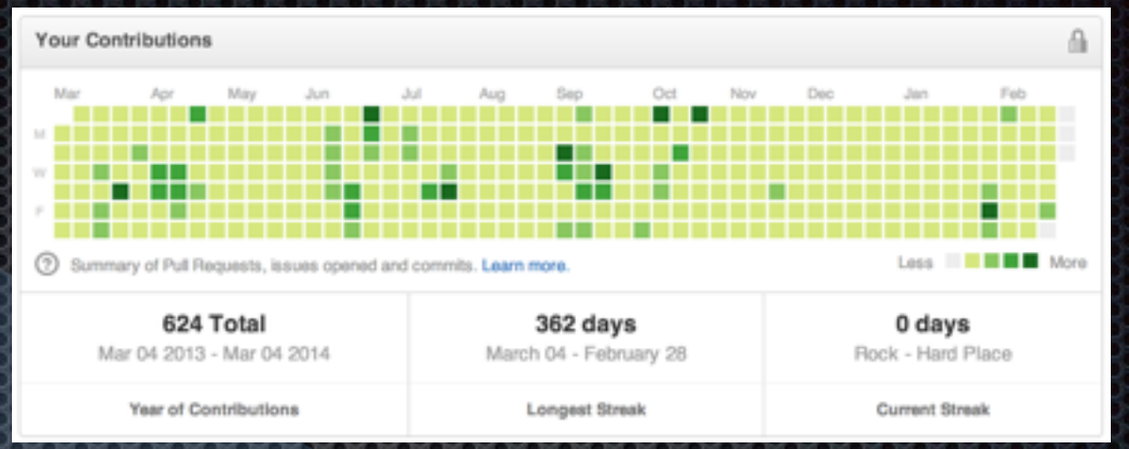
DAY OF THE MASTER

DR. VADIM ZAYTSEV

AKA GRAMMARWARE

Introduction



- Universiteit van Amsterdam (2013–2014)
- Centrum Wiskunde & Informatica (2010–2013)
- Universität Koblenz-Landau (2008–2010)
- Vrije Universiteit Amsterdam (2004–2008)
- Universiteit Twente (2002–2004)
- Rostov State Transport University (1999–2008)
- Rostov State University (1998–2003)

Vadim Zaytsev

SOFTWARE ENGINEERING

- One year Master of Science programme at UvA
- Drifted away from computer science
- We teach software construction, evolution, testing, architecture, process, requirements engineering, etc
- Programmer in, software engineer out

http://www.software-engineering-amsterdam.nl

From code-monkey...

Ogrons in *Day of the Daleks*
http://anewviewonolddoctorwho.files.wordpress.com/2013/01/ogrons.png

...TO THE MASTER

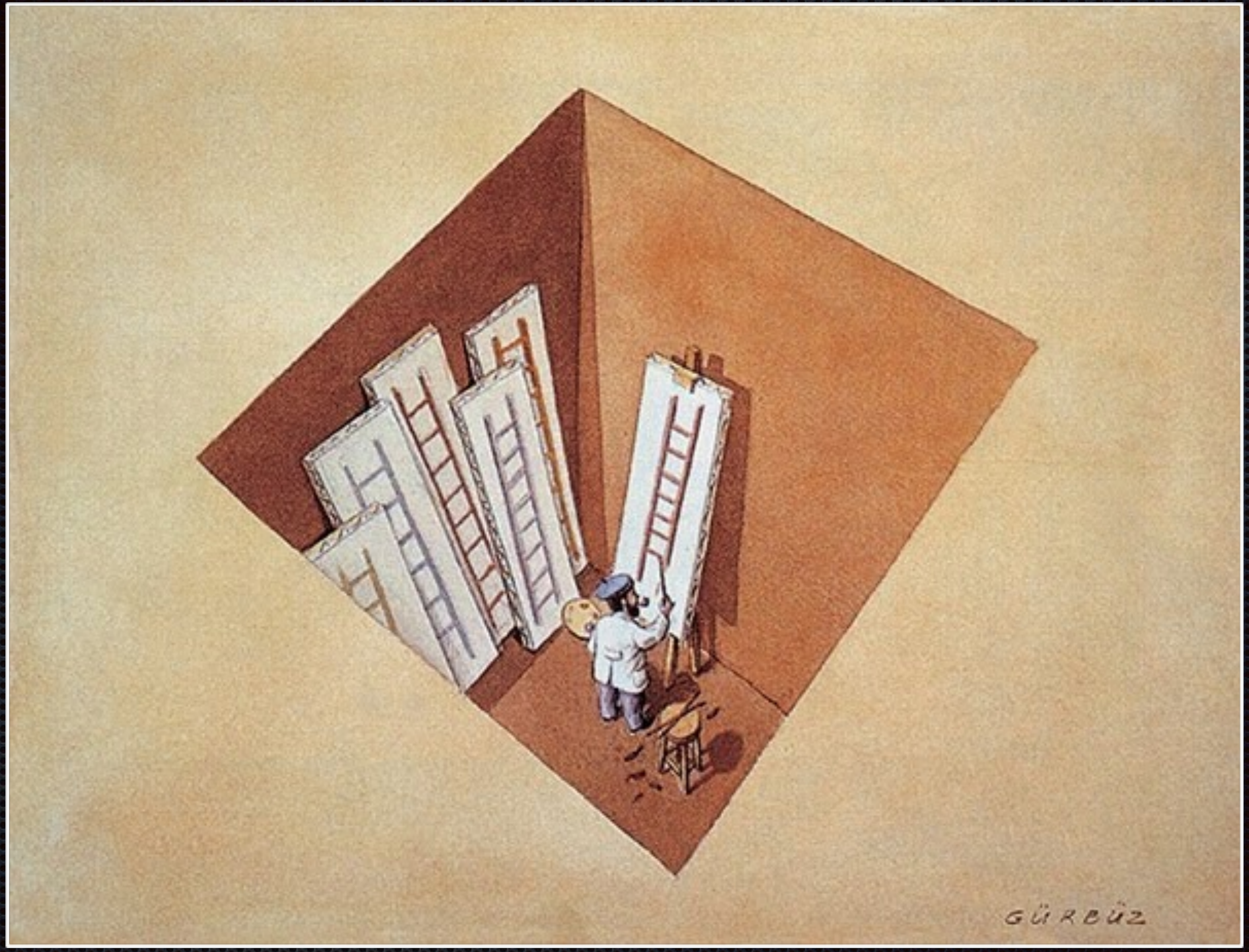
Roger Delgado as The Master in *The Claws of Axos*
http://www.eyeofhorus.org.uk/images/photo/03pertwee/clawsaxos/master-delgado.jpg

ENGINEERING?

