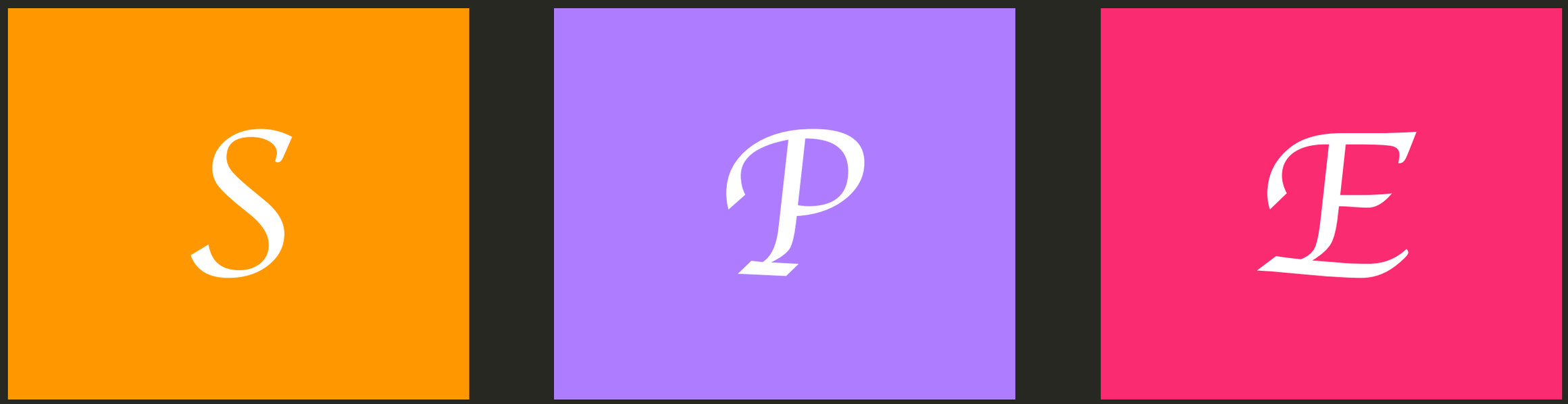
Software Evolution: Conclusion, Discussion, Future Work

Dr. Vadim Zaytsev aka @grammarware
UvA, MSc SE, 7 December 2015

Roadmap

W44	Introduction	V.Zaytsev
W45	Metaprogramming	J.Vinju
W46	Reverse Engineering	V.Zaytsev
W47	Software Analytics	M.Bruntink
W48	Clone Management	M.Bruntink
W49	Source Code Manipulation	V.Zaytsev
W50	Conclusion	V.Zaytsev
W51	Legacy and Renovation	D.Blasband

Software Types



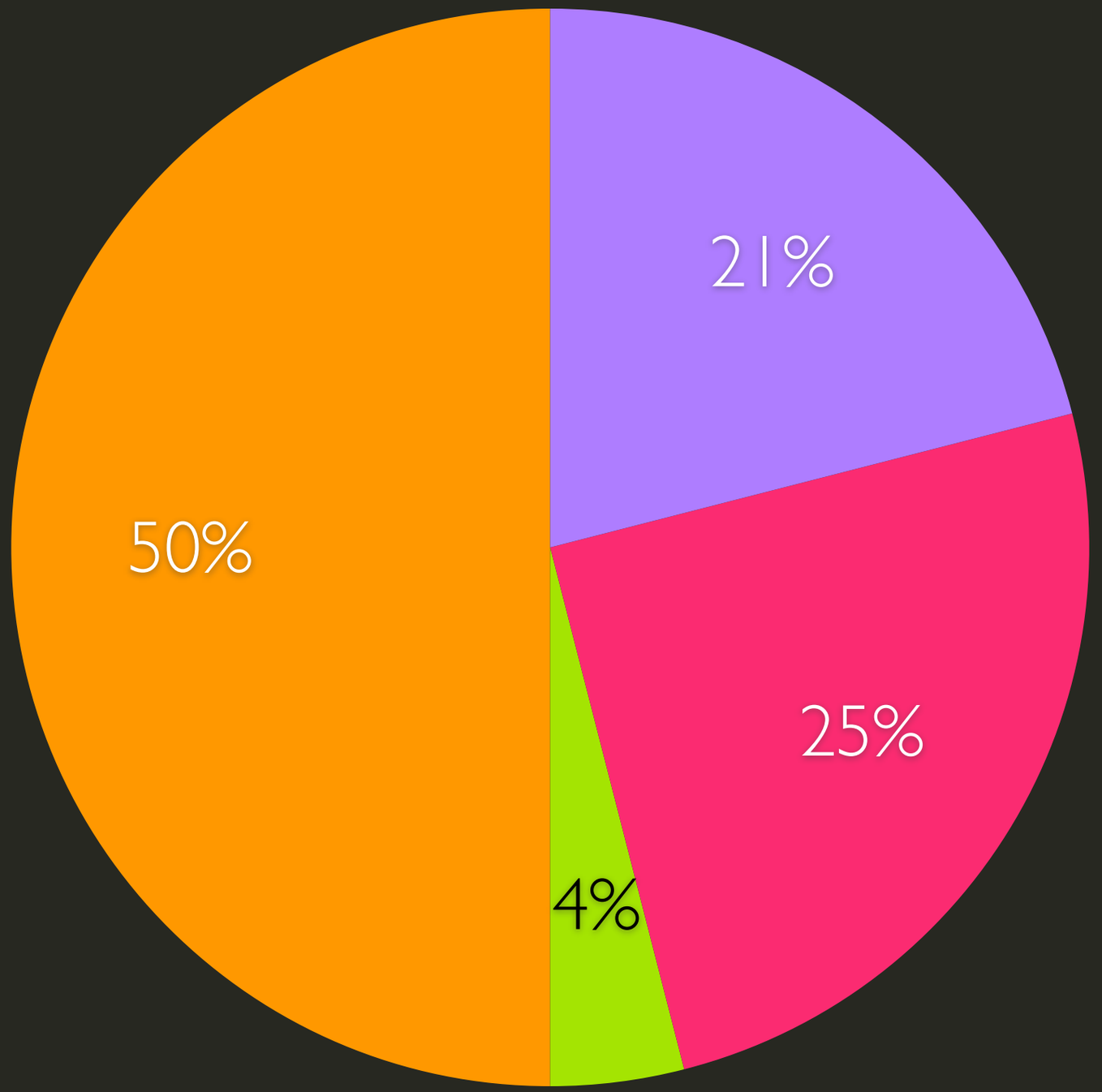
M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Laws of Software Evolution

