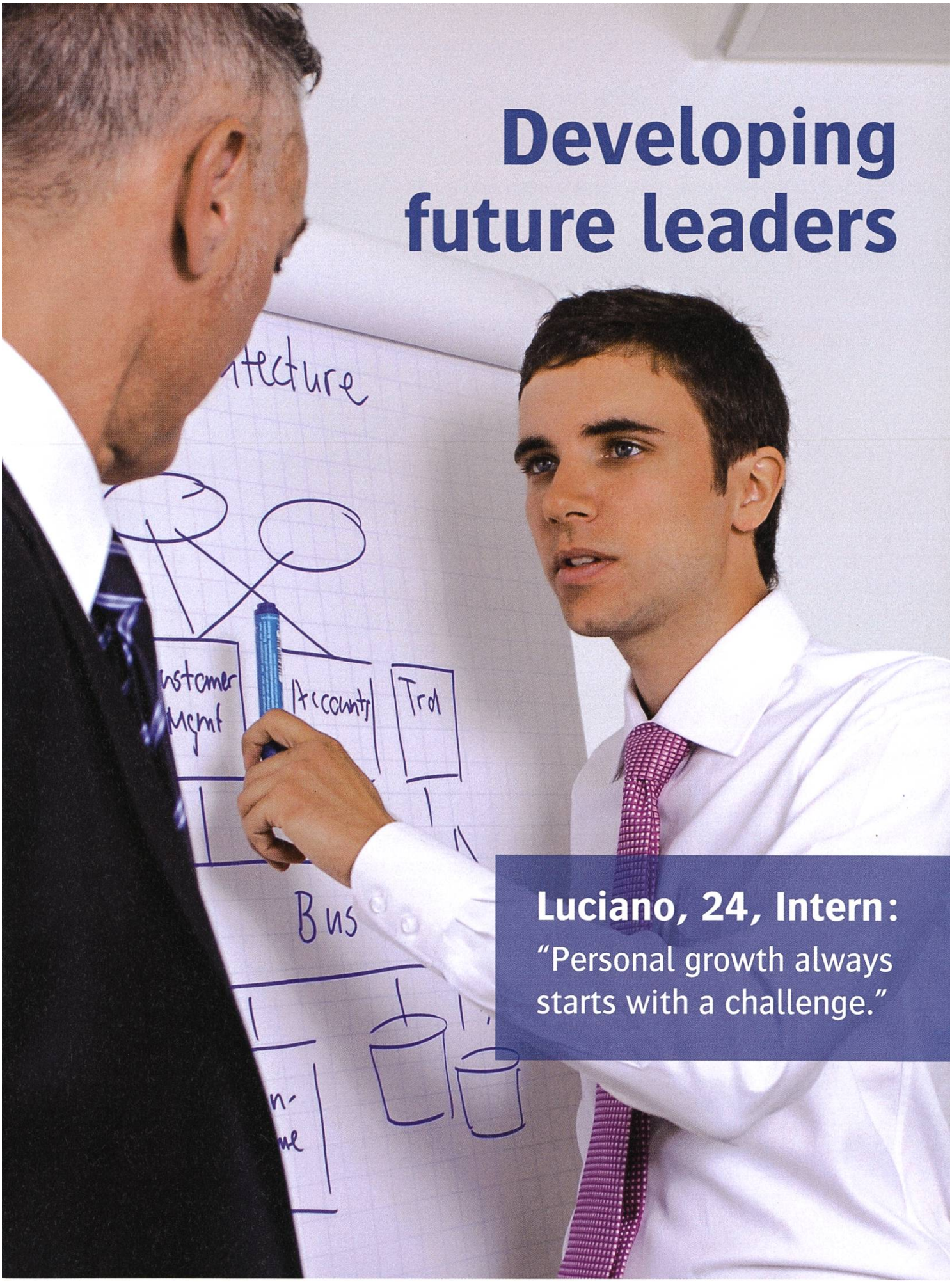
```
// prints:
//  There is a pizza called
//    margarita
//  There is a pizza called
//    prosciutto
object XmlExample {
  def main(args: Array[String]) {
    val aLittleBitOfXml =
<pizzas>
  <pizza><name>margarita</name>
  </pizza>
  <pizza><name>prosciutto</name>
    <topping>ham</topping>
  </pizza>
</pizzas>;
    for (pizza <- (aLittleBitOfXml
      \\ "pizza")) {
      println("There is a pizza
        called " +
        (pizza \ "name").text)
    }
  }
}
```

Cleaner Is Better, Forget ;

Well, this may be a insignificant detail, but ever since learning Eiffel in the Introduction to Programming classes, ending statements with semicolons seems somewhat lame. Scala happily does away with this anachronism.

Who's Your Daddy? Traits Mixin It For A Little More Inheritance

It's not exactly clear if multiple inheritance should be allowed in programming languages or not, but having single inheritance and interfaces as in Java can sometimes be a little constraining. Traits come to rescue by being something between an interface and a class. However, traits are more than just interfaces ➔



We are constantly looking for talented people.
Accept your challenge and apply today.
www.brandis.ch

MARC BRANDIS
STRATEGIC CONSULTING

with method implementation. Through traits, Scala supports mixin class composition with which programmers can reuse all new definitions that are not inherited.

```
class FoodItem(name: String) {
  def properties():
    List[String] = List()
  override def toString() =
    name + " is a" +
    properties.mkString(" ",
    ", ", " ") + "FoodItem"
}

trait Tasty extends FoodItem {
  override def
    properties() =
    super.properties :::
    List("tasty")
  def taste() = println("mmmh,
    this was good")
}

trait Cuttable extends FoodItem
{
  override def
    properties() =
    super.properties :::
    List("cuttable")
  def cut() = println("cut it to
    pieces")
}

// objects can be defined
// directly (the scala way to
// do singletons)
object Hungry {
  def main(args: Array[String])
  {
    val pizza = new
      FoodItem("Pizza") with
      Tasty with Cuttable
    println(pizza) // It's a
     // tasty, cuttable FoodItem
    pizza.cut() // mmh, this was
                // good
    pizza.taste() // cut to
                  // pieces
  }
}
```

Static Typing With Type Inference

This is a question of taste, but personally I'm happy that Scala doesn't go down the possibly error prone road of dynamic typing as seen in Ruby or Python. However, Scala does provide type inference that works surprisingly well, at least for variables with simple types the type can be omitted (in any case, the compiler will complain if type inference failed for a value).

Tuples

Ok, most of us learnt, or are still learning Eiffel. Talking with fellow students one can hear very poignant feelings about this language. Personally, I actually liked it, and one little detail that I have often missed sorely in many other languages are tuples. Sometimes a function just logically should return more than one value. And without tuples one often has to resort to inelegant tricks to do the same. Well, you've guessed it, Scala allows for tuples, and here is a totally senseless example to show how they are used (I guess I've just run out of good examples for today).

```
object SenseLessExample {
  def main(args: Array[String]) {
    var twoValues =
      ("Rock It", 10)
    twoValues =
      mustReturnTwoValues();
    println(twoValues._1 + ","
      + twoValues._2)
  }

  def mustReturnTwoValues() :
    Tuple2[String, Int] = {
    return ("First Value", 300)
  }
}
```

«I am currently doing my Master Thesis in Java and feeling quite 1990s about it.»

Overloading Operators

I'm not sure if one should view operator overloading as really fun feature or as huge source of nasty bugs and irritations. However, Scala was written with DSLs in mind and there operator overloading just makes sense. In addition to simple overloading of operators, Scala allows the use of methods that have a single parameter as infix operators. Just look at the cool use of the method bigger, were already on our way to a sort of DSL.

```
class Complex(val real:Int,
  val imaginary:Int) {
  def +(operand:Complex):Complex
    = {
    new Complex(real +
    operand.real, imaginary
    + operand.imaginary)
  }

  override def toString() = {
    real + (if (imaginary < 0)
      "" else "+") +
      imaginary + "i"
  }

  def bigger(that: Complex) :
    Boolean = {
    if (that.real*that.real +
      that.imaginary*
      that.imaginary <
      this.real*this.real +
      this.imaginary*
      this.imaginary) {
      return true
    } else {
      return false
    }
  }
}
```

```
object Complexity {
  def
    main(args:Array[String]) {
    var firstNumber =
       new Complex(100, 20)
    var secondNumber =
      new Complex(200, 10)
    var thirdNumber =
      firstNumber +
      secondNumber
    println(thirdNumber)
    if (firstNumber bigger
      secondNumber) {
      println("First number
        is bigger"
    } else {
      println("Second
        number is bigger")
    }
  }
}
```

Conclusion

Programming in Scala is fun! As I am currently doing my Master Thesis in Java and feeling quite 1990s about it, it's nice to see that there are alternatives out there that really add new stuff. ✦

Artistic Computing with Piet and L-Systems

DER-YEUAN YU — PROGRAMS AND ART SITTING IN A TREE...

While we can all appreciate the art of computer programming so vividly illustrated by Knuth, it has never been easy to impress your random cousin with those lines of code. Why don't we at least try to make programs look nicer with our innate artistic imaginations?

Painting Programs with Piet

Piet[1], developed by Dr. David Morgan-Mar and inspired by Dutch painter Piet Mondrian, is a programming language where operations of a program are determined by the colours in an image. The name Mondrian was unfortunately already used as a name of a scripting language developed at Utrecht University.

In Piet, each predefined block of pixels with a single colour is defined as a codel, which is the atomic element of a program. One could of course use a single pixel as a codel, but this wouldn't make a nice big painting that we can hang on the wall now would it? The basic element is the colour block, which is a contiguous block of codels with the same colour. A codel can be one of 20 colours, including white and black. White codels, in addition to any other undefined colours, are essentially no-ops that can we can use to customize the painting. Black codels, as well as the edges of the program image are used as borders that redirect program flow. The remaining 18 colours span a 3 by 6 table where they are cyclically related as in Figure 1.

The interpreter maintains a stack and reads instructions in the painting starting from the upper-left-most codel, proceeding in the direction defined by the DP (direction pointer), initially set to the right. An instruction in the painting is defined by 1) an operand, which is the number of codels in the current colour block, and 2) an operator, which is defined by the colour transition to the next colour block.

Traversal to the next colour block is defined with the help of the CC (codel chooser), initially set to the left. The next codel to enter is chosen as the codel that is adjacent, in the direction of the DP, to the codel in the current colour block that is farthest from the current codel, in the direction of the CC, which shares the same edge of the current colour block that is farthest in the direction of the DP. Upon failure of codel entry, caused by attempting to enter black-coloured ➔

Figure 1. Colour table for Piet

light red	light yellow	light green	light cyan	light blue	light magenta
normal red	normal yellow	normal green	normal cyan	normal blue	normal magenta
dark red	dark yellow	dark green	dark cyan	dark blue	dark magenta

Super computing systems

Weltenretter?

Zugegeben die Welt konnten wir bis dato nicht retten, wohl aber verbessern und zwar im Bereich unseres Fachwissens, der Computertechnologie. Hier sind wir zu hause und verändern dank innovativem Querdenken festgefahrene Strukturen, loten das Spektrum der Möglichkeiten aus und mischen Innovation und Technologie zu neuen marktfähigen Produkten.

Egal wie jung oder alt du bist, wenn du Innovation als Herausforderung und Leidenschaft definierst, dann bieten wir dir bei uns im Team tolle Einstiegsmöglichkeiten. Willkommen in der Welt des innovativen Querdenkens und professionellen Umsetzens.

Supercomputing Systems AG • Technoparkstrasse 1 • 8005 Zürich
hrm@scs.ch • Telefon 043 456 16 60 • www.scs.ch

codels or crossing the image boundary, another attempt is made after alternately mirroring the CC or rotating the DP clockwise, and if all possible directions fail the program terminates. Now if I haven't lost you with all that codel talk, let's take a look at Figure 2, an example of pushing 1 and 2 onto the stack then adding them up. Note that pushing a value onto the stack is represented by transitioning to a colour that is one level darker, and addition of two values in the stack is represented by transitioning into a colour that is 1 cyclic step to the right.

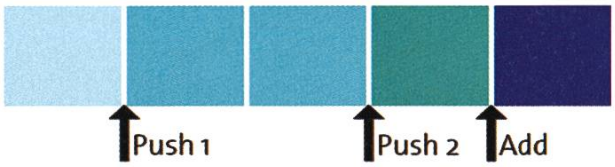
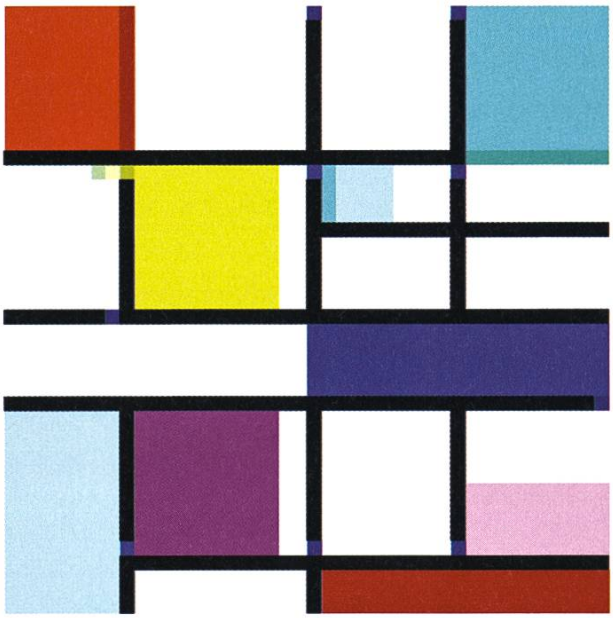


Figure 2. Example code for Piet

The interested reader is encouraged to find the complete instruction set on the reference website, but let us now just marvel at how artistic programs can look. I took the pleasant time of a whole day and painted the program in Figure 3 that prints "VIS."