- Science
  - solves problems
- Engineering
  - solves problems

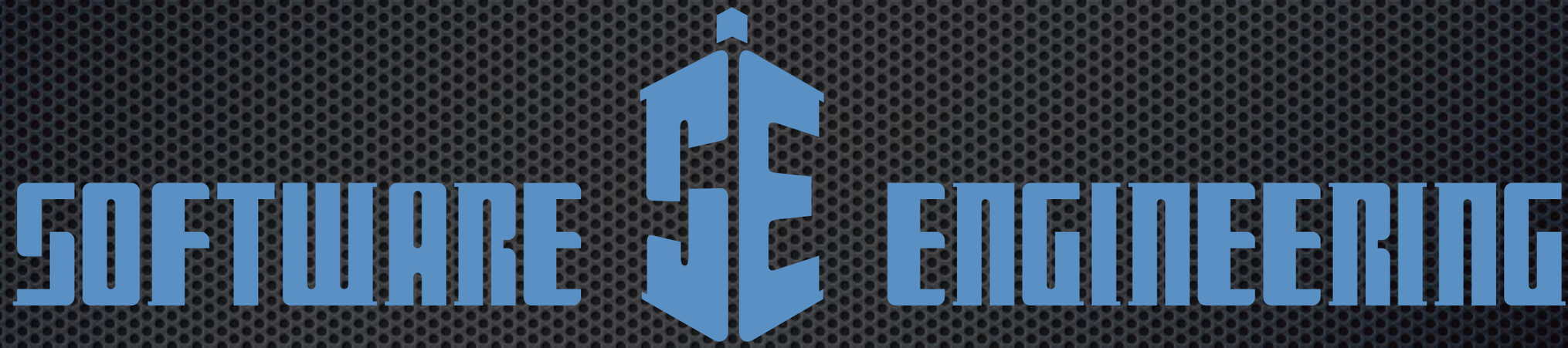
Gürbüz Doğan Ekşioğlu, http://www.gurbuz-de.com/merdivenler-e.html

Gürbüz Doğan Ekşioğlu, http://www.gurbuz-de.com/merdivenler-e.html



Gürbüz Doğan Ekşioğlu,
http://markovart.wordpress.com/2014/01/03/surrealism-by-gurbuz-dogan-eksioglu/

WHAT IS IMPORTANT IN

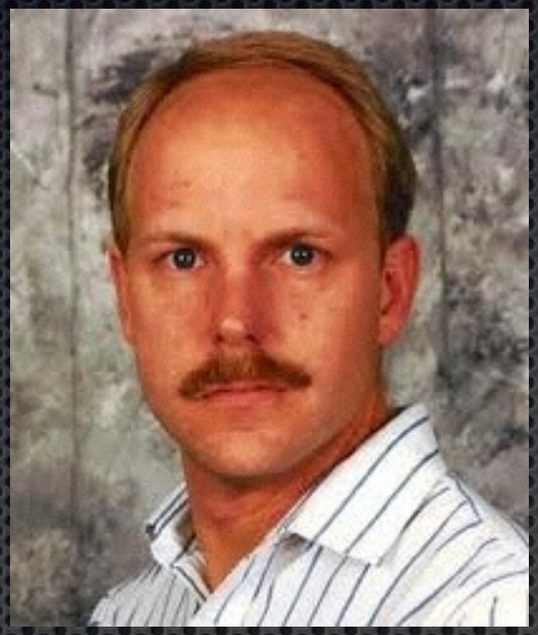


?

What's important?

- Domain analysis
- Educated choices
- Tradeoff awareness
- Human factors
- Communicating with management

(collected during the workshop)



"I'm not a great programmer, I'm just a good programmer with great habits"

–Kent Beck

Martin Fowler, Kent Beck, John Brant, *Refactoring: Improving the Design of Existing Code*, 2012. P. 97.

SOFTWARE ENGINEERING

- One year Master of Science programme at UvA
- Drifted away from computer science
- We teach software construction, evolution, testing, architecture, process, requirements engineering, etc
- Programmer in, software engineer out

http://www.software-engineering-amsterdam.nl

CODING DOJO

Zurfa, Hacker Dojo - Main Classroom, CC-BY-SA, 2013.

Wang Ming, Noma Dojo, 2006, CC-BY-SA, 2007.

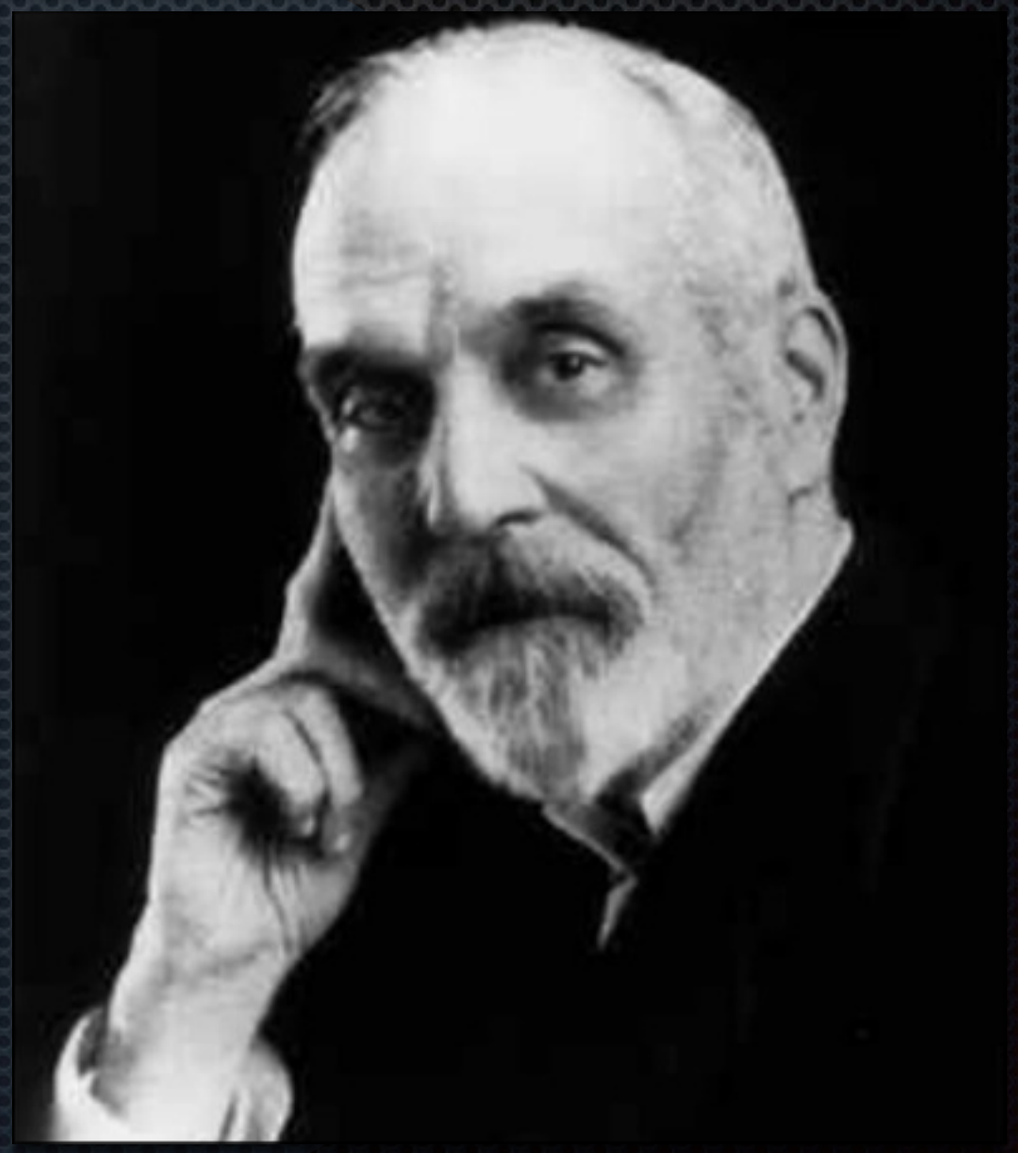
The Doctor fencing with The Master, *The Sea Devils*, s09e03.