http://www.computer.org/web/awards/mills-meir-lehman

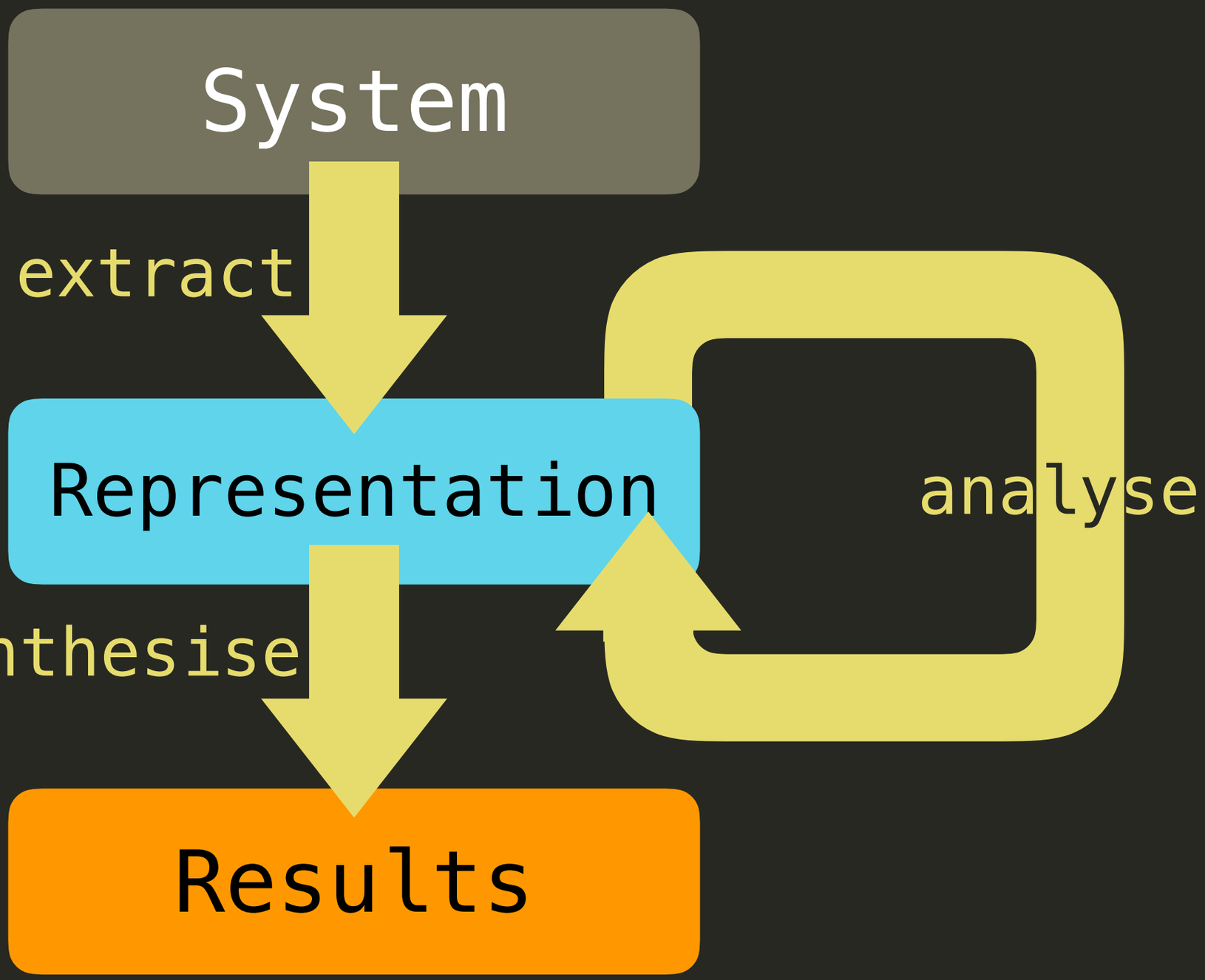
M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Types of maintenance



B.P.Lientz, E.B.Swanson, *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, 1980.