Figure 3. A Piet program that prints "VIS"



Richard Mitton painted Figure 4[2] that literally computes and prints Pi from dividing the area of a circle by the square of its radius. Of course one could arbitrarily increase the accuracy by making it bigger.

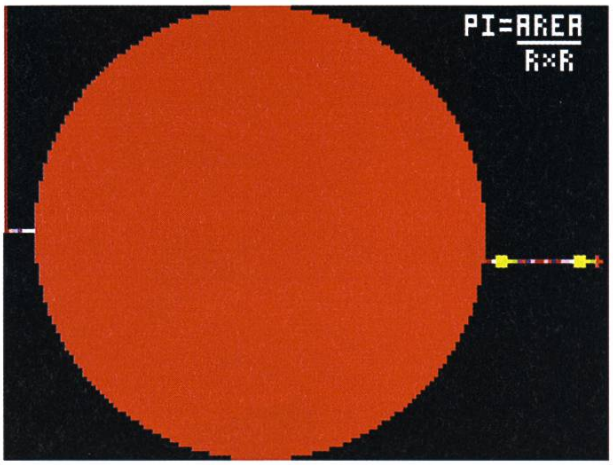


Figure 4. A Piet program that calculates and prints Pi

Programming Paintings with L-Systems

In Piet, language is manifested from structured art, whereas in computer graphics, art can be generated from structured language. The L-system[3], coincidentally introduced also at Utrecht University by theoretical biologist Aristid Lindenmayer, is a formal language that specifies the recursive productions of symbols. Suppose we have the following conditions.

Variables: F (draw forward)
Constants: + (turn right 60°), - (turn left 60°)
Initial State: F++F++F
Production rule: F → F-F++F-F

Pick any initial point and direction on a paper, and we can recursively generate Koch snowflakes as shown in Figure 5[4].

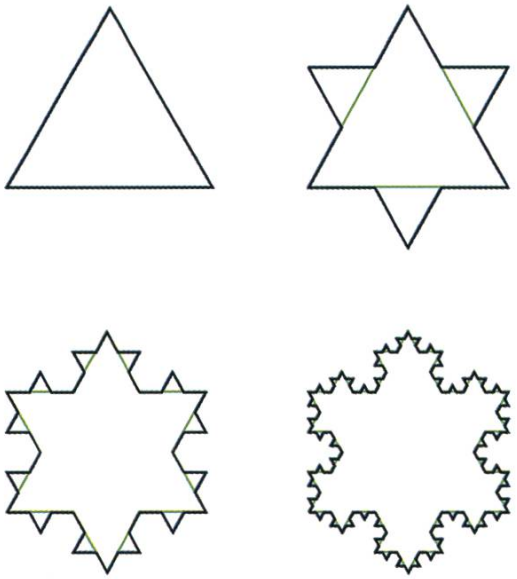


Figure 5. Koch snowflakes

L-systems are widely used to generate fractal structures and can even grow trees (left of Figure 6[5])! We can even add randomness to select from numerous production rules and generate different structures of the same species, as well as apply gravitational effects to long branches (right of Figure 6).

Outlook

When linking art with programs, one would want to tread lightly when mentioning only a specific type of art. Indeed, there is even the Shakespeare Programming Language (SPL)[6] where programs can look like Shakespeare's plays with Romeo and Juliet crying at each other to establish a secret key. On the other hand, if you are a food advocate, you might want to check out Chef[7], where programs are ever so appetizing when they look like cooking recipes.

Personally, Piet reminds me of the punched cards used by our predecessors, and one might fancy the idea that such long scrolls of patterned holes could also be a form of art. Anyway, Piet, SPL and Chef are just the tip of an iceberg if you are up for an escape into the world of esoteric programming languages. Perhaps in the future we won't be surprised if someone submits a programming assignment by singing a song. ✦

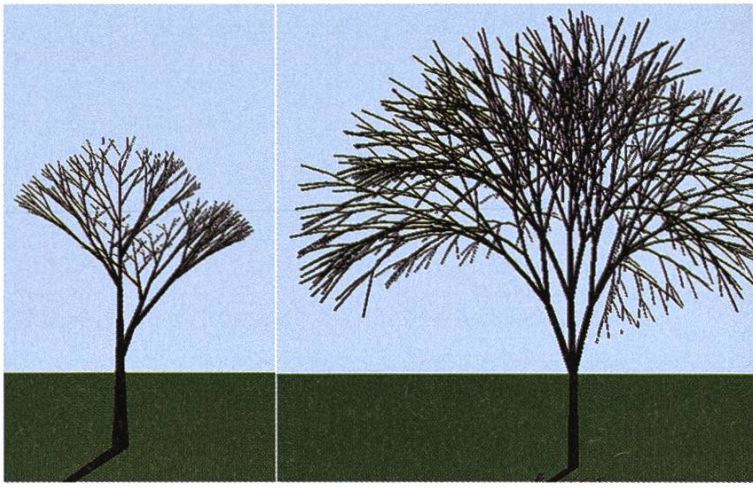


Figure 6. Growing trees with L-systems

Download the examples

The Piet example programs from this article are available at
http://pages.vis.ethz.ch/visionen/2011-3/piet/

References

[1] David Morgan-Mar, Piet
http://www.dangermouse.net/esoteric/piet.html

[2] Courtesy of Richard Mitton

[3] L-system – Wikipedia
http://en.wikipedia.org/wiki/L-system

[4] CC BY-SA 3.0 Author Wxs at Wikipedia

[5] Courtesy of Alex Horton, a promising kid with oversized Hobbit feet.

[6] The Shakespeare Programming Language
http://shakespearelang.sourceforge.net/

[7] David Morgan-Mar, Chef
http://www.dangermouse.net/esoteric/chef.html

Fictional Languages

RMS — WOULD ACTUALLY RATHER WORSHIP SUNE AND OGHMA

Drow

So you want to learn the Drow language, spoken by the noble dark elves of the Underdark. With them being Chaotic Evil in original canon, you might profit from this skill, and have rightly chosen a both beautiful and simple language to step into the world of fictional languages.

The thing to remember is: The structure of Drow society depends completely on the region you are in and the deity that is worshiped there. Most probably, you will find that this position is held by Lolth, the Queen of Spiders, who encourages inducing order through chaos and intrigue. These regions are governed by a strongly hierarchical matriarchy. However, should you find yourself in a city where Eilistraee, the goddess of song, the moon and beauty is the deity of choice, the gender roles and in fact also alignment are very much inverted. As a human or a surface elf, the latter company might be the only one to willingly accept you even as a passing traveler.

Drow grammar is incredibly similar to Common and as such should not constitute a large obstacle to the inclined reader: basic Subject-Predicate-Object order is preserved and most sentences can be built with simple syntactic replacement.

I	worship	the	godess		at midnight.
Usstan	onhir	l'	quar'valsharess	a	alantha.

Pronunciation is also blissfully related to Common: the apostrophe indicates a glottal stop (refer to the exclamation "Uh-Oh!", where it is represented by a hyphen), and all letters can be pronounced as if Latin, or indeed German. Try it out! Below you find some phrases which you might find useful.

Hello	Vendui
Goodbye	Vedaust
Human	rivvil
What is your name?	Vel'bol zhah dosst kaas?
My name is [Imoen].	Ussta Kaas zhah [Imoen].
I am from [Candlekeep].	Usstan tlun dal [Candlekeep].
I love you!	Usstan che dos!
Do you want to sleep with me? (as in *"just sleep"*)	Xun dos ssinssrin ulu v'dri xuil uns'aa?
Do you want to sleep with me? (as in copulation)	Xun dos ssinssrin ulu vith?
Would you like to see my Turing machine?	Orn'la dos saph ulu kyori ussta Turing vantua?

For further reading including a Common-Drow-Common dictionary and a grammar cheat sheet, consult the Chosen of Eilistraee at [1]. I wish you good luck on your travels. Aluve!

Klingon

(For typographical convenience, we have used here the transliteration into Latin script of all Klingon phrases.)

Much more than for Drow, we know about the roots of the Klingon language ("tlhIngan Hol" in Klingon): While in Star Trek: The Original Series (running from 1966 to 1969), Klingons spoke English to the general benefit of the viewer, Star Trek: The Motion Picture (1979) introduced the general sound of the language through a few sparse words invented by James Doohan, the actor portraying "Scotty". Later, American linguist Marc Okrand was hired to flesh out the language, and from then on was involved in language coaching for the many actors, and as consultant right up to the 2009 feature film Star Trek (don't even get me started on the naming problems).

The principle underlying the language design is the maximization of foreignness to the ears of the English-speaking audience. This means that to all speakers of German (or even Swiss German), Klingon might offer less resistance than you might have previously thought! Guttural and fricative sounds, strange rules of capitalization, and the distinct impression that even "I love you" would sound like a declaration of war; what's not to love!

Without a doubt, at this point you are going to ask yourself about the practical use of this fictional language. Fear not! You have not wasted your time reading through this tutorial. In fact, you might one day count yourself to the few lucky people on this earth who have become fluent enough to enjoy 'u', the first opera completely in Klingon, which premiered in September 2010 and has enjoyed repeat performances since then.

If this has motivated you into further studies, I will gladly refer you to the wikibook on Klingon, which has served as base for this tutorial, but has a far bigger scope (and fascinatingly enough, claims to be written completely in a variant of English called E-Prime). It can be found at [2]. You can also find hints and more phrases at the Klingon Language institute at [3].

✦

Where is the bathroom?	nuqDaq 'oH puchpa''e'
What time is it?	'arlogh Qoylu'pu'?
Today is a good day to die.	Heghlu'meH QaQ jajvam
Your mother has a smooth forehead!	Hab SoSll' Quch!

References

[1] http://www.eilistraee.com/chosen/language.php

[2] http://en.wikibooks.org/wiki/Klingon

[3] http://www.kli.org/

Rückblick: Kontaktparty 2011

TOM LAMPART — CIO KPK

Nach etwas über acht Monaten intensiver Vorbereitung war es wieder soweit: Die 26. VIS Firmenkontaktparty, kurz KP, öffnete erneut ihre Tore. Rund 80 Firmen und 250 interessierte Studenten nutzten diese Gelegenheit, um sich gegenseitig in ungezwungener Atmosphäre kennenzulernen.

Nach einem zögerlichen Start zu früher Stunde, am Samstag um 11:00 Uhr, wurde es je länger je belebter. Gut vorbereitete und motivierte Studenten führten interessante Gespräche mit freundlichen und sehr zuvorkommenden Firmenvertretern und nutzten die Gelegenheit, sich ein konkreteres Bild von ihren bevorzugten potentiellen Arbeitgebern zu verschaffen. Viele Studenten liessen sich auch von bisher nicht bedachten Firmen positiv überraschen und konnten so ihren Karrierehorizont vielversprechend erweitern.

Ob allgemeine Softwareanbieter, branchenspezifische Dienstleister, Strategieberater, Banken, Versicherungen, Mobile Developers, Betriebe aus der Telekommunikation, Medizinal- oder Energietechnik, alle waren sie da. Im Angebot waren potentielle Festanstellungen, Praktika, Masterarbeiten, Stellen als Werkstudenten und dergleichen.

Ebenso fanden sich Bachelor- und Masterstudenten, Doktoranden oder ehemalige Studenten mit Abschluss ein. Das studentische Publikum war dabei deutlich breiter gefächert als in den vorigen Jahren. Die Studenten strömten nicht nur von der ETH Zürich, sondern auch von anderen Hochschulen wie z.B. der Universität Zürich, Universität Basel oder diversen Schweizer Fachhochschulen herbei. Aus Sicht der Kontaktparty Kommission ein voller Erfolg und eine Genugtuung für unser Ressort Studentenmarketing.

Und aus Sicht der Firmen? Hauptsächlich sehr positiv. Konkret gab es z.B. ein begeistertes Feedback von einer Firma, die über 30 Gespräche führen konnte. Jedoch auch diejenigen mit nicht ganz so vielen Besuchern stellten erfreut fest, dass der grösste Teil der Studenten sehr gut vorbereitet war und dadurch massgeblich zum konstruktiven Verlauf der Gespräche beigetragen hat. Aber eine – schon seit Jahren – oft gehörte Kritik bleibt: "Zu wenig Absolventen"! Nun, Fakt ist, dass es davon einfach nur sehr wenige gibt, die Nachfrage übersteigt das Angebot

«Nirgendwo sonst ist es so einfach, mit diesen hochbegehrten Individuen in Kontakt zu treten.»

um ein Vielfaches. Doch nirgendwo sonst ist es so einfach, mit diesen hochbegehrten Individuen in Kontakt zu treten, was auch dieses Jahr wieder von vielen Firmen erlebt und bestätigt wurde.

Zu nennen bleibt noch die wohl unumstritten wichtigste Neuerung der diesjährigen KP: Der Umzug vom Campus ETH Zentrum in den ETH Science City Campus auf dem Hönggerberg. Der Entscheid, die vertrauten Gewässer der Polymensa zu verlassen, zog während der Vorbereitungen völlig unerwartete und nicht gerade kleine Probleme mit sich. Nichts desto trotz können wir mit Stolz festhalten, dass der Umzug schlussendlich ein voller Erfolg war. Natürlich ist nicht alles reibungslos verlaufen, es wurde durchaus Verbesserungspotential entdeckt. Doch genau dieses lässt uns bereits jetzt mit Vorfreude auf die nächste KP blicken, wenn wir von den gemachten Erfahrungen profitieren können, um die KP noch besser zu gestalten.