The Doctor fencing with The Master, *The Sea Devils*, s09e03.

WARM-UP!

Henry Ernest Dudeney

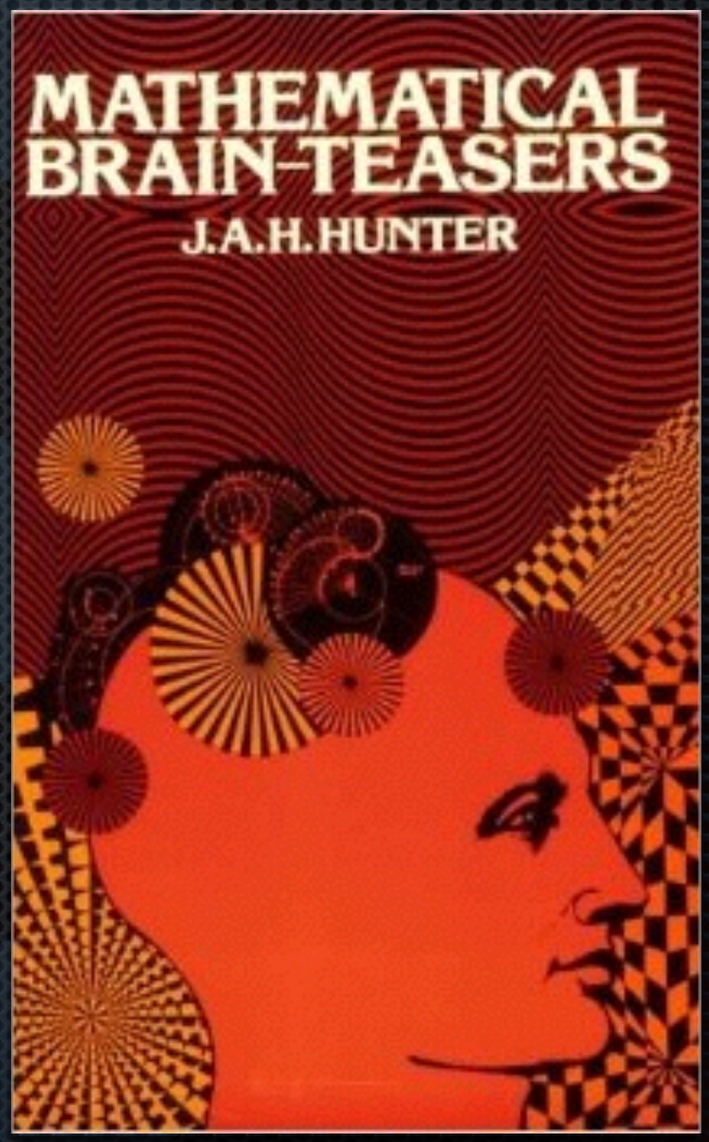
- Recipe (1924):
  - take a numerical calculation (2*2=4)
  - replace digits by letters (A*A=B)
- Results in:
  - cryptarithm



http://www.cut-the-knot.org/cryptarithms/st_crypto.shtml
http://en.wikipedia.org/wiki/File:Henry_Dudeney.jpg

James Hunter, 1955

- Cryptarithm
  - with numbers as meaningful words
  - equations as meaningful phrases
- Results in
  - alphametic



http://www.cut-the-knot.org/cryptarithms/st_crypto.shtml
http://www.amazon.com/Mathematical-Brain-Teasers-James-H-Hunter/dp/0486233472

SEND MODE MONEY!

9567

+ 1085

10652

SEND

+

MORE

MONEY

NO GUN NO HUNT!

87

908

87

1082

NO

GUN

NO

HUNT

WILL OBEY DALEK!

6099 + 7825 = 13924

WILL + OBEY = DALEK

EXTERMINATE! EXTERMINATE!