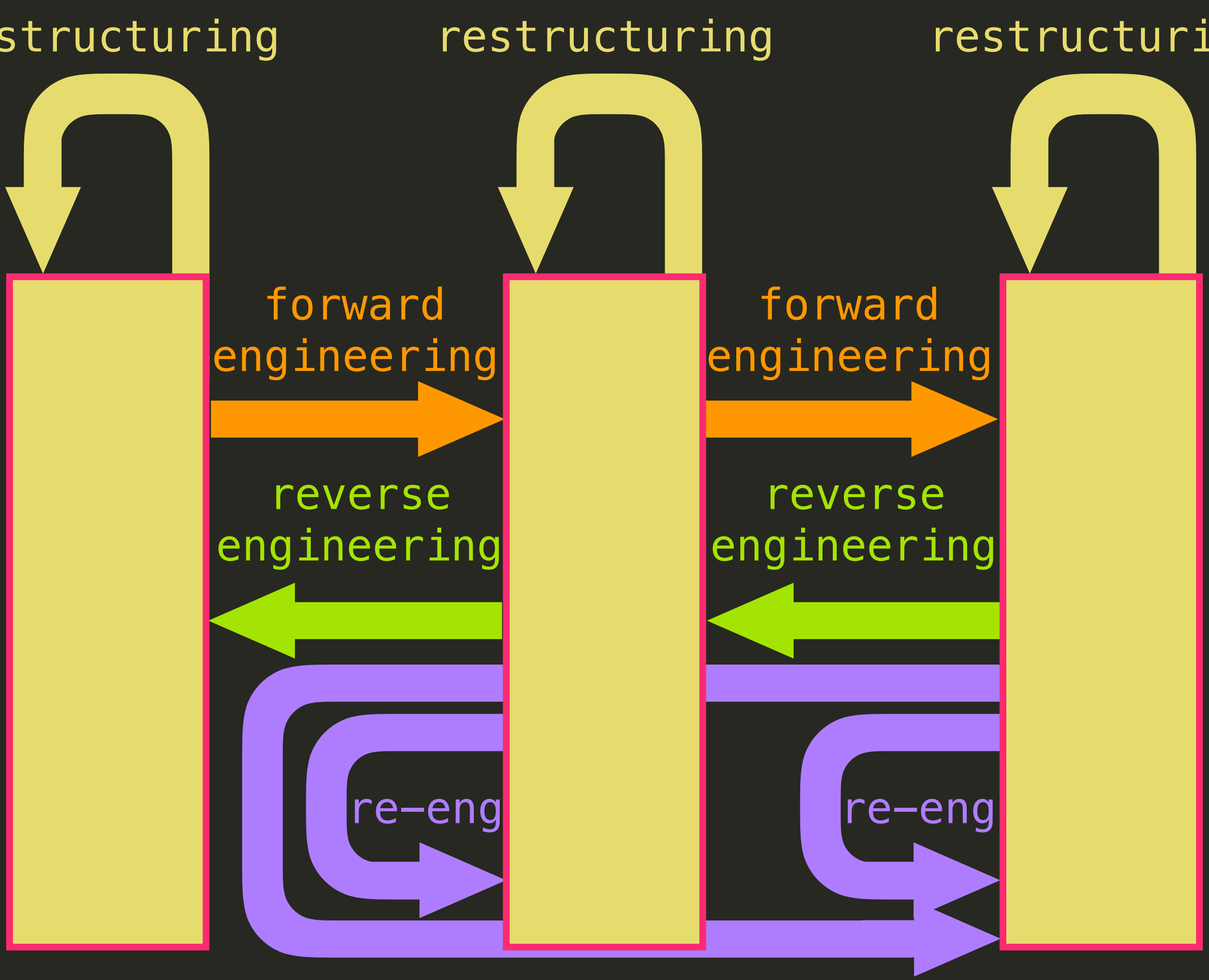
EASY



P. Klint, T. v/d Storm, J. Vinju, EASY Meta-programming with Rascal, GTTSE 2009.

Rascal

http://rascal-mpl.org

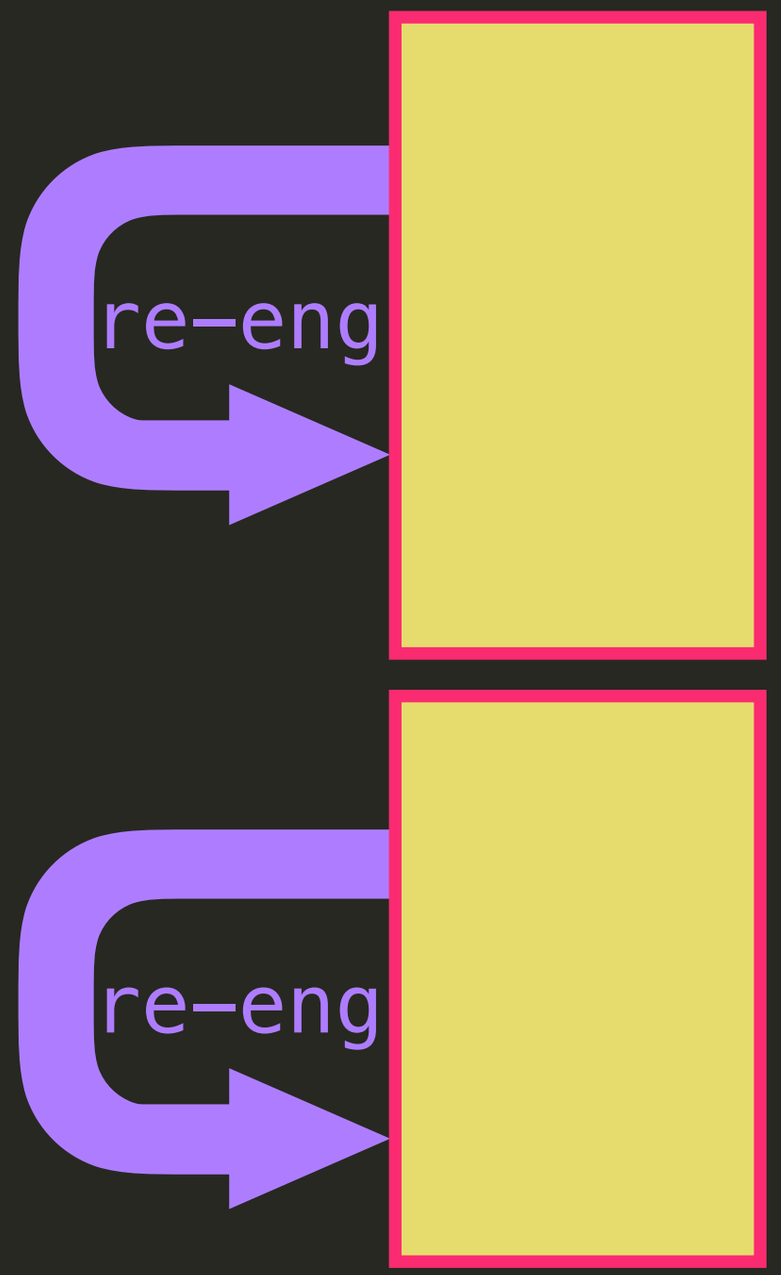


E.Chikofsky, J.H.Cross II, Reverse Engineering and Design Discovery: A Taxonomy. IEEE Software 7:1, 1990.

Reverse Engineering

* Parsing, slicing, exploration…

* PR, KR, IR, MUD, clustering…



H.A.Müller, J.H.Jahnke, D.B.Smith, M.-A.Storey, S.R.Tilley, K.Wong, Reverse Engineering: A Roadmap, ICSE 2000. http://bibtex.github.io/ICSE-2000-Future-MullerJSSTW.html

Parsing is…

3.1 Two classes of parsing methods

A parsing method constructs the syntax tree for a given sequence of tokens. Constructing the syntax tree means that a tree of nodes must be created and that these nodes must be labeled with grammar symbols, in such a way that:

- leaf nodes are labeled with terminals and inner nodes are labeled with non-terminals;
- the top node is labeled with the start symbol of the grammar;
- the children of an inner node labeled *N* correspond to the members of an alternative of *N*, in the same order as they occur in that alternative;
- the terminals labeling the leaf nodes correspond to the sequence of tokens, in the same order as they occur in the input.

Left-to-right parsing starts with the first few tokens of the input and a syntax tree, which initially consists of the top node only. The top node is labeled with the start symbol.

The parsing methods can be distinguished by the order in which they construct the nodes in the syntax tree: the top-down method constructs them in pre-order, the bottom-up methods in post-order. A short introduction to the terms "pre-order" and "post-order" can be found below. The top-down method starts at the top and con-

deconstructing the problem into primitives. Section 7 presents a framework and algorithm for the evolution of modelling artefacts when languages evolve. Section 8 concludes the paper and describes future work.

2. Modelling languages

To allow for a precise discussion of language evolution, we briefly introduce fundamental modelling language concepts. This introduction which we elaborated in [10] is based on foundations laid by Harel and Rumpe [13] and Kühne [21]. The two main aspects of a model are its *syntax* (how it is represented) and its *semantics* (what it means).

Firstly, the syntax comprises *concrete syntax* and *abstract syntax*. The concrete syntax describes how the model is represented (*e.g.*, in 2D vector graphics or in textual form), which can be used for model input as well as for model visualisation. The abstract syntax contains the "essence" of the model (*e.g.*, as a typed Abstract Syntax Graph (ASG)—when models are represented as graphs).