Die Geschichte der VIS-Firmenkontaktparty, der mittlerweile grössten akademischen IT-Recruiting-Messe der Schweiz, war, ist und bleibt eine Erfolgsgeschichte. ✦

Praktikumsbericht: Ein halbes Jahr Ergon

THORBEN BOCHENEK — HAT MAL ETWAS NÜTZLICHES GETAN

Ein Semester Praktikum machen. Keine Prüfungen, keine Vorlesungen, endlich mal "dort draussen" zeigen, was man kann! Sicherlich spielen einige von euch mit dem Gedanken, das mal zu tun. Falls ihr noch nicht wisst wo, ich habe da einen Vorschlag: Ergon.

Für mein Industriepraktikum wollte ich eine Firma, die nahe an der Praxis und weit weg von der Forschung ist. Ich wollte ernsthaft ausprobieren, wie das ist, wenn man als Informatiker arbeitet, ob mir das gefällt, oder ob ich mein Studium besser in eine andere Richtung lenken soll. Recht bald hat sich bei mir Ergon als eine gute Idee herauskristallisiert. Ergon hat ihre Büros fast direkt am See in der Nähe vom Bahnhof Stadelhofen. Mit 140 Personen ist die Firma gross genug, damit man vom Wissen der Kollegen profitieren kann, aber auch klein genug, um nicht in Masse oder Bürokratie unterzugehen. Praktisch alle Mitarbeitenden haben mindes-

tens einen ETH-Master in Informatik, man versteht daher sehr gut, was ein Praktikant von der ETH weiss, kann – und was er noch lernen muss.

Insgesamt habe ich etwa ein halbes Jahr bei Ergon gearbeitet und war je länger je begeisterter von meinem Praktikum. Weil ich zwischendurch noch Prüfungen an der ETH absolviert habe, bestand mein Praktikum aus zwei Teilen. So konnte ich auch in zwei ganz unterschiedliche Teams hereinschnuppern.

Im ersten Team ging es ein bisschen mehr wie in einem "normalen" Praktikum her. Ich hatte eine Aufgabe, die zwar wichtig, aber ein bisschen abseits vom Tagesgeschäft war und bei der ich mich erstmals einarbeiten sollte. Ich arbeitete an der Zeiterfassung- und Zeitplanungssoftware Zebra, die unter anderem bei Coop eingesetzt wird. Ich habe dort neue Hierachiestufen eingebaut, was cool war, weil ich vom Kern über die Datenbank bis zum Interface alles einmal sehen musste.

Der zweite Teil meines Praktikums war wesentlich spannender, weil ich vollkommen ins Team integriert wurde. Ich stiess praktisch in der Gründungsphase des Teams hinzu und wurde behandelt wie jeder andere Entwickler auch. Zum ersten Mal konnte ich hier aus erster Hand direkt Extreme Programming und Scrum kennenlernen, was wirklich eine coole Erfahrung war.

Die ganze Zeit über habe ich direkt in das Repository commitet und mein Code wurde praktisch sofort von allen anderen Entwicklern verwendet: Mit allen Konsequenzen – hilfreich und zerstörerisch – die das mit sich bringt. Das war sehr motivierend. Die Arbeit macht einfach mehr Spass, wenn man weiss, dass man etwas Nützliches tut und andere zu schätzen wissen, was man tut.

Generell habe ich mich bei Ergon sehr gut aufgehoben gefühlt. Das Klima in der Firma ist extrem angenehm und meistens recht entspannt. Einen viel besseren Ort für mein Praktikum hätte ich mir wohl nicht aussuchen können. Falls also jemand von euch mal in die "klassische Softwareentwicklung" reinschauen will, kann ich euch Ergon nur wärmstens empfehlen. ✦

COMIC

Saturday Morning Breakfast Cereal by Zach Weiner
http://www.smbc-comics.com/

Helvetic Coding Contest – Die Revanche

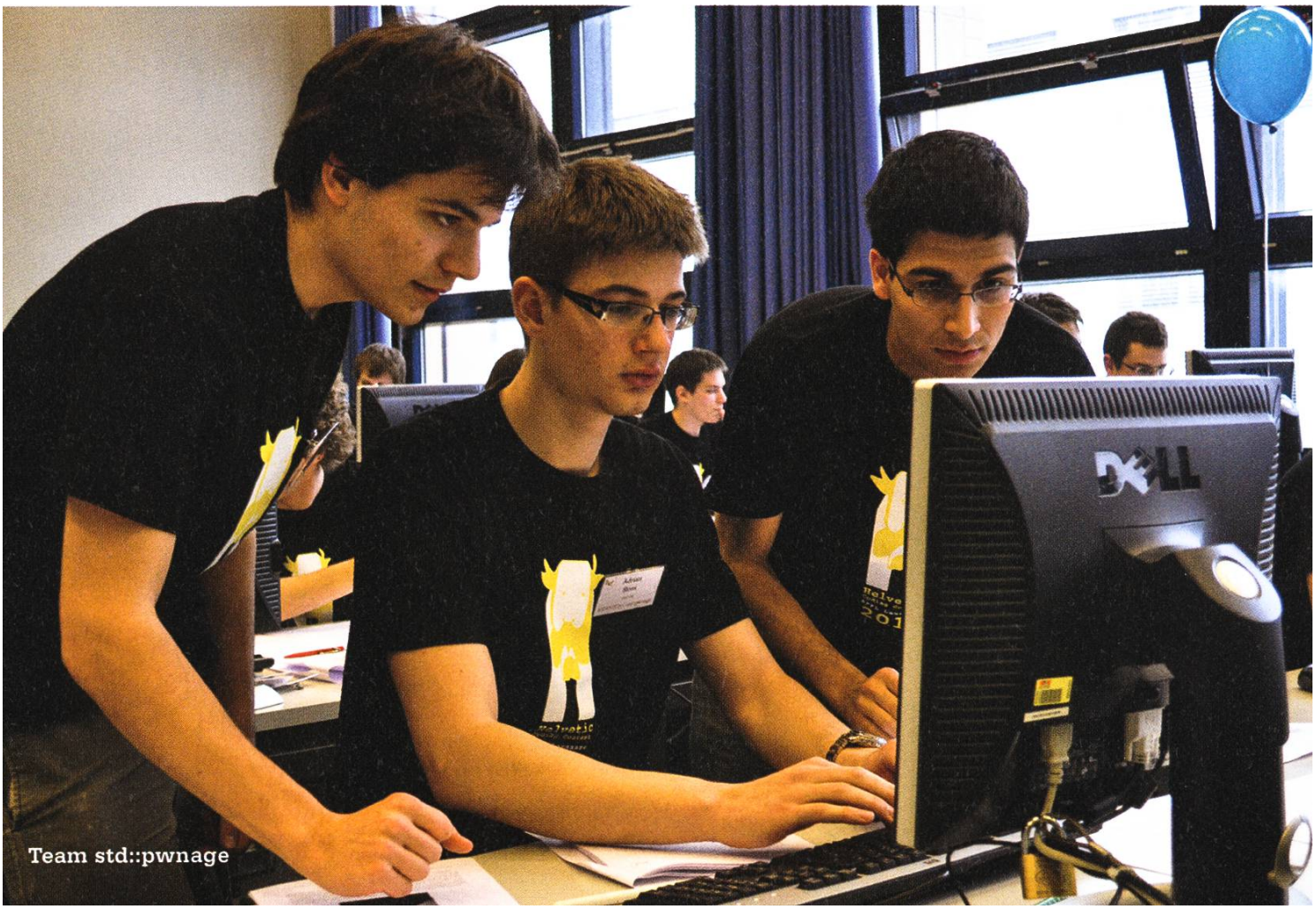
CHRISTIAN HELBLING — WAR WIEDER DABEI

Am 12. März fand zum zweiten mal der Helvetic Coding Contest[1] statt. Trotz gleichzeitig stattfindender Kontaktparty reisten einige Teams der ETH Zürich nach Lausanne. Deren Leistung liess nicht zu wünschen übrig!

Nach dem gelungenen ersten Helvetic Coding Contest vor einem Jahr (siehe Artikel in den Visionen 2010/2[2]) stellte sich die Frage, ob die nächste Ausgabe wieder so gut wird. Dies zum einen bezüglich organisatorischen Aspekten und zum anderen natürlich betreffend des Abschneidens unserer Teams.

Organisatorisch war der Anlass wieder top! Bis auf ein paar kleinere Verspätungen im Zeitplan lief alles rund. Auch unser Wunsch nach Kaffee wurde erfüllt und es gab sogar Lachsbrötchen.

Der Wettbewerbsmodus blieb gleich: Ein Team besteht aus maximal drei Personen, hat ➔



Team std::pwnage

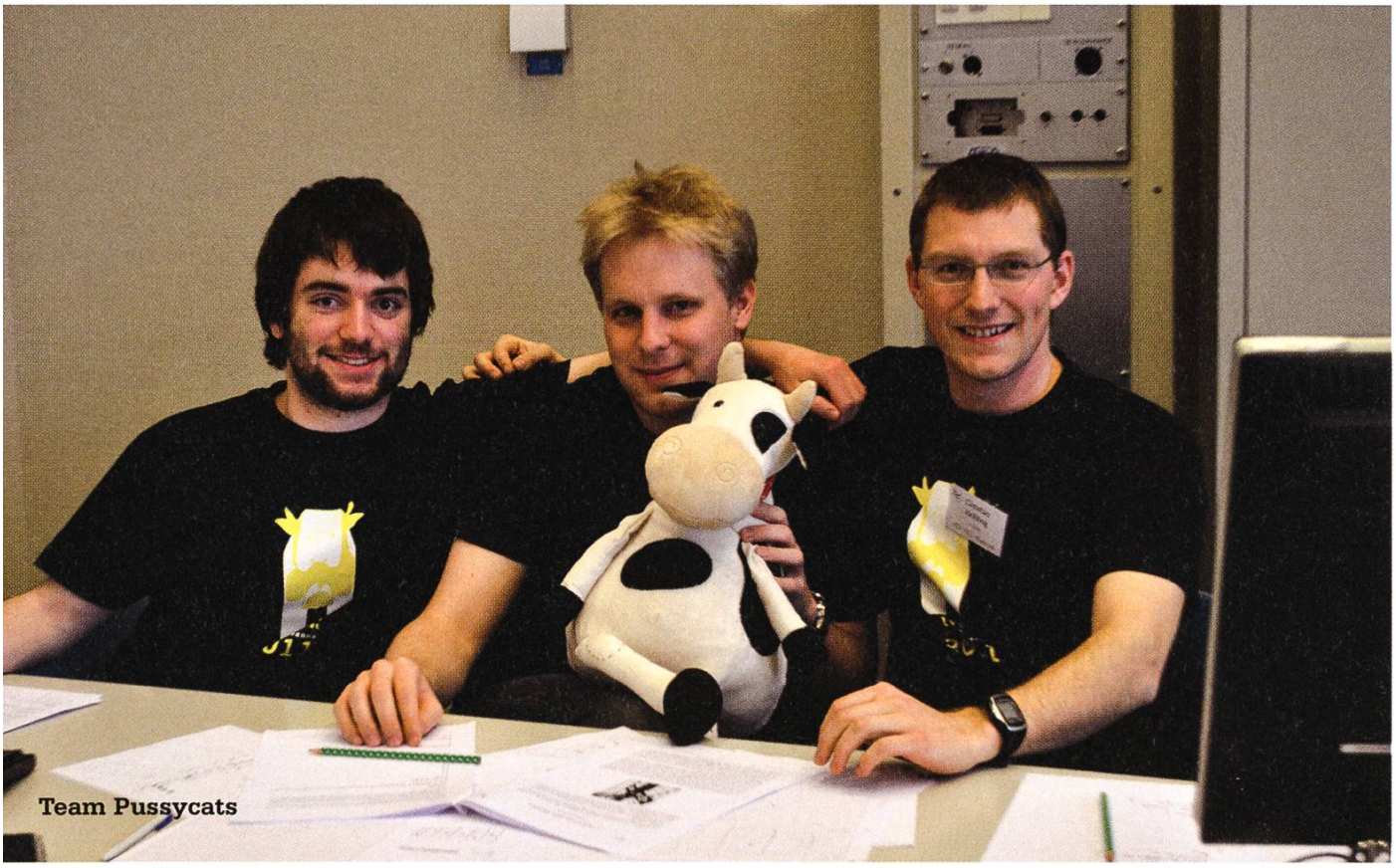
Want to work in a team of *smart* and *ambitious* people? You could fit right in.

If you're looking for a career that'll continue to stretch you long after university, UBS is a great place to start. Whatever you're studying now, from your first day at UBS you'll continue to learn from the very best in their field. And you'll be realising your potential as part of a great team that believes in succeeding together.

As well as our Graduate Program, we offer internships to students in all stages of their academic career – so visit **ubs.com/graduates** to explore a world of opportunities and be part of our success.

UBS

Zugriff auf einen Computer und soll in fünf Stunden möglichst viele der acht Probleme lösen. Dazu muss das Team Code in C++, C oder Java schreiben der eine genaue Spezifikation erfüllt. Dieser Code wird dann einem Judge-Server geschickt, der das Programm kompiliert und dessen Korrektheit anhand von zehn Testfällen überprüft. Für jeden Testfall, bei welchem das Programm die korrekte Antwort liefert und innerhalb der Zeitlimite terminiert, gibt es einen Punkt. Testfälle, die erst nach mehreren versuchen korrekt laufen, geben zusätzlich Strafpunkte. Die Strafpunkte kommen jedoch erst bei gleichem Punktestand zum Zug.