EXTERMINATE

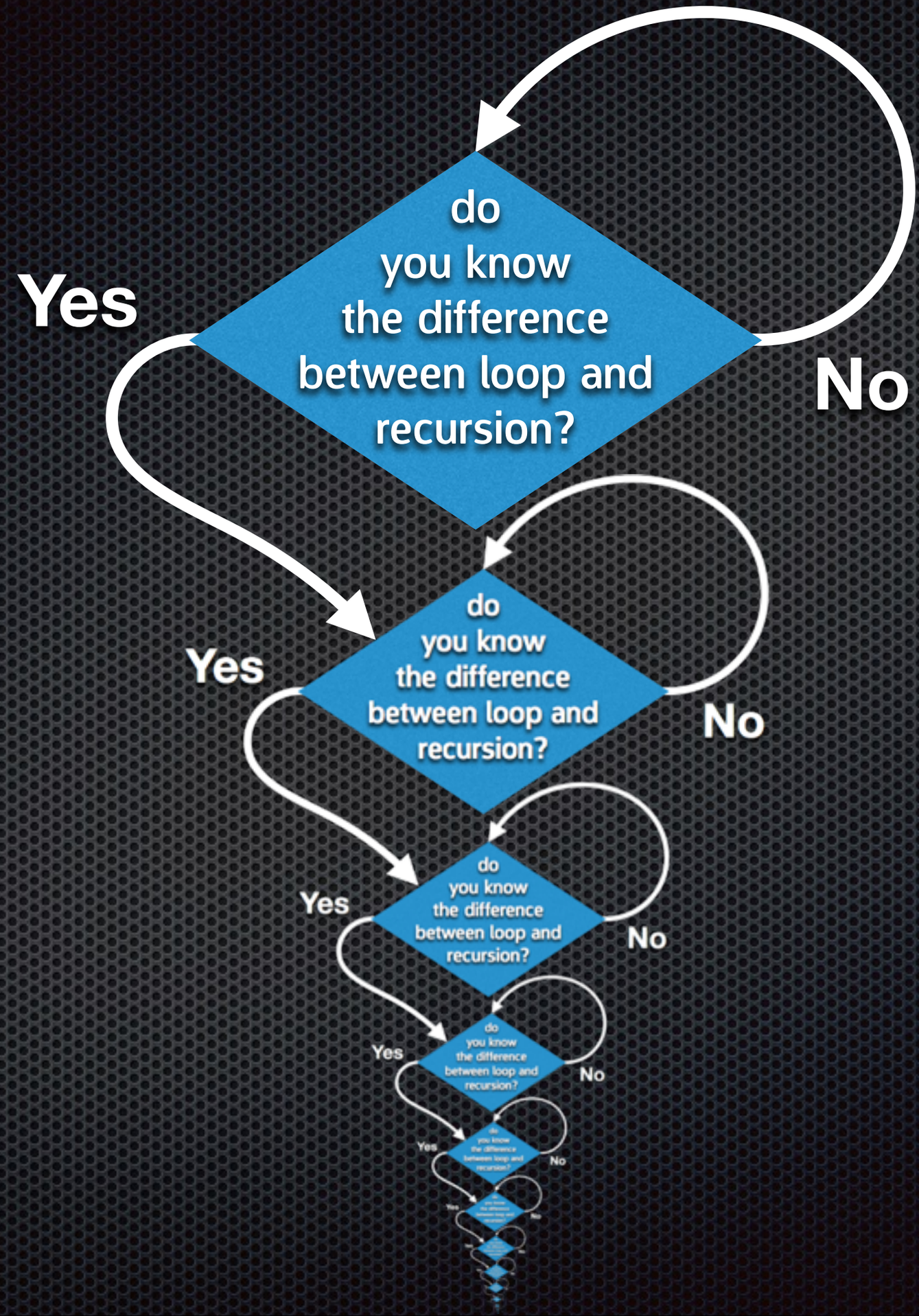
+

MONEYMAKING

CRYPTARITHM

Tasks

- Find a solution of an ~~alphametic cryptarithm~~ puzzle
- Given a puzzle, find a solution
- Given a puzzle and a solution, check compatibility
- Find a puzzle with only one solution
- Given a desired word, find valid puzzles


do
you know
the difference
between loop and
recursion?
Yes
No
do
you know
the difference
between loop and
recursion?
Yes
No
do
you know
the difference
between loop and
recursion?
Yes
No
do
you know
the difference
between loop and
recursion?
Yes
No

```
ds = {*[0..9]};
for (str solution <- {"          <N> <O>
                      '       <G> <U> <N>
                      '             <N> <O>
                      '----------------------
                      '  <H> <U> <N> <T>" |
      int G <- ds,
      int H <- ds - {G},
      int N <- ds - {G,H},
      int O <- ds - {G,H,N},
      int T <- ds - {G,H,N,O},
      int U <- ds - {G,H,N,O,T},
      G != 0, H != 0, N != 0,
      (O + 10 * N) +
      (N + 10 * U + 100 * G) +
      (O + 10 * N) ==
      (T + 10 * N + 100 * U + 1000 * H)})
println(solution);
```

```
str gen(list[str] xs)
{
	keys = sort({x | /str s <- xs, int x <- chars(s)});
	int width = 4*max([size(s) | /str s := xs])+3;
	f = "module Solver
	'import IO;
	'void solveit(){
	'ds = {*[0..9]};
	'for (str solution \<- {\""
	+ intercalate("
	'		\",
	[right(intercalate(" ",["\<<stringChar(c)>\>" | int c <- chars(s)]),width) | s <- sx[..-1]])+ "
	'		\"+
	right("",width,"-")+"
	'		\"+
	right(intercalate(" ",["\<<stringChar(c)>\>" | int c <- chars(xs[-1])]),width)+"\" | \n";
	visited = [];
	for (k <- keys)
	{
		f += " int <stringChar(k)> \<- ds - {<intercalate(",",visited)>},\n";
		visited += stringChar(k);
	}
	notzeros = sort({chars(s)[0] | /str s <- xs});
	f += "\t"+intercalate(", ",["<stringChar(c)> != 0" | c <- notzeros]) +
	";
	'	<intercalate(" +\n",["(<factorise(s)>)" | s <- xs[..-1]])> ==
	'	(<factorise(last(xs))>)})
	'	println(solution);
	'}
	'public void main(list[str] args) {solveit();}";
	println(f);
	return f;
}
```

Helping observations

- Leftmost letters cannot be 0
- The result cannot be too long or too short
- If the result is longer, its left digit is 1
- No puzzle can contain more than 10 different letters
- Brute force solution can be optimised
  - exclude obviously wrong hypotheses

Lessons Leant

- Recursion of known max depth can be rewritten as nested loops
- Harder tasks can be made simple by solving subtasks
- Easier tasks can be inefficiently solved by reuse
- Small differences in requirements matter

(collected during the workshop)

SLOC

Lines of code?