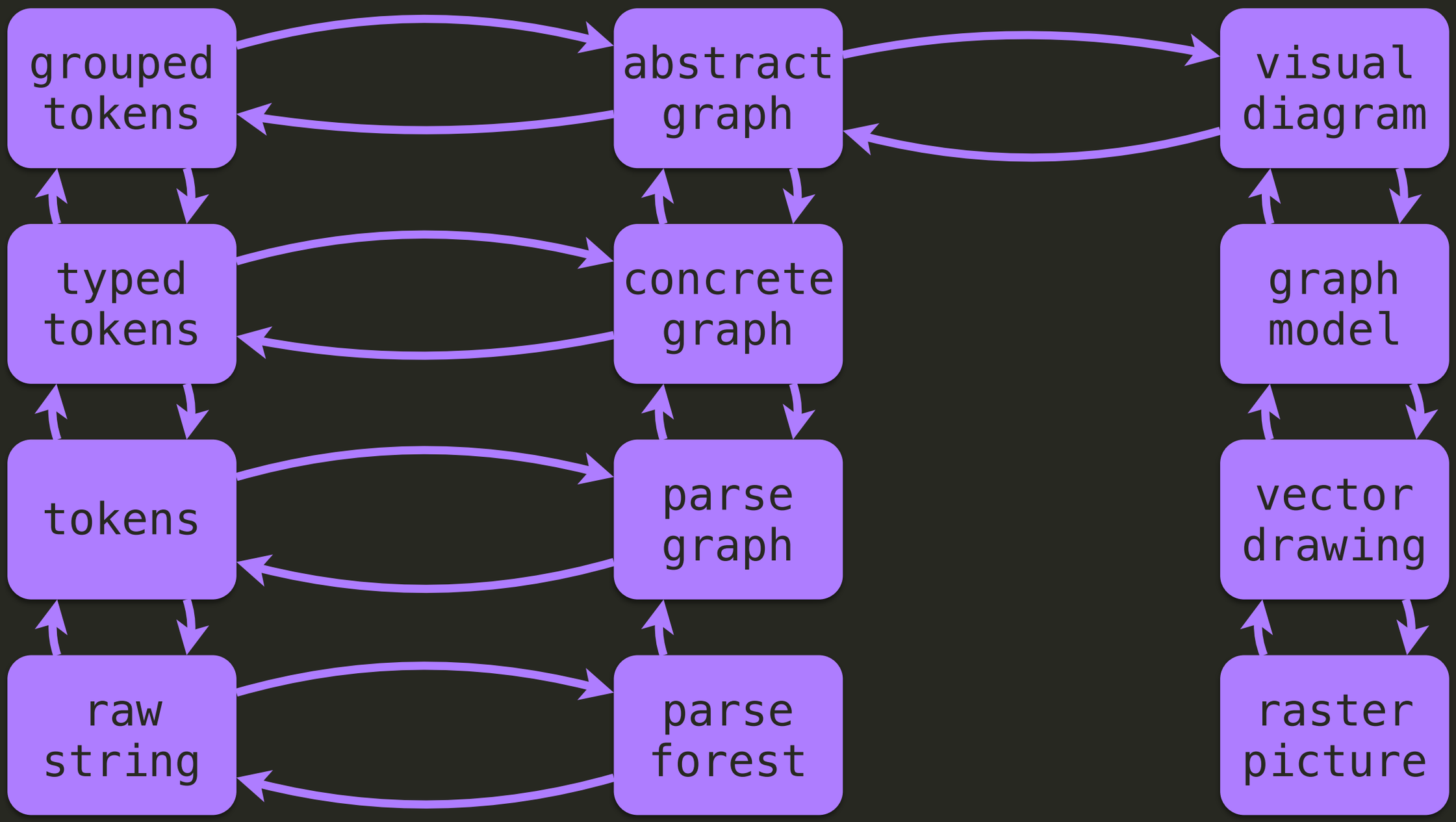
A single abstract syntax may be represented by multiple concrete syntaxes. There exists a mapping between a concrete syntax and its abstract syntax, called the *parsing function*. There is also a mapping in the opposite direction, called the *rendering function*. These are the *concrete mapping functions*. Mappings are usually implemented, or can at least be represented, as model transformations. The abstract syntax and concrete syntax of a model are related by a surjective homomorphic function that translates a concrete syntax graph into an abstract syntax graph.

Secondly, the semantics of a model are defined by a complete, total and unique *semantic mapping function* which maps every abstract syntax model onto a single element in a *semantic domain*, such as Ordinary Differential Equations, Petri nets [39], or a set of behaviour traces. These are domains with well-known and precise semantics. For convenience, semantic mapping is usually performed on abstract syntax, rather than on concrete syntax directly. More explicitly, the abstract syntax can be used as a basis for semantic anchoring [4].

A meta-model is a finite model that explicitly describes the abstract syntax and static semantics, which are statically checkable, of a language. Dynamic semantics are not covered by the meta-model. The abstract syntax of a model can be represented as a graph, where the nodes are elements of the language and the edges are relations between these elements, and also elements of the language. Instance models of the language are said to *conform to* the meta-model of the language. In [21], Kühne refers to this relation as *linguistic instance of*. The description of the abstract syntax is typically specified in a modelling language such as UML Class Diagrams [34]. Static semantics can be described in a constraint language such as the Object Constraint Language (OCL) [36]). Often, but not necessarily, the concrete syntax mapping is directly attached to a meta-model, where every element of the concrete syntax can be explicitly traced back to its corresponding element of the abstract syntax.

Fig. 1 shows the different kinds of relations involving a model *m*. Relations are visualised by arrows, "conforms to"-