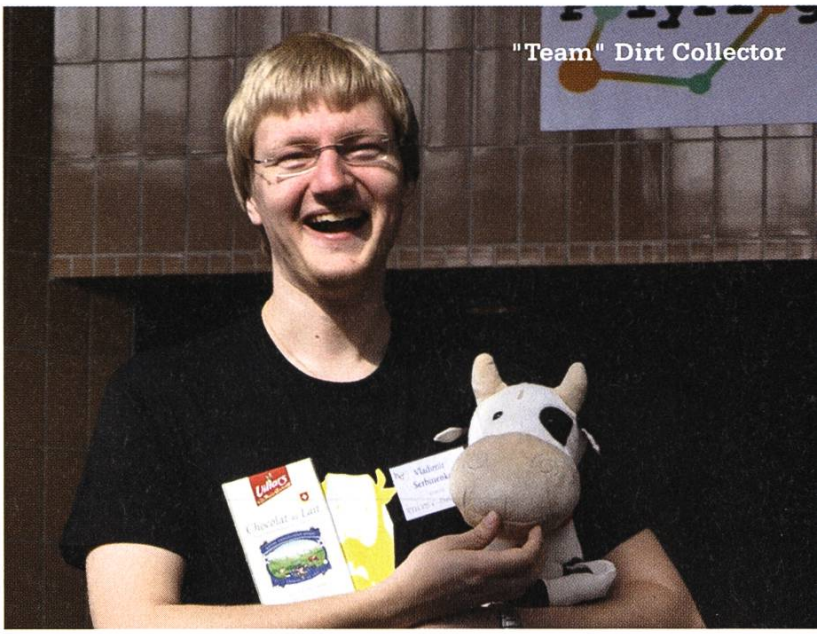
Team Pussycats

Von den acht Problemen waren drei etwas speziell. Zwei waren interaktiv, was bedeutet, dass das einzuschickende Programm Funktionen einer Bibliothek benutzen muss anstatt klassisch vom Standard-Input zu lesen und auf den Standard-Output zu schreiben. Ein weiteres Problem konnte offline gelöst werden. Das wiederum heisst, dass schon alle Testeingaben vorhanden sind und nur noch die korrekte Ausgabe eingeschickt werden muss. Somit werden hier die Beschränkungen von Programmiersprache und Zeitlimit etwas aufgehoben – es kann z.B. auch in Python programmiert werden. Auch dieses Jahr hat das Polyprog Team (so heisst die organisierende Gruppe) wieder super Arbeit geleistet und eine interessante Auswahl von Problemen zusammengestellt. Das volle Problembündel kann unter [3] heruntergeladen werden.

Die Teams der ETH Zürich waren auch dieses mal wieder top! Und zwar nochmals deutlicher als letztes Jahr: wir schnappten uns gleich die ersten drei Ränge. Bemerkenswert dabei ist, dass „Dirt Collector" Vladimir auf zwei Team-Mitglieder verzichtet und die Herausforderung

ganz alleine angenommen hat. Trotzdem wurde er Zweiter! Als Mitglied des Sieger-Teams muss ich noch sagen, dass unser Ziel eigentlich nur war, Vladimir zu schlagen. Und weil er so gut war, mussten wir halt gewinnen ;)

Nach der Rangverkündigung ging es dann noch ins Zentrum von Lausanne, um bei einer Pizza noch eine Weile über den Contest und andere Sachen zu philosophieren. Der Anlass war wirklich gelungen und wir freuen uns natürlich schon wieder auf die nächste Ausgabe! ✦


"Team" Dirt Collector

#	Country	Team	Solved	Points
1	ETH	ETH VIS I - Pussycats	32	-21
2	ETH	ETH VIS V - Dirt collector	26	-9
3	ETH	ETH VIS IV - std::pwnage	22	-17
4	EPFL	EPFL YAWN : reloaded	21	0
5	ETH EPFL	EPFL_ETH BubbleTrouble and Ivan	21	-67
6		SOI We eat your code	17	-3
7	EPFL	EPFL Serendipity	17	-14
8		unige Le Culte du Sandwich	14	0
9	EPFL	EPFL brut	14	0
10	Hes	HES_SO *NULL	13	-3
11	EPFL	EPFL Annemasse la Menace	13	-16
12	EPFL	EPFL stupide et marrant	12	-12
13	EPFL	EPFL IllegalArgumentException	10	-8
14	ETH	ETH VIS II - Fun Team	9	0
15		unige The Madigots	7	-4
16	EPFL	EPFL AleA	7	-7
17		HEIG-VD Les Bracaillons	7	-9
18	EPFL	EPFL No DNA	6	0
19		IT BFH Electrum	6	0
20	Hes	hepia Shanghai Check	5	-2

Top 20 (komplette Rangliste unter [3])

Interessiert an Programmierwettbewerben?

Neben der Teilname am HC² organisiert das ACM-Team des VIS alljährlich die ETH-Vorauswahl und -Trainings für den ACM ICPC-Wettbewerb[4]. Willst du selber auch einmal dabei mitmachen? Oder einfach nur trainieren? Melde dich einfach mal bei acm@vis.ethz.ch und/oder abonniere die **acm-interessenten** Mailinglist[5].

Referenzen

[1] http://hc2.ch/

[2] http://www.vis.ethz.ch/de/visionen/pdfs/2010/visionen_2010_2.pdf?end=24&start=20

[3] http://hc2.ch/res/hc2publish.zip

[4] http://cm.baylor.edu/welcome.icpc

[5] https://mail.vis.ethz.ch/mailman/listinfo/acm-interessenten

[i][A][E][T][H]

Informatik-Alumni ETH Zürich

IAETH Berufsumfrage 2011

Der Verein Informatik Alumni ETH Zürich (IAETH) führt alle zwei bis drei Jahre eine Berufsumfrage unter seinen Mitgliedern durch. Dieses Jahr beantworteten fast 350 ehemalige Informatikstudierende die 42 Fragen, womit eine Rekordbeteiligung erreicht wurde. Einmalig sind die Zeitreihen der gleichen Fragen über mehr als ein Jahrzehnt hinweg. Dies erlaubt uns, die Entwicklung und Zukunftsaussichten des Informatik Marktes Schweiz über viele Jahre hinweg zu beurteilen.

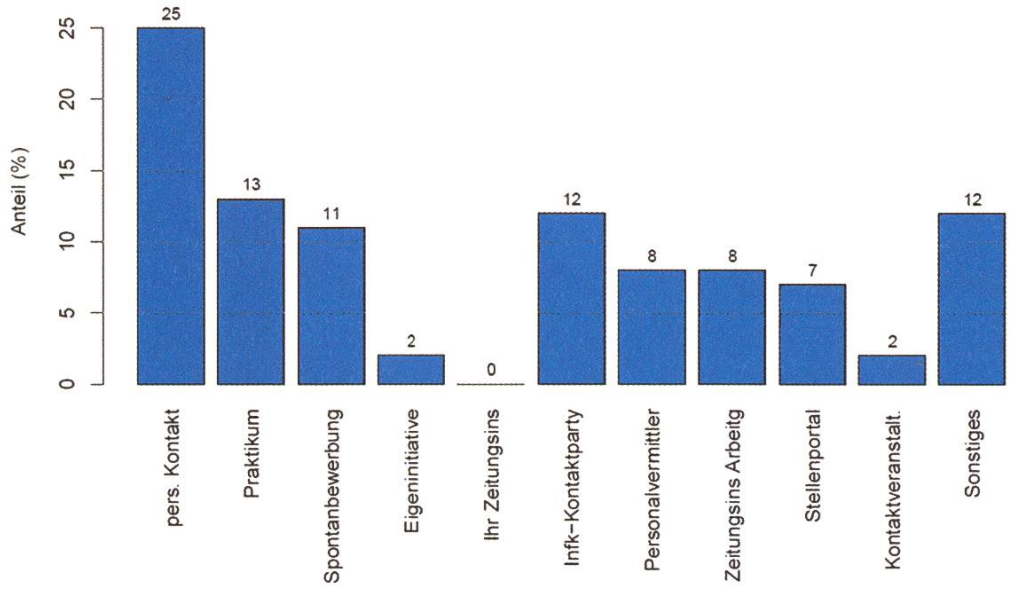
Zudem wurden neue Fragen eingeführt um den Zusammenhang des Gehalts mit Branche, Region, Berufsjahre Weiterbildung, Arbeitszeit etc., aufzuzeigen.

In diesem Artikel wird ein Ausschnitt präsentiert. Die detaillierte Analyse ist nicht öffentlich, sondern exklusiv und ausschliesslich den IAETH-Mitgliedern zugänglich. Folgende Erkenntnisse stellen wir Euch vor:

- Wie bekommt man einen Job?
- Welche Weiterbildungen werden gemacht?
- In welcher Branche arbeiten die Informatikalumni?
- Was sind die Haupttätigkeiten von Informatikalumni?

Wie bekommt man einen Job?

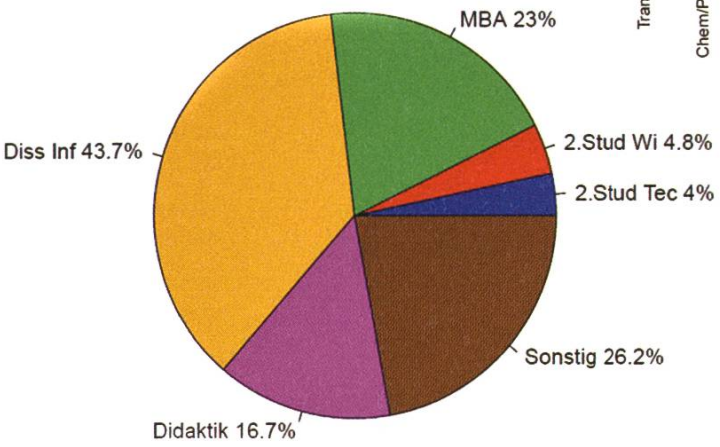
Wir haben die Umfrageteilnehmer gefragt, wie sie die jetzige Stelle gefunden haben. Die Antworten sind eindeutig:



Die meisten haben ihren Job über persönliche Kontakte gefunden. Hier bietet der IAETH ausgezeichnete Möglichkeiten, persönliche Kontakte in der Informatik zu erhalten und auch neue aufzubauen. Wir führen regelmässig einen Stammtisch durch, wo man ungezwungen ins Gespräch kommt, Leute kennenlernen kann und alte Gesichter wieder trifft.

Weiterbildungen

Rund 45% unserer Alumni haben über die Jahre zusätzlich zum Informatik Studium eine Weiterbildung besucht. Die häufigsten sind: Doktorat, Didaktik, MBA:



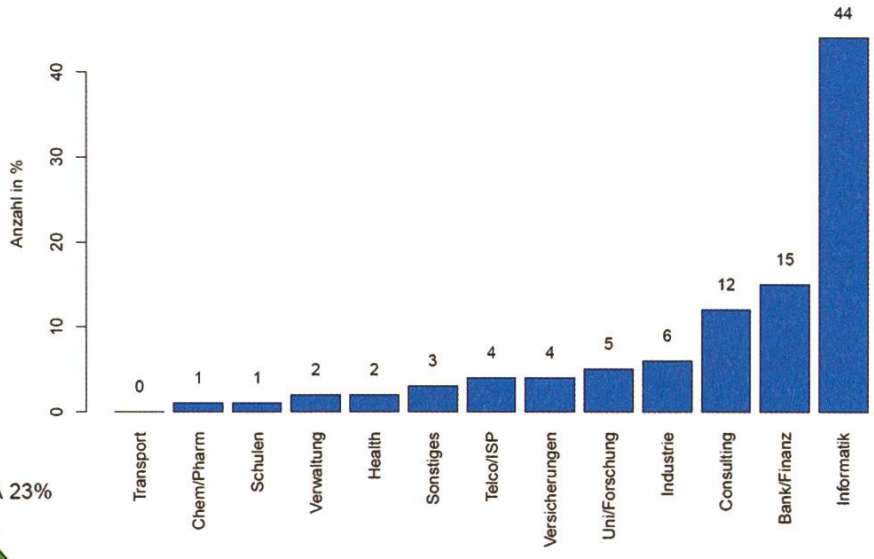
Die Berufsumfrage analysiert unter anderem die Abhängigkeit des Gehalts von der gewählten Weiterbildung und der Anzahl Berufsjahre:

Das durchschnittliche Gehalt ist in erster Linie von der Anzahl der Berufsjahre abhängig.

Das MBA als Zusatzausbildung wirkt sich am stärksten positiv auf das Salär aus.