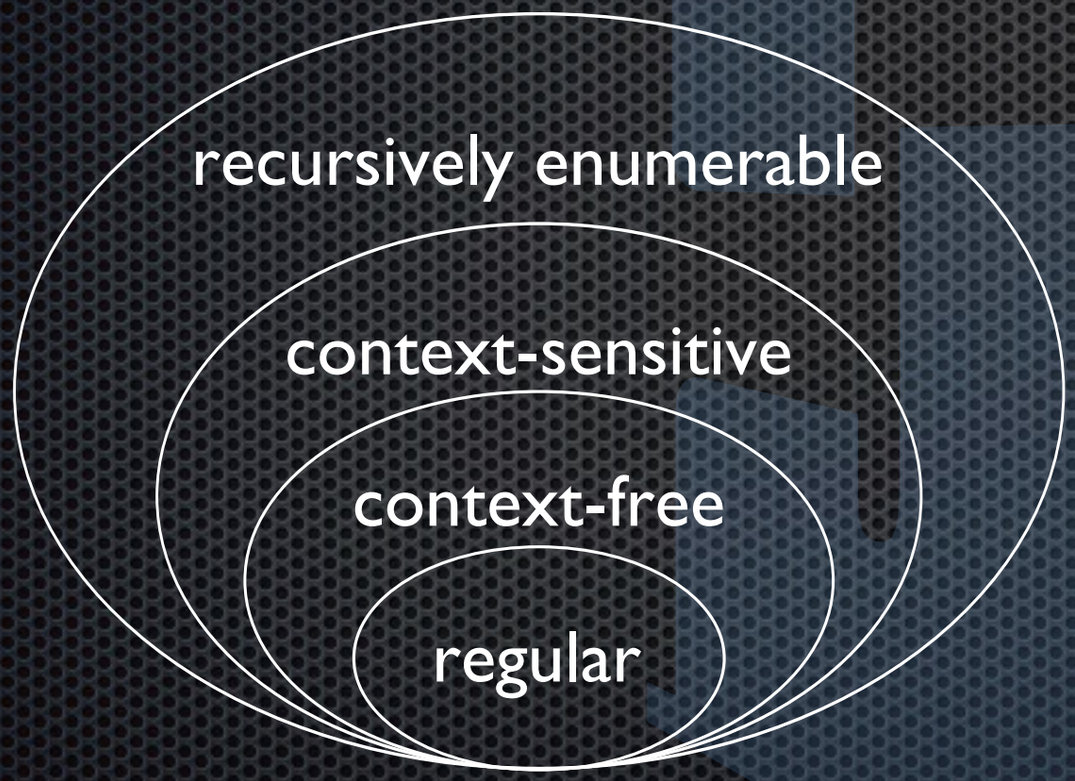
- Count the number of lines of source code in a file
- Disregarding
  - indentation and whitespace
  - empty lines
  - comments

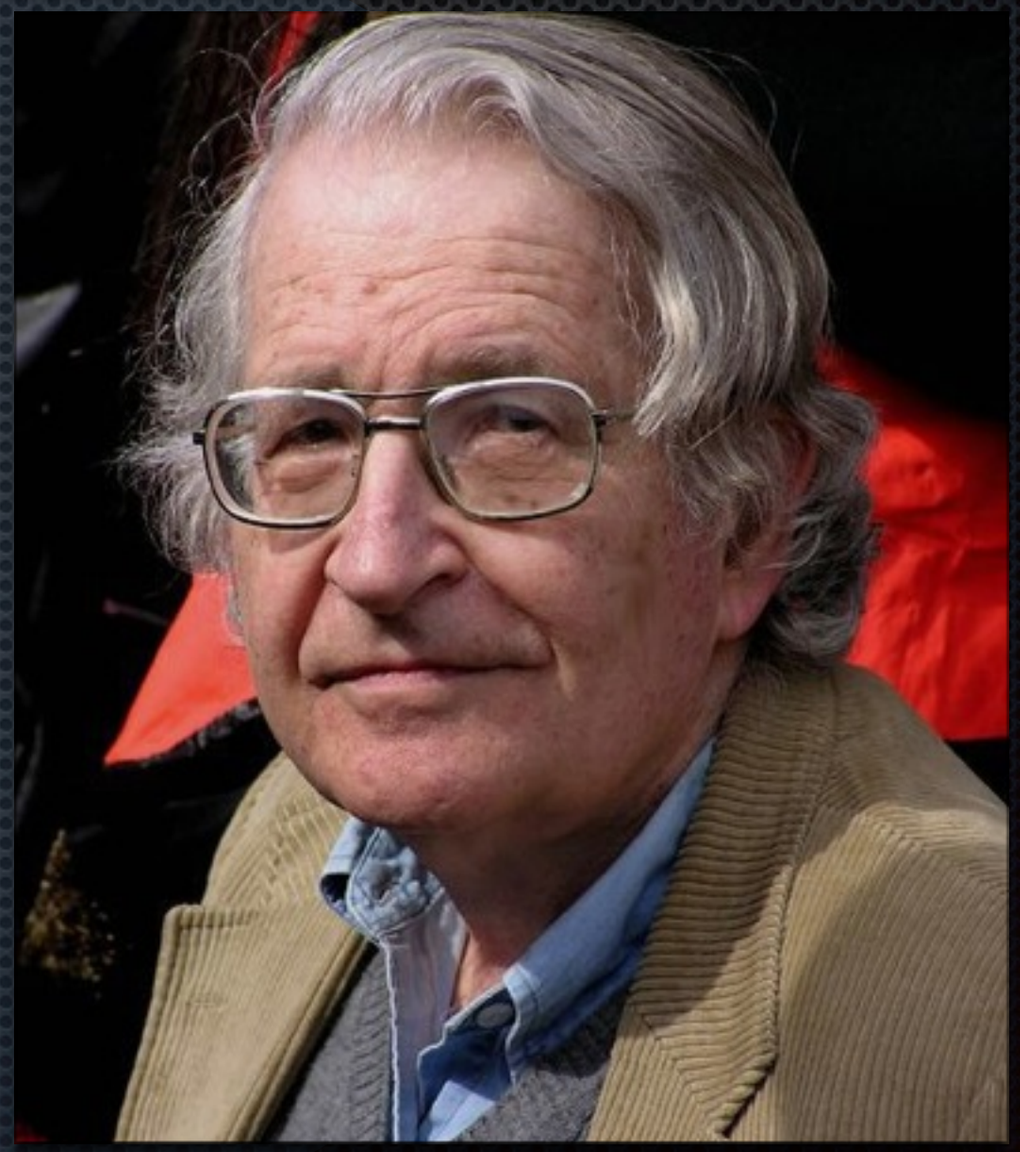
Solution

- Looping over lines
- Trimming/stripping
- Regular expressions for comments
  - trouble with combinations of //, /* */ and “”

Regular languages

below context free in the Chomsky hierarchy!