Program Models



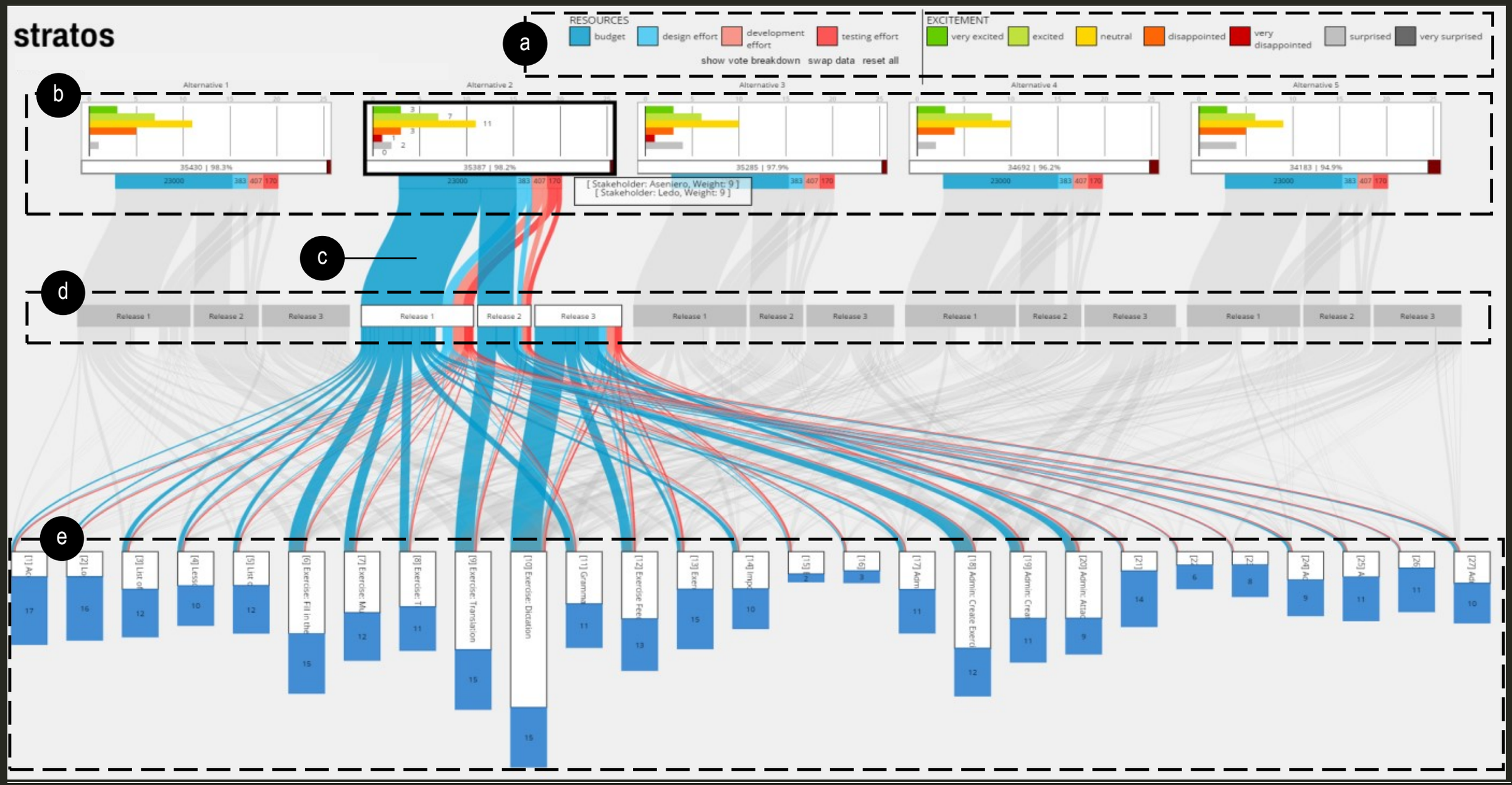
V.Zaytsev, A.H.Bagge, Parsing in a Broad Sense, MoDELS 2014.
http://bibtex.github.io/MoDELS-2014-ZaytsevB.html

Program Slicing

```
read(text);
read(n);
lines = 1;
chars = 1;
subtext = "";
c = getChar(text);
while (c != ‘\eof’)
      if (c == ‘\n’)
      then lines = lines + 1;
           chars = chars + 1;
      else chars = chars + 1;
           if (n != 0)
           then subtext = subtext ++ c;
                n = n – 1;
      c = getChar(text);
write(lines);
write(chars);
write(subtext);
```