Branche

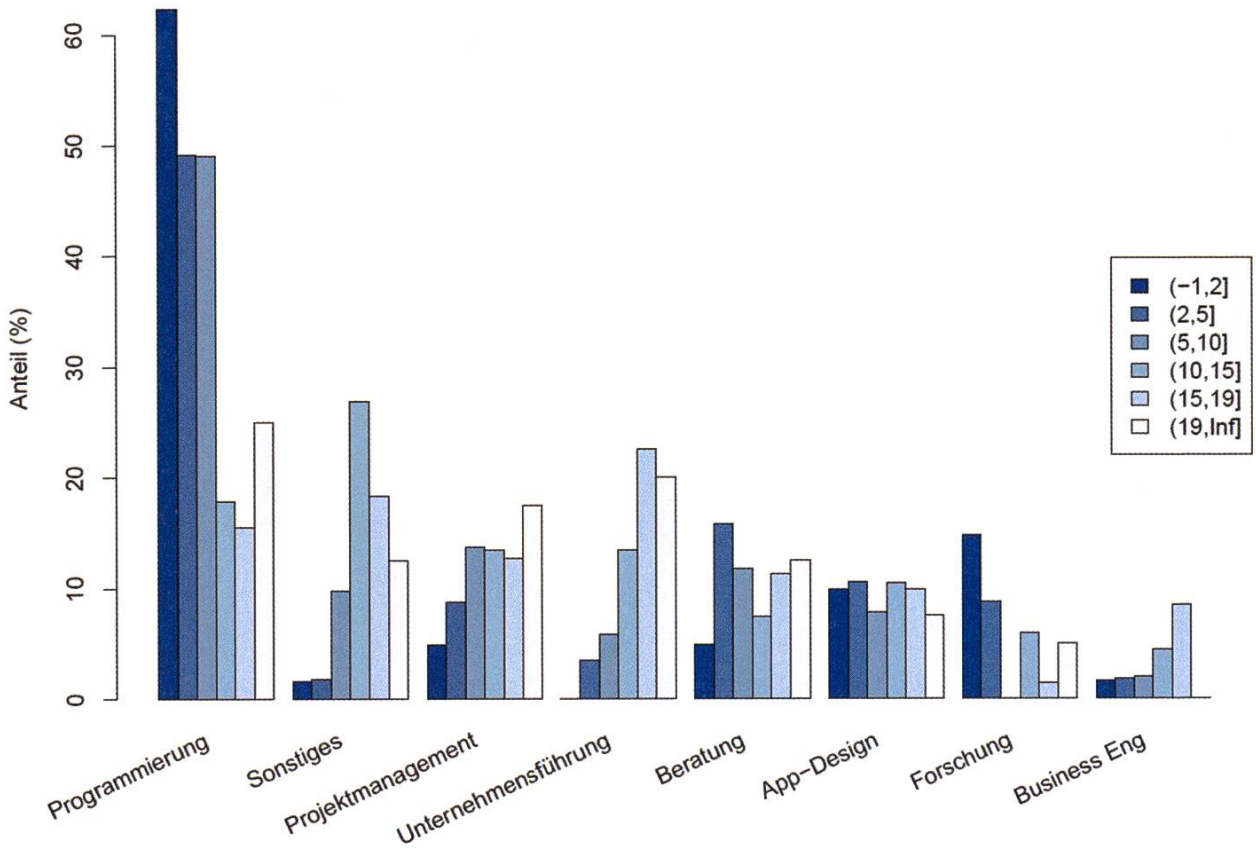
Die meisten Informatikalumni bleiben in ihrer Karriere der Informatik treu.



Tätigkeit

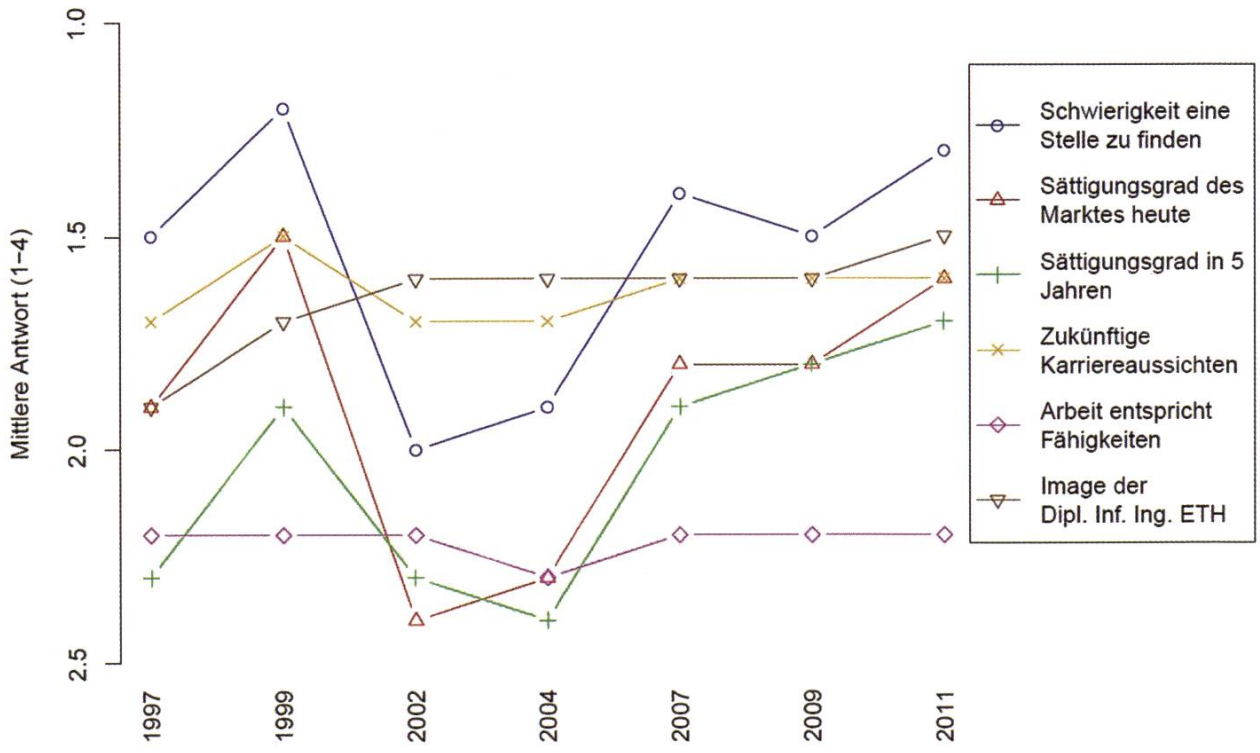
Die Haupttätigkeit der Informatikalumni verschiebt sich im Verlauf der Berufsjahre. Unmittelbar nach dem Studium ist für über 60% die Programmierung die wesentlichste Aufgabe. Dabei gilt es die verschiedenen Konzepte aus dem Studium praxisnahe einzusetzen. Über die Jahre verschiebt sich das Schwergewicht mehr auf Management Aufgaben wie Projektmanagement oder gar Unternehmensführung.

➔



Marktentwicklung

Bei jeder Umfrage wird die Stimmung in der Branche erhoben und mit den bisherigen Umfragen verglichen.



Die Stimmung ist schon fast wieder so optimistisch wie zur Jahrtausendwende während dem Internet-Hype. Die Alumni schätzen es als einfach ein heute eine Stelle zu finden. Auch in fünf Jahren wird der Sättigungsgrad des Marktes mit Informatikern als tief eingeschätzt und damit bleibt es einfach eine Stelle zu finden. Dies sind doch gute Aussichten für alle Studierenden.

Dank

Der IAETH bedankt sich nochmals bei allen Mitgliedern, welche sich die Zeit genommen haben, an der Umfrage teilzunehmen! Zudem möchten wir uns insbesondere bei Martin Meier bedanken, der die Umfrage durchgeführt und die anonymisierten Daten ausgewertet hat.

Mitglieder des IAETH haben Zugriff auf die detaillierten Auswertungen aller Fragen inklusive der durchschnittlichen Gehaltserhöhung pro Jahr und der weiteren statistisch signifikanten Einflussfaktoren.

Jeder Informatik-Alumni, welcher noch nicht Vereinsmitglied ist, kann sich unterhttp://www.iaeth.ch/beitreten/beitrittsformular.html dafür anmelden. Mit dem individuellen Login und Passwort finden sich im geschützten Memberbereich alle Auswertungen der aktuellen und der bisherigen Berufsumfrage.

Für den Vorstand
Melanie Raemy

✦

COMICS

Saturday Morning Breakfast Cereal by Zach Weiner
http://www.smbc-comics.com/

Jobs mit Visionen

VORSTAND

Visionäres Denken und Freiwilligkeit bilden das Fundament unseres Fachvereins. Auf diesen Werten basieren sämtliche Dienstleistungen des VIS. Doch im Verlauf der letzten Jahrzehnte wurde unser Fundament leicht brüchig. Freiwilligkeit scheint eine selten gesehene Tugend zu werden. Visionen werden oft als unrealistische Tagträumerei abgestempelt. Doch wo liegt der Ursprung dieses Sinneswandel einer ganzen Gesellschaft? Wahrscheinlich liegt ein Teil davon bei der immer und überall fehlenden Zeit. Aber wie soll man jemals Zeit haben ohne sie sich aktiv zu nehmen?
Sowohl Visionen wie auch freiwillige Arbeit benötigen genau diesen Schritt – den aktiven Entscheid sich Zeit zu nehmen – aus Überzeugung und nicht aus leistungsorientierten Überlegungen. Jedem Informatikstudenten wird wohl während seinem Studium bewusst, dass oft nicht der effizienteste Weg der lehrreichste und eleganteste ist. Also warum sich nicht aktiv und kreativ im VIS engagieren?

Aus Überzeugung

Martin Otth

Quästur

Die Quästur befasst sich mit allen finanziellen Bereichen des VIS (Ausnahme Kontaktparty-Kommission, sie hat eine eigene Quästur). Dadurch werden von der Quästur die Kasse sowie die Konten des VIS verwaltet. Weiter führt die Quästur die Buchhaltung des VIS. Somit unterliegt die Abschlussrechnung und das Budget, welches an jeder Mitgliederversammlung präsentiert wird, diesem Ressort.

Als Quästorin hat man dadurch einen tiefen Einblick in einen Grossteil der Fachvereinsaktivitäten. Arbeitest du also gerne selbstständig, zuverlässig und suchst nach einer neuen Herausforderung, welche sich vom Studienalltag unterscheidet? Dieses Ressort könnte genau das richtige für dich sein.

Systemadministration

Der VIS unterhält für die Vereinsarbeit sowie auch zwecks Dienstleistungen für Studenten diverse Server und Workstations, welche vom Sysadmin gehegt und gepflegt werden. Viele Aspekte dieser Pflege sind mittlerweilen automatisiert; sei es das automatische und kontinuierliche Backup via DRBD und rdiff-backup oder die Administration mittels Puppet. Zudem sind die meisten Server als VMs realisiert, was das Management erleichtert.

So bleibt einem als Sysadmin zusätzlich zur Instandhaltung der Hardware auch noch Zeit, mit neuen Technologien zu experimentieren oder am bestehenden Setup zu feilen und zu optimieren. Dazu ist der VIS in der vorteilhaften Lage, dass wir von der guten EDV-Infrastruktur der ETH profitieren können; so haben wir ein Rack in einem Serverraum der ETH, natürlich inklusive der gewohnten Gigabitanbindung.

Auch an Hardware fehlt es dem VIS trotz den geringen Investitionen nicht, da der VIS vieles von der ETH oder anderen Stellen erben kann. So warten auf einen neuen Sysadmin beispielsweise auch mehrere fancy Server, die wir von einem Simulationscluster der Firma Sulzer Innotec geerbt haben. Solltest du Freude an Hardware, Linux und Servern haben, ist der Sysadmin-Posten im VIS-Vorstand für dich genau das Richtige. Du kannst natürlich auch direkt neue Services auf die Beine stellen oder bestehende verbessern, die Infrastruktur dafür ist vorhanden und problemlos nutzbar. Zudem wirst du nicht ins kalte Wasser geworfen, der alte Sysadmin ist nachwievor noch da und steht mit Rat und Tat zur Seite.

Webadministration

Bei der Webadministration geht es primär und die Wartung und Verbesserung der VIS-Webseite. Dazu gehört die Administration der Fotogalerie, allfällige Events und dessen Anmeldungen, die Suche nach Verbesserungsmöglichkeiten, und das Überprüfen der Seite auf veralteten Inhalt. Für das Erfassen von Inhalt ist der Webadmin im allgemeinen nicht verantwortlich, vielmehr muss er sicherstellen, dass andere Personen die Unterseiten für ihren Zuständigkeitsbereich bearbeiten können.

Auch zum Zuständigkeitsbereich des Webadmins gehört der interne Bereich, in den viele Tools zur Vereinfachung der Arbeit für andere gehören. Der Webadmin muss auch diese Tools warten und allfällige Störungen beheben (was aber nur selten nötig ist).

Last but not least hat der Webadmin auch auf alles Zugriff, und agiert daher auch etwas als zweiter Systemadministrator. Je nach Absprache mit dem Sysadmin kann der Webadmin auch die Aktualisierung der Software auf den verschiedenen Server übernehmen, es ist also im Interesse aller, wenn der Webadmin auch ein wenig über die Administration von Linux-Servern und Desktops weiss.

Das Interessante an der Webadministration sind natürlich nicht die oben beschriebenen Wartungsarbeiten, sondern vielmehr das Ausdenken und Realisieren von neuen Projekten, ➔

Tools und Webseiten. Falls du also Lust auf Coden und irgendeine Idee hast (optional, von denen gibt es immer genug), dann könnte die Webadministration etwas für dich sein. Voraussetzungen wären Erfahrung mit oder Interesse an Linux (und diverse Software um Linux-Server) und Python.

Prüfungssammlung

Die Prüfungssammlung ist der perfekte Einstiegsjob, bei dem du zwar bei den regelmässigen Vorstandsmeetings und bei den Vorstandsessen dabei bist, aber nicht allzu viele Verpflichtungen hast.

Der VIS verwaltet eine Sammlung an alten Prüfungen für die Studenten und druckt jedes Semester Prüfungsbündel. Zu den Aufgaben der Prüfungssammlung gehört es, die Professoren nach neuen Prüfungen zu fragen und die Prüfungsbündel zu erstellen und beim SPOD zu drucken. Suchst du eine Herausforderung, kannst du auch zusätzlich die online Prüfungssammlung vorantreiben.

Infrastruktur

Nervt es dich, wenn es im VIS-Büro keine Kaffeebohnen mehr gibt oder die Kaffeemaschine einmal mehr kaputt ist? Dann ist Infrastruktur DER Job für dich.

Du bist verantwortlich für Nachschub an Getränken und Materialien, damit alle anderen Ressorts überleben können. Dabei bekommst du einen guten Einblick in die anderen Ressorts, da du auch den Überblick über deren Schränke und Inventar behalten solltest.