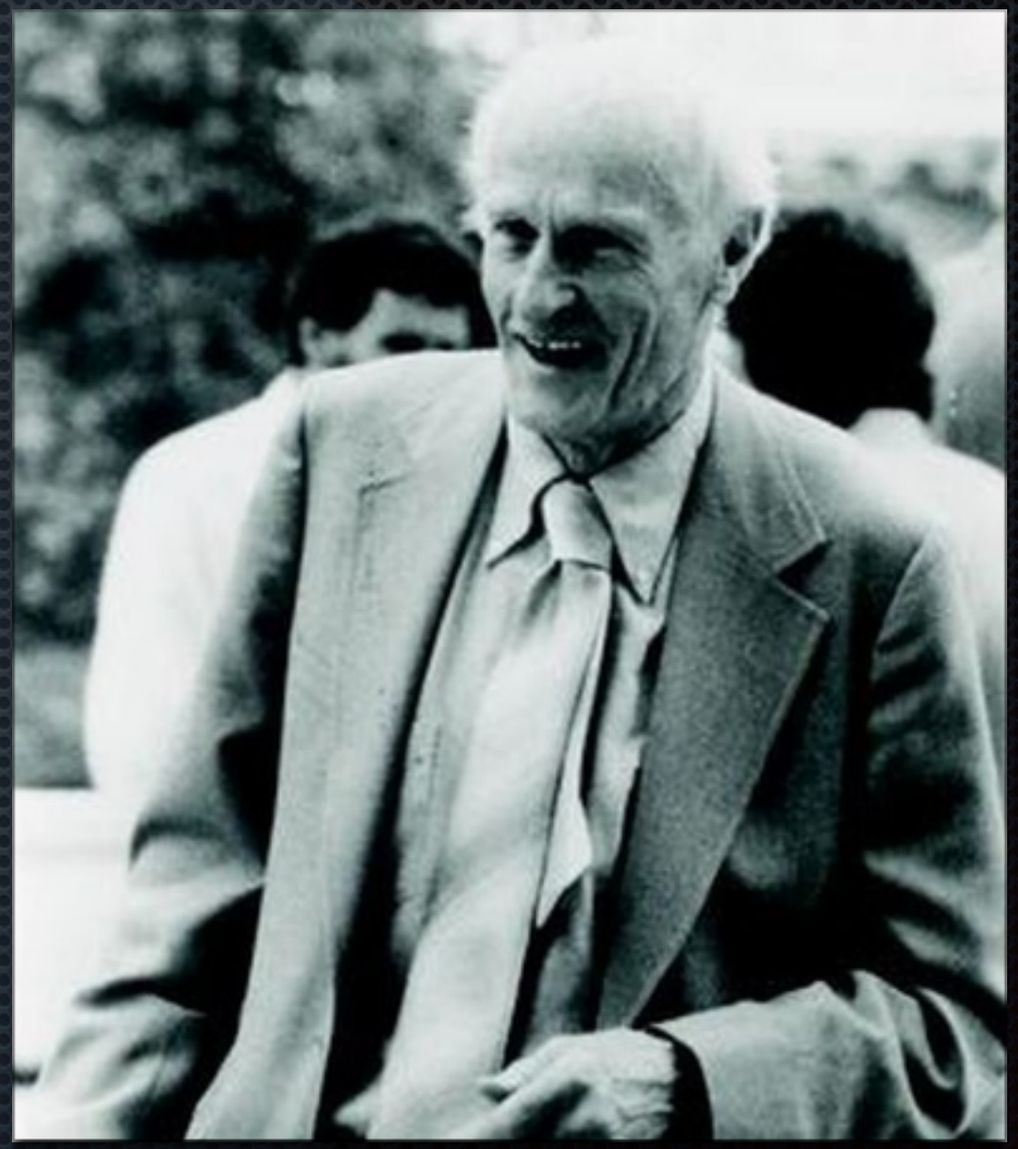




Duncan Rawlinson, Chomsky.jpg, 2004, CC-BY.
J. Finkelstein, Chomsky-hierarchy.jpg, 2010, CC-BY-SA.

REGULAR EXPRESSIONS

- Stephen Kleene invented regexps in 195x
- Ken Thompson added them to ed & grep
- POSIX standard since 1993
- PCRE by Philip Hazel (stable release Dec. 2013)



Konrad Jacobs, S. C. Kleene, 1978, MFO.
Archetypal hackers ken (left) and dmr (right).

REGULAR EXPRESSIONS

- Stephen Kleene invented regexps in 195x
- Ken Thompson added them to ed & grep
- POSIX standard since 1993
- PCRE by Philip Hazel (stable release Dec. 2013)

Konrad Jacobs, S. C. Kleene, 1978, MFO.
Archetypal hackers ken (left) and dmr (right).

Lessons learnt

- Perfect solutions are sometimes provably impossible
- “Close enough” solutions are useful
- Science: definitive proofs
  Engineering: constant incremental advancements
- Ultimate reuse: find a suitable tool
- Metrics should not be abused (careful reporting)

(collected during the workshop)