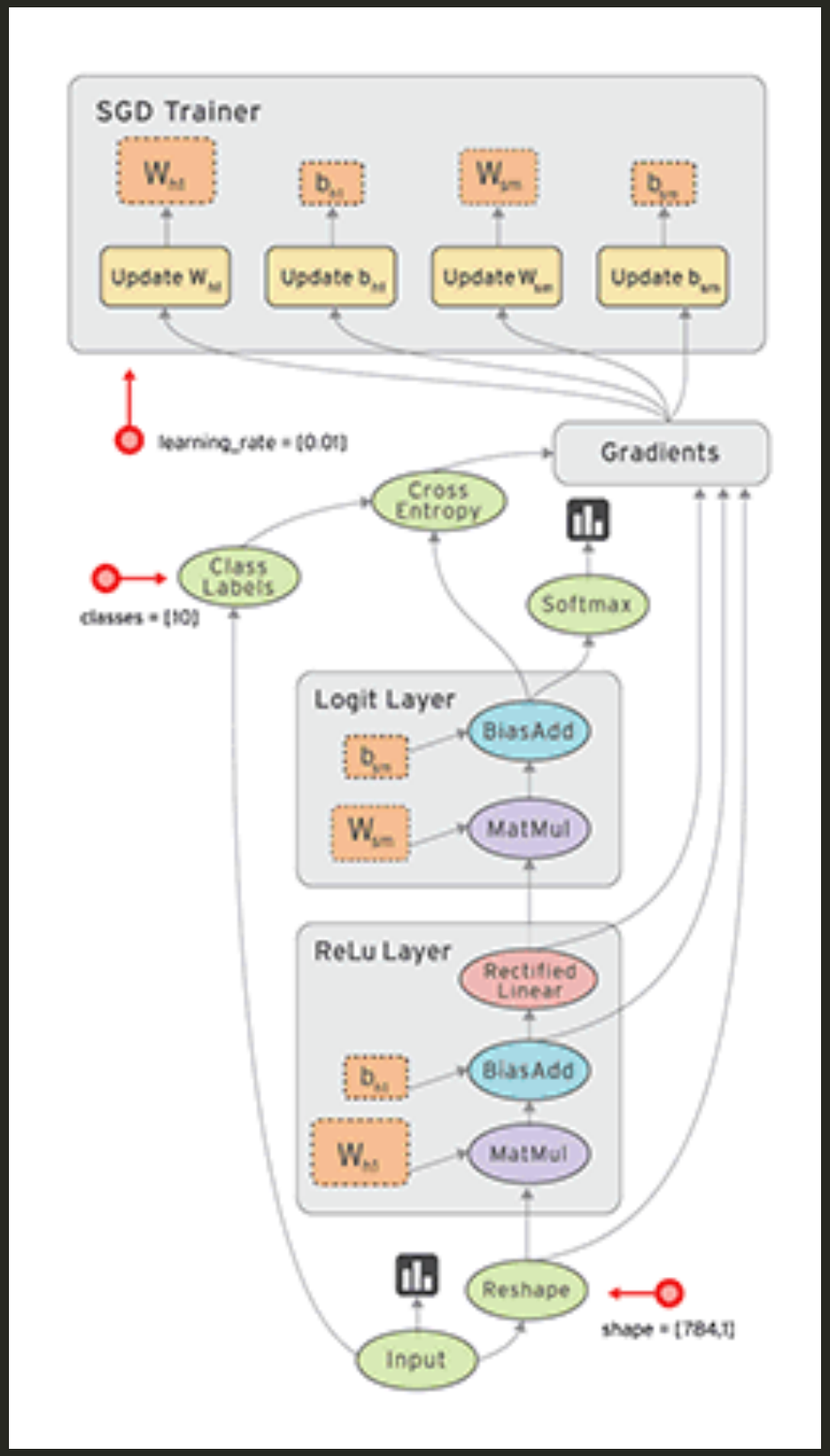
J. Silva, A Vocabulary of Program Slicing-Based Techniques, CSUR, 2012.

Exploration



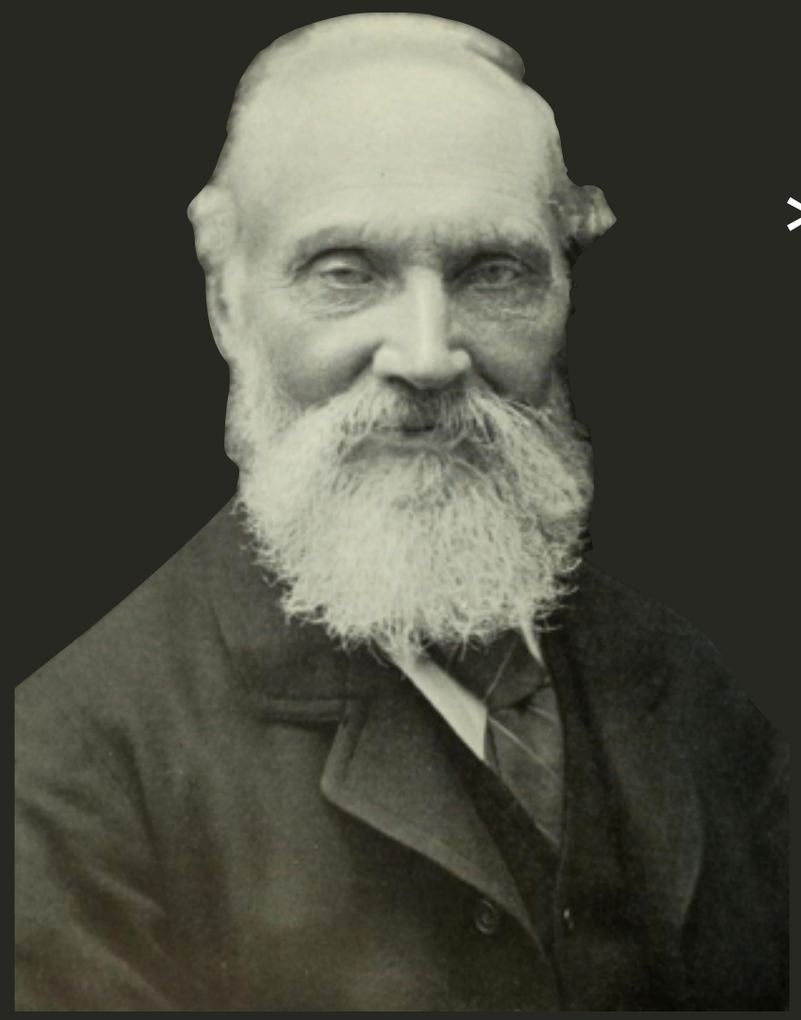
B.A.Aseniero, T.Wun, D.Ledo, G.Ruhe, A.Tang, S.Carpendale, STRATOS: Using Visualization to Support Decisions in Strategic Software Release Planning, CHI 2015. http://bibtex.github.io/CHI-2015-AsenieroWLRTC.html

Information Retrieval



To Measure is to Know

* Scales
  * nominal
  * ordinal
  * interval
  * ratio
  * absolute



* Examples
  * team size
  * code size
  * run time
  * SIG stars
  * colours

Goal–Question–Metric

* Goal – conceptual
  * purpose / issue / object / pov
* Question – operational
  * can be multiple per goal
* Metric – quantifiable
  * can be multiple per question

V. Basili, G. Caldiera, H.D. Rombach, Goal Question Metric (GQM) Approach, Encyclopaedia of Software Engineering, 2002.

Clone Terminology

* Clone
* Clone pair
* Clone class

* Type I
* Type II
* Type III
* Type IV

Clone Types

* Type I: exact
  * copy-paste + indent/comment
* Type II: parametrised
  * copy-paste + convention/typing
* Type III: near-miss
  * copy-paste + hacking/maintenance
* Type IV: semantic
  * copy-paste + refactoring

Rainer Koschke, The Vision of Software Clone Management, CSMR-WCRE 2014 keynote.