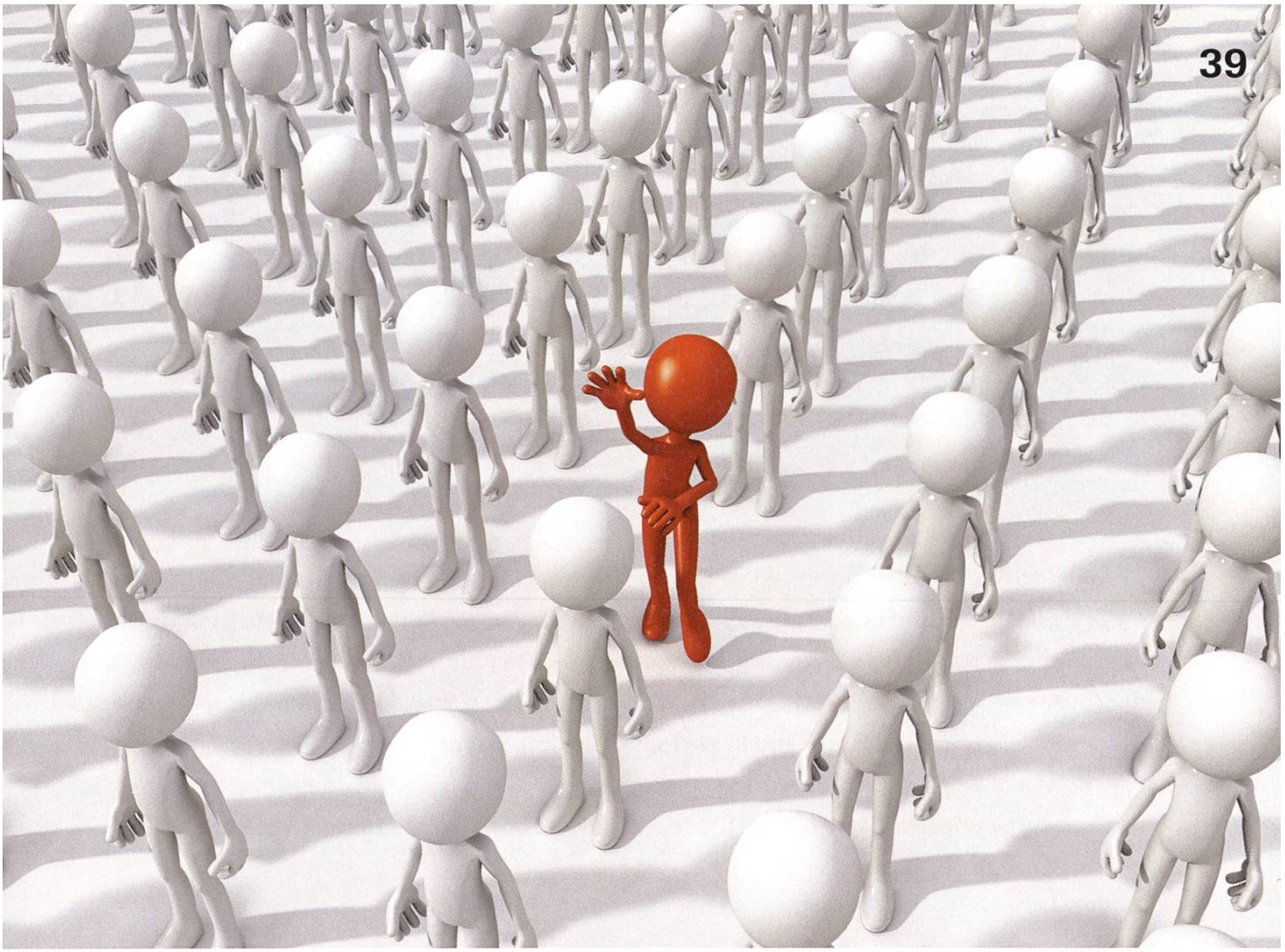
Einige Arbeiten können von zu Hause gemacht werden. Aber die meisten Arbeiten müssen im Büro und Lager erledigt werden, damit auch alles an seinem Platz ist und bleibt.

Als Infrastruktur Vorstand hast du also alles und alle in der Hand.

Information/Aktuar

Als Informationsverantwortlicher ist das Managen der verschiedenen Mailinglisten und das Beantworten von allgemeinen Fragen an den Verein der Hauptbestandteil der Arbeit. Böse Zungen behaupten, dass der Informationsminister eigentlich nur die Sekretärin des VIS ist. Da Information nur ein sehr kleines Ressort ist, wird es in der Regel vom Aktuar übernommen. Der Aktuar übernimmt das Protokollieren an den Vorstandssitzungen und an der MV.

Da die meisten Arbeiten in diesen Ressorts zu einem beliebigen Zeitpunkt von Irgendwo erledigt werden können, ist dies der ideale Job für Studenten, welche gerne auch mal von Zuhause aus arbeiten. Wenn du dich immer und überall einmischen möchtest, so geben dir diese Ressorts ganz offiziell die Möglichkeit dazu.



Layout Visionen (∉ Vorstand)

Du hast ein Auge für Design? Du wolltest schon immer eine Zeitschrift gestalten? Du interessierst dich für Typografie?

Wenn du eine (oder mehrere) der obenstehenden Fragen mit "Ja" beantworten kannst, dann bist du die richtige Person für's Visionen-Layout-Team. Als Visionen-Layouter bist du verantwortlich für Gestaltung und Layout dieses Magazins. Du erhältst die korrigierten Texte und Bilder der Artikel und die Inserate in Rohform und kannst frei entscheiden, wie du die Artikel und Inserate im Heft anordnest. Zusätzlich kannst du auch die Titelseite frei gestalten und (nach Bedarf) Artikel mit zusätzlichen Bildern illustrieren. Dazu kannst du auch Bilder von fotolia verwenden.

Der Aufwand fürs Layout ist nicht überwältigend: der totale Layout-Zeitaufwand beläuft sich auf ca. 18 Tage pro Jahr (3 Tage pro Ausgabe des Visionen). Allerdings gibts (da das Visionen in zum Voraus bestimmten Kalenderwochen erscheint) fixe Deadlines, an welchen das Layout abgeschlossen sein muss.

Hast du Fragen und/oder Interesse an einem der Vorstands-Ressorts? Oder bist du sonst einfach interessiert an Mitarbeit im VIS? Dann schau doch einfach mal im VIS-Büro (CAB E 31) vorbei oder schreib eine Mail an vis@vis.ethz.ch.

✦

The Mobile Phone as the Enabler of an Internet of Things

ROBERT ADELMANN, CHRISTIAN FLOERKEMEIER, CHRISTOF RODUNER

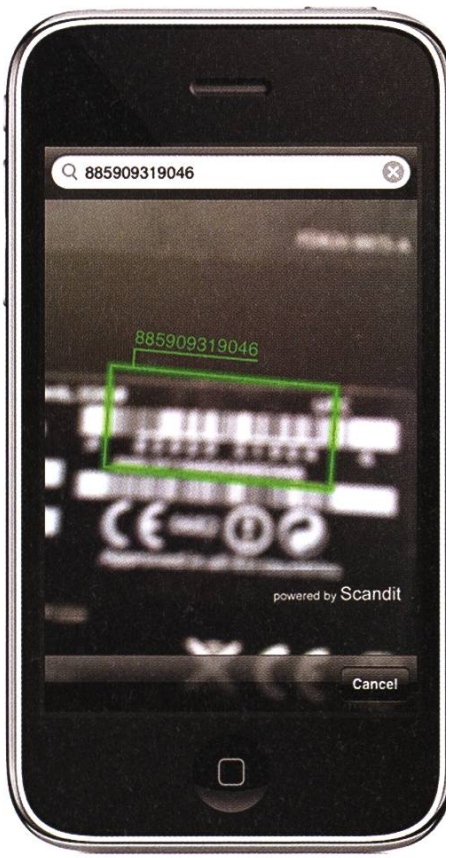
Today's Internet is largely limited to the virtual world linking computers and more recently mobile phones. At the edge of today's Internet, there remains a big gap where billions of real world objects including everything from appliances, logistical units and even everyday products are not "connected". In this article, we describe our research to bridge this gap between the physical and virtual world and in particular the important role the mobile phone can play. We showcase some of our latest research projects and describe how a PhD research project evolved into commercial software that today powers barcode scanning on millions of mobile phones around the globe.

Identifying Everyday Items

Over the past 40 years, barcodes have become a ubiquitous means of object identification. Hundreds of millions of barcodes are in use today to optimize supply chains and facilitate checkout in retail stores. Until recently, however, the scanning of barcodes required specialized scanner equipment that restricted the use of barcodes to the commercial sector and proprietary applications. Object identification via barcodes was not possible for the "masses" because cost and form factor of the scanners were prohibitive.

When the first mobile phones featured built-in cameras, this was set to change. The camera in the phone could turn every mobile phone into a barcode scanner. Leveraging the capabilities of the mobile phone such as the display, processor, networking capability, storage, and keyboard, the mobile phone could become a communication proxy for the millions of every day products without computing capabilities.

In our research group, we initially started developing mobile barcode scanning solutions for 2D optical codes because those codes were easier to decode than 1D barcodes with the early camera phones. While scanning 2D codes was easier, those codes were not common on everyday prod-

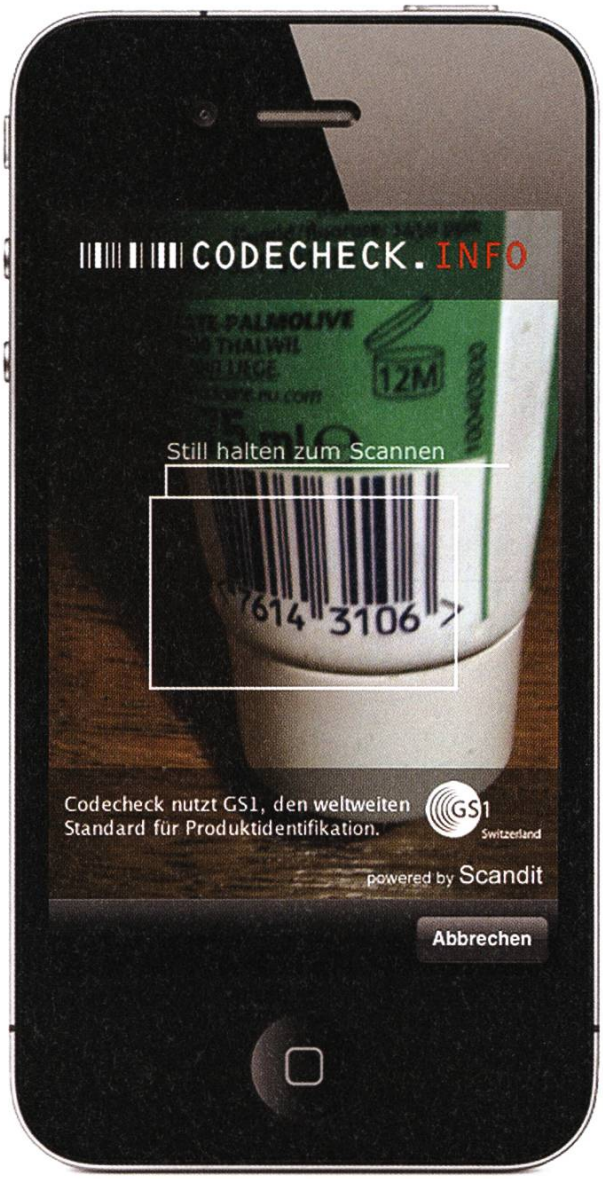


ucts. We, therefore, soon started the development of Batoo – a mobile barcode scanning technology for 1D barcodes.

Batoo – 1D Barcode Scanning for Camera Phones

The research project Batoo set out to investigate to what extent one-dimensional barcodes could be recognized using standard, off-the-shelf camera phones. This was no trivial task—mobile phone cameras often do not have autofocus, so barcode images are typically blurry. Even for phones with autofocus, motion blur, barcodes very close to the camera, lighting conditions, and warped product packaging make fast and reliable barcode scanning a challenge.

What started initially as the Batoo research project in a PhD thesis evolved into today's



fastest and most reliable commercial barcode scanning technology for mobile phones. The software known as "Scandit SDK" is being licensed via the ETH spin-off company Mirasense (www.mirasense.com) to mobile app providers around the globe. It has turned barcode scanning into a universal capability of mobile phones, with applications ranging from in-store price comparison or informing consumers about ingredients to the disruptive replacement of expensive hardware scanners in the entire supply chain. The Scandit SDK today powers for example popular Swiss mobile apps such as the Codecheck, Comparis and Ex Libris iPhone apps. →

BIT – A Browser for the Internet of Things

The recognition of barcodes with mobile phones is only one step towards the vision of an Internet of Things and the interaction with everyday objects. Once a barcode has been scanned and the underlying object has been identified, users often need to obtain information about the respective object or possibly interact with the object or its digital counterpart. Today's mobile apps with built-in barcode scanning only represent one, single information channel and this is limiting. You can either access Amazon reviews and prices, Codecheck's unbiased and independent product information, or Comparis' price comparison services. As the user you have to decide on one service before you scan the barcode or scan it again later with a different app. To address this problem, we are working on "BIT"—a universal browser for an Internet of Things where multiple information sources and different services are available to the user. He or she can scan an object once and see a list of third party services and information services at a glance.

What's next?

There is of course more to object identification than barcode scanning. Other image-based identification technologies and radio-based identification technologies such as RFID and NFC are alternative technologies to bridge the gap between the real and physical world. RFID has seen adoption in certain niche applications and in some supply chains, but has not seen significant consumer adoption beyond ski ticketing, building access and car immobilizers. Similar to barcode scanning, we believe that the mobile phone will play an essential role. Significant adoption of RFID in consumer applications will happen once the corresponding reader devices are being integrated into mobile phones on a grand scale.