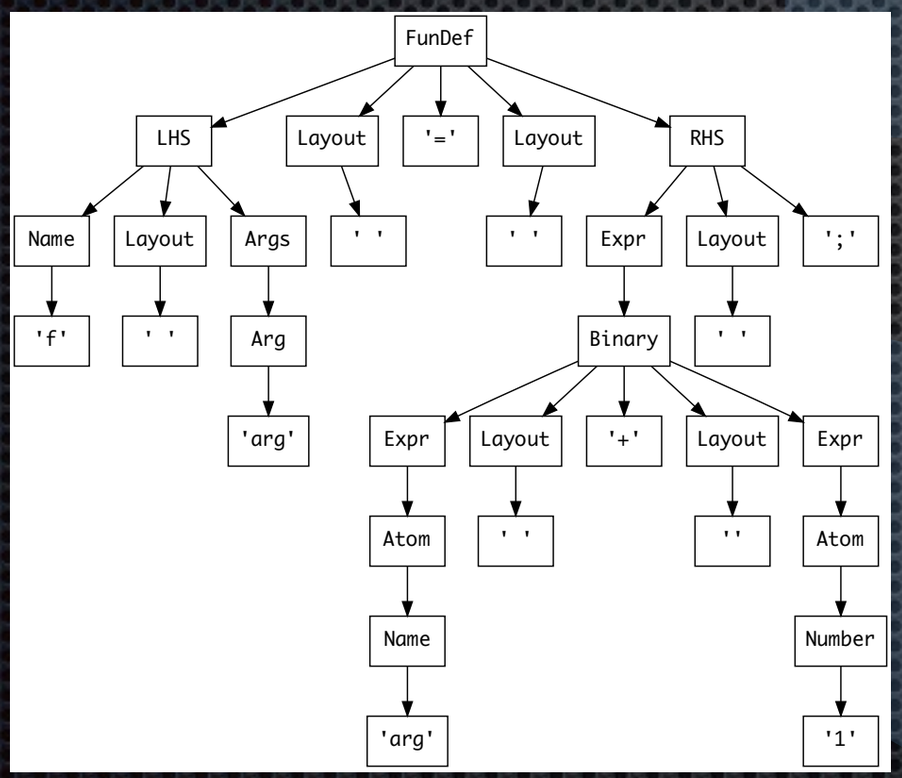
GRAMMARS

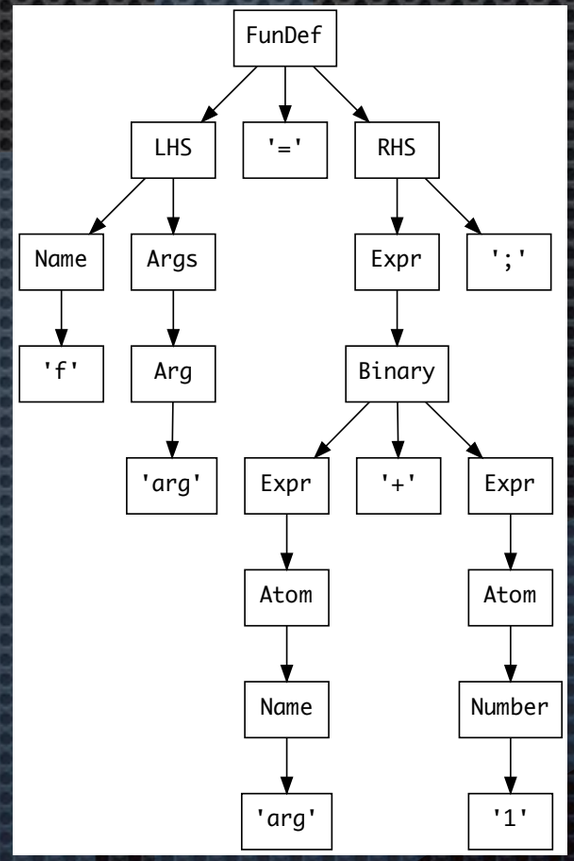
Parsing: Text in, tree out

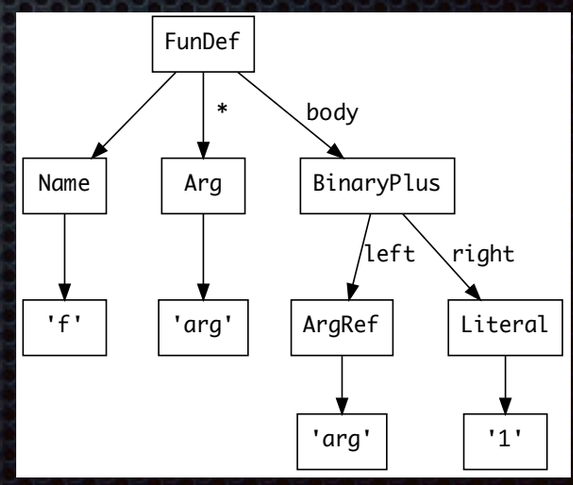
```
f arg = arg +1;
```

f	' '	arg	' '	=	' '	arg	' '	+	1	' '	;

f	arg	=	arg	+	1	;







V. Zaytsev, A. H. Bagge. *A Bidirectional Megamodel of Parsing*, under review at ECMFA 2014, http://grammarware.net/writes/#Parsing2014

Compiler construction

- Dragon Book
  - everything you wanted to know about compilers but were afraid to ask
- not needed in practice

Language Workbenches

- Rascal
  - http://www.rascal-mpl.org/
- Spoofax
  - http://strategoxt.org/Spoofax/
- ANTLR
  - http://www.antlr.org/
- MetaEdit+
  - http://www.metacase.com/
- MetaProgramming System
  - http://www.jetbrains.com/mps/
- Xtext
  - http://www.eclipse.org/Xtext/

or any old-fashioned parser generator of your choice

Rascal - exprlang/src/ExprTest.rsc - Eclipse - /Users/zaytsev/Documents/workspace

Quick Access | Plug-in Development | Rascal | Resource | Debug

Rascal

- exprlang
  - META-INF
  - bin
  - src
    - ExprTest.rsc
    - Lexicals.rsc
  - rascal
  - rascal_eclipse
- grammarlab
- joy
- zoo

***ExprTest.rsc** | Lexicals.rsc | Rascal Figure

```
1  module ExprTest
2
3  import String;
4  extend Lexicals;
5
6  import ParseTree;
7  import vis::ParseTree;
8
9  start syntax Expr
10     = "..."
11     ;
12
13 lexical XOp = [+] > [\-] > [*] > [/];
14
15 Expr getTree(str s) = parse(#start[Expr],s).top;
16 bool checkTree(str s) = /amb(_) !:= getTree(s);
17
18 test bool p1() = checkTree("2");
19 test bool p2() = checkTree(" 2 ");
20 test bool p3() = checkTree("2 // lkhjvb");
```

Outli | Ambi | Debu

- Variables (0)
- Types (0)
- Aliases (0)
- Annotations (0)
- Tags (0)
- Imports (4)
- Syntax (2)
- Functions (4)
  - getTree (1)
  - checkTree (1)
  - showme (1)
  - eval (8)
- Tests (9)

Console | Output | Progress | Problems — Store history | Terminate | Interrupt | Trace

Rascal [DEBUG, exprlang]

```
rascal>:test
ok

rascal>showme("2*2+2*2")
ok

rascal>
```

Variables

233M of 1218M

Grammar as parsing spec

- Terminals: expected text
- Nonterminals: classes or categories
- Can be combined in a sequence or with choice
- (other workshop-specific fluff)

```
A ::= B C;

B ::= “hello”;

C ::= “world”;
```

Expression Language

- Need to process an expression language
- Don't write a parser
- Write a spec
- Concrete syntax def
- Let's start with numbers

```
2 + 2

2 - (2 + 2)*2/2-2

2 - 2 - 2 - 2

2
```

EXPRESSIONS: NUMBERS

- Grammar is a spec, tests are for the parser
- Numbers do not start with zeros
- There is whitespace around them

EXPRESSIONS: BINARY OPS

- We have numbers
- Next step:
- operators
- let’s focus on the binary ones
- +, -, /, *

Expressions: Priorities

expression:
    conditional-or-expression
conditional-or-expression:
    conditional-and-expression
    conditional-or-expression "||" conditional-and-expression
conditional-and-expression:
    inclusive-or-expression
    conditional-and-expression "&&" inclusive-or-expression
inclusive-or-expression:
    exclusive-or-expression
    inclusive-or-expression "|" exclusive-or-expression
exclusive-or-expression:
    and-expression
    exclusive-or-expression "^" and-expression
and-expression:
    equality-expression
    and-expression "&" equality-expression

Microsoft, C# Language Specification 4.0, 2010, http://slps.github.io/zoo/cs/csharp-msft-ls-4.0.html
Pelf Nyok, Stacked-up river terrapins, 2006, PD.