Getting involved

Do mobile computing and backend infrastructures fascinate you? Do you want to build a browser for an Internet of Things, design scalable architectures to store and analyze millions of barcode scans and RFID reads, or develop next generation product identification technologies? There are many ways to get involved. We offer exciting lab projects, semester and Master's theses on a continuous basis within the Distributed Systems Group or in the context of the spin-off company Mirasense. At Mirasense, we are also looking for talented software engineers to help us develop better product interaction technologies and services. For more information, contact us directly via email (rodunerc@inf.ethz.ch) or check out the projects at http://www.vs.inf.ethz.ch/edu/theses.html. We are also collaborating with the Auto-ID Lab at the Massachusetts Institute of Technology (MIT) which offers the possibility to do your semester or masters thesis abroad at MIT. ✦

High-Tech am Zürichsee

C#, Python, agile Entwicklung und UML sind wichtige Schlüssel für unser starkes Wachstum. Starte deine Karriere in einer professionellen und dynamischen Umgebung.

www.sensirion.com

SENSIRION
THE SENSOR COMPANY

Interview: Michel Ott, Accenture

JASCHA GRÜBEL

Im Rahmen der VIS-Vortragsreihe im letzten Herbst war Accenture an der ETH. Leider war ich zu diesem Zeitpunkt durch mein Studium verhindert. Anfang Februar hatte ich jedoch die Möglichkeit, Accenture an ihrem Firmensitz in Zürich zu besuchen und ein Interview mit Herrn Ott zu führen, der seit 12 Jahren bei Accenture arbeitet.

Guten Morgen, Herr Ott. Um den Lesern der Visionen eine Vorstellung von Accenture zu geben, würde ich Sie bitten, in Ihren Worten Accenture zu beschreiben.

Accenture ist ein Managementberatungs, Technologie- und Outsourcing-Dienstleister. Das heisst, wir unterstützen Unternehmen Ideen, Visionen und Strategien zu erarbeiten und umzusetzen. Unsere Arbeit ist erst dann abgeschlossen, wenn jede Kleinigkeit im alltäglichen Einsatz funktioniert. Wir begleiten also ganze Veränderungs- und Transformationsprozesse von A bis Z, damit die Unternehmen die gesteckten Ziele erreichen und am Markt erfolgreich sind. In diesem ganzheitlichen Ansatz steckt unsere grosse Kompetenz.

Michel Ott

Können Sie für unsere Leser Ihre Arbeit bei Accenture beschreiben und wie würden Sie Ihre Arbeitserfahrung bei Accenture zusammenfassen?

Accenture gliedert sich grob in fünf Branchengruppen: 'Financial Services', Products', 'Resources', 'Communications & High-Tech' und 'Health & Public Services'. Ich bin Partner im Bereich 'Financial Services' und dort zuständig für den Versicherungsbereich. Ich unterstütze Versicherungen in Transformationsprojekten, das heisst Reorganisation und Prozessveränderung, aber auch Einführungen von neuen Systemen, die die Geschäftsstrategie von Unternehmen optimal unterstützen. Der Fokus dabei liegt auf IT. Ein Beispiel: Kürzlich haben wir bei einem Versicherer das rund dreissigjährige Kernversicherungssystem ersetzt, weil es die Geschäftsstrategie nicht mehr vollständig unterstützte. Das alte System hat die Hand-

lungsmöglichkeiten des Kunden stark limitiert; das haben wir ausgemerzt.

Da haben Sie uns gerade einen schönen Einblick in ihren Arbeitsalltag gegeben und da würde ich auch gerne weitermachen und Sie fragen, was bisher ihr grösster Erfolg bei Accenture war.

Da gäbe es einiges! Ich erinnere mich gerne an ein Projekt, kurz nach meinem Start bei Accenture. Es ging darum, im Versicherungsbereich eine Schadensreorganisation zu machen. Ich war für das Design von einem bestimmten Prozess verantwortlich. Die Mitarbeiterinnen und Mitarbeiter des Kunden wurden dann mit den neuen Prozessen vertraut gemacht und darauf trainiert. Das Schöne daran: Ein Jahr nach Abschluss der Arbeiten zeigte eine Kundenzufriedenheits-Umfrage dieses Versicherers, dass wir entscheidend mitgeholfen haben, die Zufriedenheit massiv zu steigern. Das Unternehmen war effizienter aufgestellt und die Mitarbeiter haben realisiert, dass sie mit den optimierten Prozessen und dem System ihre Kunden besser bedienen konnten. Vom Kunden des Kunden positives Feedback zu hören, ist eine ganz tolle Sache. Das schafft unglaubliche Befriedigung. Haben die Unternehmen hinterher wirklich ihr Ziel erreicht? An dieser Frage messen wir unsere Leistungen. Wenn wir hier ja sagen, dann ist das jedes Mal auch ein sehr grosser persönlicher Erfolg. Was das vorherige Beispiel betrifft: Ich war damals noch sehr jung; es war mein erstes Grossprojekt und ich konnte viel Verantwortung übernehmen. Das war sehr motivierend.

«Vom Kunden des Kunden positives Feedback zu hören, ist eine ganz tolle Sache.»

Über Accenture

Accenture ist ein weltweit agierender Managementberatungs-, Technologie- und Outsourcing-Dienstleister mit rund 215.000 Mitarbeitern, die für Kunden in über 120 Ländern tätig sind. Das Unternehmen bringt umfassende Projekterfahrung, fundierte Fähigkeiten über alle Branchen und Unternehmensbereiche hinweg und Wissen aus qualifizierten Analysen der weltweit erfolgreichsten Unternehmen in eine partnerschaftliche Zusammenarbeit mit seinen Kunden ein. Accenture zählt derzeit 94 Unternehmen der Global Fortune 100 und in der Schweiz 13 der 20 SMI-Unternehmen zu seinen Kunden. Weltweit wird alle vier Stunden ein Accenture-System in Betrieb genommen. Accenture erwirtschaftete im vergangenen Fiskaljahr (zum 31. August 2010) einen Nettoumsatz von 21,6 Mrd. US-Dollar. Die Internetadresse lautet www.accenture.ch.

Ich kann mir sehr gut vorstellen, dass das ein Erfolgserlebnis war. Ich möchte nun auf die Informatik zu sprechen kommen. Was bietet Accenture Informatik-Absolventen der ETH im Besonderen? Was macht Accenture für uns Informatiker attraktiv?

Grosse Abwechslung und verantwortungsvolle Tätigkeiten, das macht den Einstieg bei Accenture interessant!

Ich habe bereits auf die fünf übergeordneten Branchengruppen hingewiesen. Damit decken wir 17 Branchen mit unseren Dienstleistungen ab. In den Tätigkeitsfeldern, die wir anbieten, geht es von Managementberatung über Outsourcing bis zur Technologieberatung, aber auch Systemintegration und Programmierung. Die Letzteren sind sicherlich für ETH-Absolventen sehr interessant. Wenn ein Mitarbeiter sich für uns interessiert und bei uns anfängt, absolviert er ein Trainingsprogramm, wo er lernt, was er für

➔

seine Arbeit bei Accenture braucht. Gewisse Bereiche haben sogenannte Jump-Start-Programme. Dort wird zwei Monate lang intensiv trainiert. Diesen Einstieg bieten wir zum Beispiel für Java, SAP an.

Wir suchen Mitarbeiterinnen und Mitarbeiter, die für solche Herausforderungen bereit sind. Im deutschsprachigen Raum werden wir in diesem Fiskaljahr (Anm.: 1.9.10-31.8.11) rund 1400 Personen einstellen. Das sind allein in der Schweiz über 300, von denen viele über ein Studienprofil der ETH- und EPFL verfügen. Entsprechend wichtig sind uns die beiden Institute.

Sie sind sehr schön auf Ihre Verantwortung im Unternehmen eingegangen. Wie sieht es im Allgemeinen aus mit der Selbstentfaltung der Mitarbeiter? Wie viel Eigeninitiative wird von den Mitarbeitern gefragt, vielleicht sogar gefordert?

Eigeninitiative ist ein zentraler Punkt. Wir sind ein Dienstleistungsunternehmen, das selbstredend für Kunden arbeitet und diese Kunden erwarten, dass Accenture-Mitarbeiter Eigeninitiative zeigen und optimale Lösungen für die Kundenbedürfnisse erarbeiten. Wir schiessen am Ziel vorbei, wenn wir einfach ein vordefiniertes Programm abliefern und dem Kunden vorsetzen. Wir prüfen immer: Macht jeder einzelne Schritt Sinn? Können wir noch weitere Vorschläge einbringen? Ich will in solchen Situationen keine Probleme vom Team hören, sondern Lösungsvorschläge. Nur das bringt uns weiter. Vielleicht ist es nicht der Lösungsvorschlag, der am Ende umgesetzt wird, aber er regt zum Denken an. Bei uns geht es um die Verbindung von Technologie und Business. Das heisst, ich will Technologie nicht um der Technologie willen einsetzen, sondern um ein Geschäftsziel zu erreichen. Man muss also immer wieder kritisch fragen: Kann ich das Ziel, das mir das Business gesetzt hat, mit dieser Technologie erreichen?

Sie sind bisher auf die allgemeinen Aufgaben eines Informatikers bei Accenture eingegangen, wie sieht denn der Alltag eines Informatikers bei Accenture aus?

Die Arbeit ist sehr abwechslungsreich, weshalb es den „typischen Alltag" so nicht gibt. Was bei allen Projekteinsätzen gleich ist: Alles ist auf einen Kunden ausgerichtet. Man muss umsetzen, was dem Kunden letztlich hilft, ein Businessziel zu erreichen. Abhängig von der Phase, in der man sich gerade befindet, zum Beispiel beim Design einer Software zur Schadenssachbearbeitung, muss man genau identifizieren: Was sind die Bedürfnisse? In Workshops erarbeitet man in gemischten Teams mit dem Kunden einzelne Funktionen und bestimmt, wie diese aussehen sollen. Ist der Vorgang abgeschlossen, beginnt man mit Programmieren und zwar so, wie man es mit dem Kunden vereinbart hat. Man geht dann sein Arbeitspaket Schritt für Schritt durch und hinterher wird zusammen mit dem Kunden die Applikation getestet. Massgabe dafür ist die Frage, ob die Applikation schliesslich erfüllt, was mit dem Kunden zuvor vereinbart wurde? Da geht es um fachliche Themen, aber auch um technische Fragen wie zum Beispiel Performance. Läuft alles so, wie man es sich vorgestellt hat? Mit Testscripts und Testzyklen überprüft man, was erarbeitet wurde. In Abhängigkeit von der Projektausgestaltung führen wir die Testscripts selbst aus oder der Kunde übernimmt dies und wir unterstützen ➔


BE YOU IMAGINED
Lernen Sie ein Unternehmen kennen, das Ihnen mehr Chancen, Herausforderungen und Zufriedenheit bietet. Ein Unternehmen, das auf Teamwork und Zusammenarbeit setzt. Ein Unternehmen, das Technologietrends mitgestaltet und in der Schweiz, Deutschland und Österreich Top-Unternehmen hilft, ihr Geschäft neu zu erfinden. Zum Beispiel bei 13 der 20 SMI-Unternehmen. Unser Spektrum ist so breitgefächert, dass Sie sogar den Job wechseln können, ohne das Unternehmen zu wechseln. Sprechen Sie mit uns und entdecken Sie Ihre Möglichkeiten.
entdecke-accenture.ch
accenture
High performance. Delivered.
BE GREATER THAN
©2011 Accenture. All rights reserved.

bzw. koordinieren die Tätigkeiten. Tritt ein Fehler auf, so muss man koordinieren, dass der Programmcode rechtzeitig und qualitativ hochwertig überarbeitet und erneut getestet wird. Zur Arbeit gehört das Management dieses gesamten Ablaufs.

Accenture ist ein international sehr gut aufgestelltes Unternehmen, das weltweit agiert. Welche Arbeitsmöglichkeiten bieten sich da, wenn man bei Accenture angestellt ist, nicht nur an einem Ort zu arbeiten, sondern auch im Ausland?

In der Schweiz haben wir eine Niederlassung in Zürich und zusätzlich sogenannte „Virtual Offices" in Basel, Bern und Genf. Diese geben uns und unseren Mitarbeiterinnen und Mitarbeitern die Möglichkeit näher bei den Kunden zu sein. Wenn ein neues Projekt beginnt, so erfolgt die personelle Besetzung nach Möglichkeit lokal in Abhängigkeit von der Verfügbarkeit von Mitarbeitern mit den entsprechenden Fähigkeiten und Kenntnissen. Für kurzfristige Projekte im Ausland kommen auch Österreich und Deutschland in Frage. Wenn sich die Mitarbeiterin oder der Mitarbeiter aber besondere Fähigkeiten erwirbt oder über solche verfügt, die in anderen Märkten gefragt sind, dann gibt es Möglichkeiten, ausserhalb des deutschsprachigen Raums eingesetzt zu werden. Entscheidend ist dann die Fachexpertise.

Vielen Dank für Ihre Ausführungen. Ich würde gerne noch eine abschliessende Frage stellen. Was würden sie einem ETH-Absolventen, der auf der Arbeitssuche ist, mit auf den Weg geben wollen?

Sehr wichtig ist: Der CV darf nicht 0815 sein. Er muss klar machen, weshalb jemand diese Stelle will und warum diese Person besonders gut passt. Entsprechend muss man den CV und das Bewerbungsschreiben ausrichten. Wichtig ist dabei, sich ausgewogen und authentisch darzustellen; keine falsche Bescheidenheit, keine Übertreibungen. Ausgewogen heisst auch, dass man neben den Studienleistungen zeigen kann, welche ersten Berufserfahrungen man schon im In- und Ausland gemacht hat. Ausserdem ist es gut, wenn man aufweisen kann, dass man schon bereit war Verantwortung zu übernehmen, bspw. in studentischen Organisationen. Das kann für eine Anstellung durchaus ausschlaggebend sein.