EXPRESSIONS: PRIORITIES

```
equality_expression:
        shift_expression
        equality_expression EQ_OP relational_expression
        equality_expression NE_OP relational_expression
shift_expression:
        additive_expression
        shift_expression LEFT_OP additive_expression
        shift_expression RIGHT_OP additive_expression
additive_expression:
        multiplicative_expression
        additive_expression '+' multiplicative_expression
        additive_expression '-' multiplicative_expression
multiplicative_expression:
        cast_expression
        multiplicative_expression '*' cast_expression
        multiplicative_expression '/' cast_expression
        multiplicative_expression '%' cast_expression
```

Jeff Lee, jutta@pobox.com et al, ANSI C Yacc grammar, http://www.quut.com/c/ANSI-C-grammar-y.html
Unknown source (PD?), http://imgur.com/09C5G

EXPRESSIONS: PRIORITIES

```
void LogicalORExpression() : {}
{       LogicalANDExpression() [ "||" LogicalORExpression() ]           }

void LogicalANDExpression() : {}
{       InclusiveORExpression() [ "&&" LogicalANDExpression() ]         }

void InclusiveORExpression() : {}
{       ExclusiveORExpression() [ "|" InclusiveORExpression() ]   }

void ExclusiveORExpression() : {}
{       ANDExpression() [ "^" ExclusiveORExpression() ]     }

void ANDExpression() : {}
{       EqualityExpression() [ "&" ANDExpression() ]   }

void EqualityExpression() : {}
{       RelationalExpression() [ ( "==" | "!=" ) EqualityExpression() ]     }

void RelationalExpression() : {}
{       ShiftExpression() [ ( "<" | ">" | "<=" | ">=" ) RelationalExpression() ]        }
```

Doug South, Tom Copeland, C grammar defintion for use with JavaCC, 1997, https://java.net/downloads/javacc/contrib/grammars/C.jj

Murata Seimin, Five Turtles, 1761 and 1837, http://art.thewalters.org/detail/37228

Side story: Parsing techniques

- Top-down recursive descent parsers: LL(k) etc
  - right recursion is ok, left recursion is deadly
- Bottom-up parsers: CYK, LR(k), LALR, etc
  - right recursion is suboptimal, left recursion is ok
- Top-down parsing, bottom-up lookahead: Earley etc
  - right recursion is ok, left recursion is faster

Expressions: Priorities

```
Expression:
        Expression1 (AssignmentOperator Expression1)?
Expression1:
        Expression2 Expression1Rest?
Expression1Rest:
        ("?" Expression ":" Expression1)?
Expression2:
        Expression3 Expression2Rest?
Expression2Rest:
        (Infixop Expression3)* | "instanceof" Type
Expression3:
        PrefixOp Expression3
Expression3:
        "(" (Expression | Type) ")" Expression3
Expression3:
        Primary Selector* PostfixOp*
```

J. Gosling, B. Joy, G. Steele, G. Bracha, Java Language Specification, 1998, §18: Syntax, pages 449–456,
http://slps.github.io/zoo/java/java-2-jls-impl.html

Expressions: Priorities

```
Expression:
        Expression1 (AssignmentOperator Expression1)?
Expression1:
        Expression2 Expression1Rest?
Expression1Rest:
        ("?" Expression ":" Expression1)?
Expression2:
        Expression3 Expression2Rest?
Expression2Rest:
        (Infixop Expression3)* | "instanceof" Type
Expression3:
        PrefixOp Expression3
Expression3:
        "(" (Expression | Type) ")" Expression3
Expression3:
        Primary Selector* PostfixOp*
```

J. Gosling, B. Joy, G. Steele, G. Bracha, Java Language Specification, 1998, §18: Syntax, pages 449–456,
http://slps.github.io/zoo/java/java-2-jls-impl.html

EXPRESSIONS: PRIORITIES

```
exp:
    'nil' | 'false' | 'true' | number | string | '...' |
    functiondef | prefixexp | tableconstructor |
    exp binop exp | unop exp;
binop
    : '+' | '-' | '*' | '/' | '^' | '%' | '..'
    | '<' | '<=' | '>' | '>=' | '==' | '~='
    | 'and' | 'or';
unop
    : '-' | 'not' | '#';
```

Kazunori Sakamoto, Lua 5.2 grammar in ANTLR4,
https://github.com/antlr/grammars-v4/blob/master/lua/Lua.g4

Expressions: Interpreter

- Recogniser tells us which program is correct
- Parser builds a parse tree
- Some want a semi-auto-derived abstract syntax tree
- We need smth to walk that tree and execute it
  - e.g., perform calculation
  - e.g., check for correctness

EXPRESSIONS: VARIABLES?

- how to introduce variables into a software language?
  - many different ways
    - each one has implications
  - very hard to do w/o background

$(\lambda x . 2 + x)\ 2$

$2 + x$ *where* $x=2$

{x=2;return 2 +x}

Lessons Leant

- Full TDD: start with failing tests, adapt, repeat
- [1-9][0-9]*|[0] is uglier than [0-9]+
- Comments should not show up explicitly in the grammar
- Separate tree validation function is useful
- Parsing can be ambiguous
- Several ways to specify/encode priorities

(collected during the workshop)

Summary

- Software engineering vs computer science
- Alphametic cryptarithms
- Calculating lines of code in a software system
- Software language engineering
- ~~Mining software repositories~~ (no time)

http://grammarware.net/talks/#DayOfTheMaster2014

Coding Dojo Feedback

- What went well?
- What did not?
- What did we learn?

KendoSnowman, Jon Takagi dojo 1970, CC-BY-SA, 2014

Feedback

- Hacking is fun. Moar hacking!
- Brief SLE intro appreciated
- Background info feels CS, not SE

(collected during the workshop)

SOFTWARE ENGINEERING

- One year Master of Science programme at UvA
- Skills covered:
  - programming
  - critical thinking
  - solving unsolvable
- Skills uncovered:
  - reading scientific papers
  - collaborating in bigger projects
- http://www.software-engineering-amsterdam.nl

vadim@grammarware.net

- Designosaur by Archy Studio
- Heuristica by Andrej Panov
- MATT SMITH DOCTOR WHO BY THEJAMJAR
- GNU Typewriter by Lukasz Komsta
- DALEK BY K-TYPE

- Doctor Who screenshots by BBC
-  logo by Vadim Zaytsev
-  logo by Tobias Baanders
- Everything used strictly for educational purposes
- Yours, @grammarware