Wenn man eine Anstellung gefunden hat, ist es das Wichtigste, authentisch und neugierig zu bleiben. Es kommt beim Arbeitgeber immer gut an, wenn man seine Begeisterung offen zeigt. Man muss aber auch immer bereit sein zu lernen und nachzufragen, wenn etwas unklar ist.

Im Namen der Leser von Visionen bedanke ich mich ganz herzlich für das Interview.

Sehr gern geschehen, es war mir eine Freude.

✦

Student Exchange Program

Accenture bietet ein „Student Exchange Program" an. Auch dieses Frühlingssemester konnten sich Studierende für das elfwöchige Programm bewerben. Wer ausgewählt wird, kann bei Mitarbeitern von Accenture in sogenannten Off-Shore-Locations – z.B. Indien – ein Praktikum machen, um zu sehen, wie eine solche Zusammenarbeit genau funktioniert. An der ETH arbeitet Accenture mit Herrn Professor Kossmann zusammen, der das Programm seitens der Hochschule begleitet. Das Programm wurde nun zum vierten Mal durchgeführt und auch dieses Jahr gingen zahlreiche Bewerbungen dafür ein. Das Programm gilt auch für Alumni der ETH.

Bearicatures

ALEX MEUCHE — TRIED OUT A NEW COLORING STYLE

Jürg Gutknecht

«Bearknecht»

QUOTE:

«Probieren Sie es! Es ist mühsam, unschreibbar und am Schluss unlesbar!»

HAPPENING:

Animations are to be made with at least 20 Powerpoint Slides.

RATING:

★★★★★★☆☆☆☆

Sometimes good, sometimes unexpressive slides…

✦

The Copy-Paste Pattern:

History and insights into the modern and often resorted to programming pattern—or art.

ANDREAS BRAUCHLI — CASUAL COPY-PASTER

Programming patterns are in every coder's mouth nowadays—books reaching almost as far back as the byte itself describe them in all gory details. This article digs into this most spread but rarely documented pattern and sheds some light on the darker sides involved in its usage or abuse.

Coders are generally renown for their epic laziness. As such they tend to rip code from dark corners of the web to put into their own software thereby making use of said copy-paste pattern. It gets more funny, or sad, when a self-proclaimed programmer has no idea about what he's actually copying. Assuming he can magically make it work.

It's been hard to determine the exact origins of the pattern due to its rapid spreading, but it's, for some unknown reason, widely believed to have Indian roots. Even though there seems to be virtually no written documentation about its origin or existence it is still seen in practically all software. Sometimes it even appears to be much more than a simple pattern but rather a design philosophy, a way of life and from time to time also a cheap means to fast money.

Who wonders why the overall quality of today's software is about the quality of an iPhone™ app from the iPhone™ app store™? The problem of the latter being a deadly overdose of the former and certainly a rich portion of abuse of this precise pattern.

Interestingly the pattern doesn't solely apply to programming and software but rather is embezzled the whole academic world despite all morals and high standards. Bertrand Meyer hit the nail on the head when he blogged: *«1440: Gutenberg discovers the secret of producing, out of one text, many books for the benefit of many people. 2007: Von und Zu Guttenberg discovers the secret of producing, out of many texts, one book for the benefit of one person.»*

Note that your author is not generally seeing the copy-paste pattern as a source of evil, as it

All you need.

might seem, but rather documents its life and intends to point out the issues resulting from overly abuse. There are no doubt legitimate cases where the usage is totally justified and almost inevitable like in boilerplate code.

Summarizing, we can state that over usage of the copy-paste pattern may—and abuse will—result in hard consequences for modern software being drastic loss of quality at the benefit of drastic increases in the number of software applications, lines of code per software and thus bugs for the user. Look forward to a bright copy-paste future. Look forward to a bright copy-paste future. Look forward to a bright copy-paste future. ✦

Bildnachweise

Umschlag: Hieroglyphs – Leiden Museum of Antiquities (NL) 2008; by Urville Djasim; www.flickr.com/photos/urville_djasim; CC 2.0 by-nd

Seite 39: person different from the crowd; © Mike Kiev - Fotolia.com

Beni Koller: Klassenunterschied

MICHAEL GROSSNIKLAUS — EIN MONUMENT.

Benis Klassentreffen findet im wunderschön gelegenen Waldhotel am See statt. Als Beni und Anina mit ihrem VW Passat dort eintreffen, ist der Parkplatz schon gut gefüllt. Einzig zwischen einem brandneuen Audi R8 und einem rostigen Honda Civic findet Beni noch ein leeres Feld. „Sieht aus, als wären wir die Ersten!" spottet Beni ironisch, nachdem er sich aus dem Wagen gewunden hat. Vor lauter Angst, den Audi zu beschädigen, hatte er seine Türe nur einen Spalt weit geöffnet. „Kein Wunder, bei diesem Grossmutter-Tempo, das du fährst!" kommt sofort die gehässige Antwort von Anina. Es war nicht die erste Stichelei, die sie heute in seine Richtung absetzte und Beni war nicht ganz klar, was genau los war. Er versucht, sie aufzumuntern und scherzt: „Die Rostlaube auf deiner Seite ist sicher Erich Formanns letztes Eigentum." Wortlos zieht Anina ihren Rollkoffer aus dem Auto und geht in Richtung Rezeption davon.

Nachdem Beni und Anina eingecheckt und ihr Zimmer bezogen haben, steht als erstes ein Willkommensapéro auf der grosszügigen Hotelterrasse auf dem Programm. Beim Betreten der Terrasse lässt Beni seinen Blick in alle Richtungen schweifen. Es ist jedoch nicht die atemberaubende Aussicht auf den See, die es ihm angetan hat, sondern der Versuch, Erich Formann in der Menge zu entdecken. Beni erblickt ihn in einer gut gelaunten Gruppe, die sich bei einer riesigen Topfpflanze in der Nähe des Geländers gebildet hatte. Sofort legt Beni seinen Arm energisch um Anina und geht, Anina mit sich reissend, schnurstracks auf die Gruppe zu.

„Besser als die Büchsenravioli deiner Mutter?" greift er Erich an, der sich gerade ein Lachscanapée in den Mund schiebt. „Hallo Beni! Schön, dass du auch hier bist!" antwortet ihm Erich und fährt fort: „Ich wohne allerdings nicht mehr bei meiner Mutter". „Notschlafstelle?" faucht Beni zurück. „Nein, nein, ich bin in eine Fünfzimmerwohnung in der Stadt gezogen", erwidert Erich und erklärt dem sichtlich enttäuschten Beni: „Nachdem ich letzte Woche eine leitende Stelle bei einer Investmentfirma angeboten bekommen habe, kann ich mir das leisten". Etwas verdutzt stottert Beni: „Das ist übrigens meine Freundin Anina". Anina streckt Erich ihre Hand hin und benutzt diese Geste, um sich aus Benis enger Umarmung zu lösen. Erich küsst Aninas Handrücken und seufzt: „Ich wünsche euch all das Glück in der Liebe, das mir verwehrt geblieben ist". Beni grinst: „Jawohl!" und triumphiert: „Wenigstens bei den Frauen geht

die Runde klar an mich!" Erich schaut ihn verständnislos an und wirft dann einen besorgten Blick in Richtung Anina. Diese hat Tränen in den Augen und entschuldigt sich: „Ich habe Kopfweh und gehe wohl besser aufs Zimmer ein Aspirin nehmen". Beni will sie begleiten, doch sie erklärt ihm, sie brauche ihre Ruhe und er könne nichts für sie tun.

Beni hat keine Lust mehr, sich mit Erich abzugeben und gesellt sich deshalb zu der Gruppe, in der sich die Burkhart Zwillinge und Stefan Heinz unterhalten. „Heinz und die Burkhart-Zwillinge, wie geht es euch?" ruft ihnen Beni von weitem zu. „Sehr gut!" antwortet ihm Heinz und holt aus: „Meine Kaffeerahmdeckelsammlung kam ins Guiness-Buch der Rekorde und ich leite seit einigen Jahren das Marketing des Schweizer Milchverbandes". „Ausserdem hat er gut geheiratet: nämlich mich!" fährt ihm Johanna Burkhart ins Wort. „Und wie geht's dem Schwesterherz?" will Beni nun wissen. „Mir geht es auch gut", antwortet ihm Elisabeth Burckhardt und erkundigt sich: „Dir ist aber schon klar, dass wir nicht wirklich verwandt sind und dass diesen Running Gag schon in der Schule niemand lustig fand?" Beleidigt wendet sich Beni wieder Heinz zu, der auf seinem iPhone herumdrückt. „Hast du dir das iPhone gekauft, damit du den schlechten Empfang verantwortlich machen kannst, wenn dich niemand anruft?" giftelt er ihm zu. Nun reisst Johanna die Geduld: „Weisst du was Beni? Lass uns doch einfach in Ruhe!" Und Elisabeth doppelt nach: „Deine verletzenden Spässe kamen noch nie gut an! Wenn du andere klein machen musst, um dich grösser zu fühlen, dann tu dies doch bitte woanders".

Gerade als Beni die Türe zum Hotelzimmer öffnen will, hört er ein wehmütiges Stöhnen aus dem Innern und tritt sofort besorgt ein. Auf den ersten Blick versteht Beni nicht, was los ist und starrt gelähmt auf die zwei Silhouetten, die sich unter den Leintüchern winden. Als er sich räuspert, erscheint ein Arm und zieht das Leintuch zurück. Beni erschrickt, als darunter Anina und Erich zum Vorschein kommen. Plötzlich ist es Beni hundeübel, er dreht sich um, stürmt aus dem Hotelzimmer, den Gang hinunter und durch die Lobby hinaus ins Freie, wo er sich nicht unweit der grossen Eingangstreppe übergibt. „Ich bin wohl nicht der einzige, der zu viel getrunken hat", bemerkt eine schwer verständliche Stimme mit starkem Akzent. Es ist Toshiki Nakata, der japanische Austauschschüler, der extra für das Klassentreffen zurück in die Schweiz gereist war. ✦

(Fortsetzung folgt…)

Impressum

VISIONEN

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

Ausgabe Juni 2011

Periodizität	6x jährlich
Auflage	1400
Jahresabonnement	CHF 25.–

Chefredaktion
Rudolf Maximilian Schreier
visionen@vis.ethz.ch

Layout
Daniel Saner
layout@vis.ethz.ch

Inserate
Adrian Blumer
inserate@vis.ethz.ch

Anschrift Redaktion & Verlag
Verein Informatik Studierender (VIS)
CAB E31
Universitätsstr. 6
ETH Zentrum
CH–8092 Zürich

Inserate (4-farbig)

½ Seite	CHF 850.–
1/1 Seite	CHF 1500.–
1/1 Seite, Umschlagsseite (U2)	CHF 2500.–
½ Seite, Rückumschlag (U4)	CHF 2500.–

Andere Formate auf Anfrage.

Lektorat
Benjamin Ernst
Mark Nevill

Redaktion
Tobias Heinzen
Fabian Hahn
Thorben Bochenek
Jascha Grübel
Carla Hofer
Alexandra Meuche

und freie Mitarbeiterinnen und Mitarbeiter

Druck
Binkert Druck AG
5080 Laufenburg
http://www.binkert.ch/

VIS

VSETH
VERBAND DER STUDIERENDEN AN DER ETH

Der VIS ist Teil des Verbandes der Studierenden an der ETH (VSETH).

Puzzled (Solutions)

BARBARA KELLER — DOES NOT WANT TO STOP (EATING CHOCOLATE)

ROGER WATTENHOFER — STOPPED THINKING ABOUT A TAG LINE BEFORE ACTUALLY HAVING ONE

The games in the last issue were variants of an important question in real life: When is the right time to stop? In the first game the player can bet on the next card being black, starting with a card deck with 26 red and 26 black cards. One card after the other is revealed to the player. If she says stop, she bets on the next card being black. It may seem that the player can gain an advantage if she waits for the first moment when the remaining black cards outnumber the remaining red cards. But does this small advantage compensate for potential losses? Surprisingly not—the game is completely fair. Neither can the player increase her chance of winning, nor can she diminish it. This game is best understood by slightly modifying it: Instead of betting on the next card, the player bets on the very last card. As the last card has at every moment the same probability of being black as the next card, any strategy of the player should lead to the same advantage in the modified game. However, the modified game is quite boring, as the probability of the last card being black is exactly 50%, independent of any strategy.

In the second game, an adversary chooses two cards of the same suit from a Swiss card deck. The player can sample one of the cards, and must then decide whether the still hidden card is higher than the card that was revealed. Contrary to intuition, a player can gain an advantage in this game. To simplify our explanations, we assign the numbers 11 to 14 to the cards jack, queen, king and ace, so the original Swiss deck consists of the cards 6 to 14. Here is a winning strategy: The player randomly inspects c, one of the two adversarial cards. Then the player chooses a random value r out of $\{6.5,7.5,8.5,9.5,10.5,11.5,12.5,13.5\}$. If $r > c$, the player does not stop, and bets that the still hidden card is higher. If $r < c$, the player stops and bets on the already seen card to be higher. It's simple to see that this works: If r is smaller (or larger) than both cards, the probability that the randomly inspected card c was the larger of the two cards is exactly 50%. However, if the randomly chosen value r is between the two cards, the player wins for sure! Since the player chooses r in between any two cards with probability at least 1/8, we get a total winning probability of at least $9/16 > 50\%$. ✦

AZB
PP/Journal
CH – 8092 Zürich

Falls unzustellbar, bitte zurück an:
Verein Informatik Studierender
CAB E31
Universitätsstr. 6
ETH Zentrum
CH-8092 Zürich