Other Clone Types

* Structural clones
  * implementation patterns & notations
* Artefact clones
  * entire files, classes, functions…
* Model clones
  * not-quite-code
* Contextual clones
  * duplicate due to usage patterns

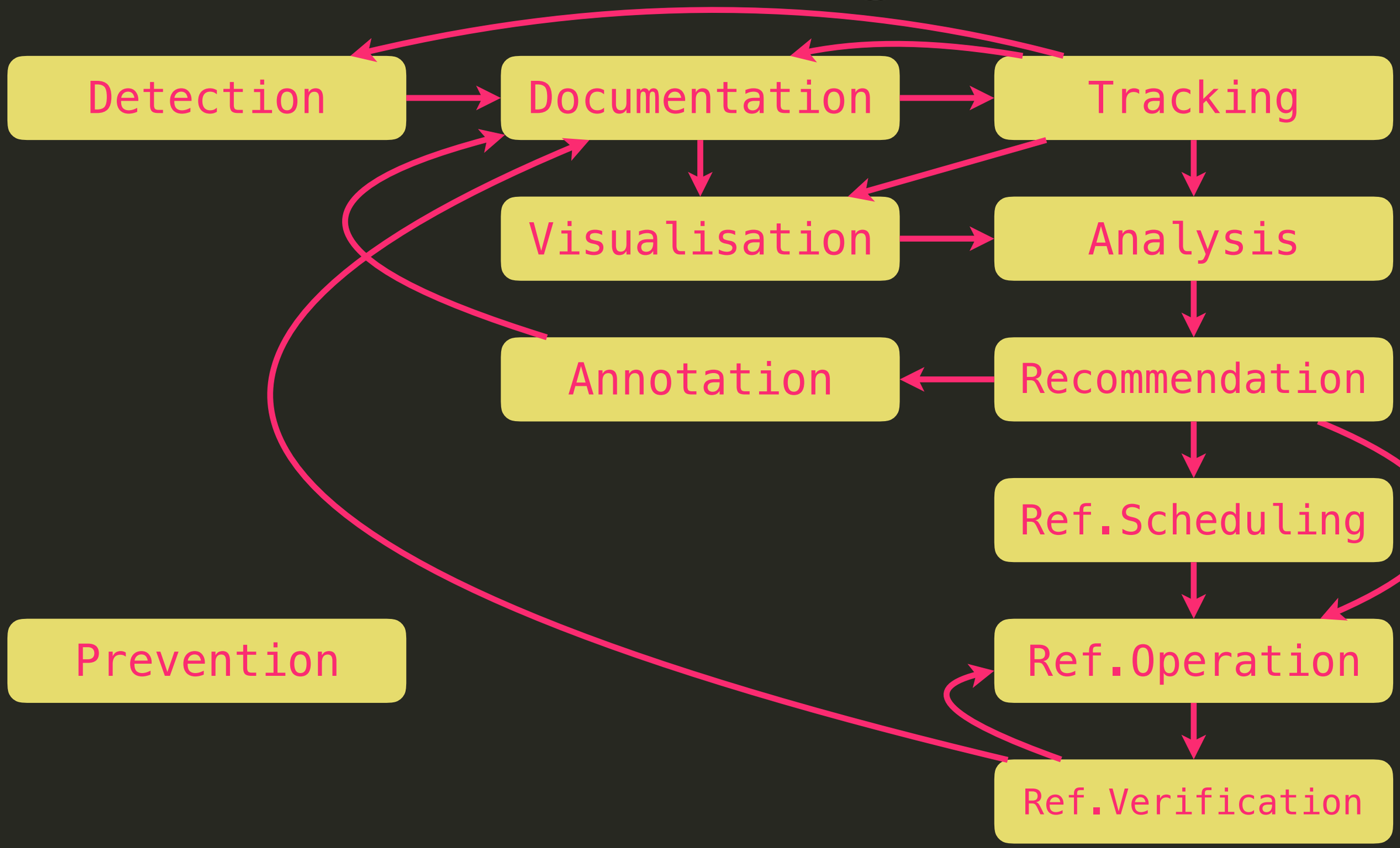
Hamid, Zaytsev, Detecting Refactorable Clones by Slicing Program Dependence Graphs, SATToSE 2014.

How Much Code is Cloned?

* 12.7% [Baxter et al. ICSM'98]
* 10–15% [Kapser & Godfrey JSME'06]
* 7–24% [Roy & Cordy JSME'10/WCRE'08]
* 50% [Ducasse et al. JSME'06]
* 7–23% [Baker WCRE'95]

Rainer Koschke, The Vision of Software Clone Management, CSMR-WCRE 2014 keynote.

Clone Management



Rainer Koschke, The Vision of Software Clone Management, CSMR-WCRE 2014 keynote.

IDE-based Approach

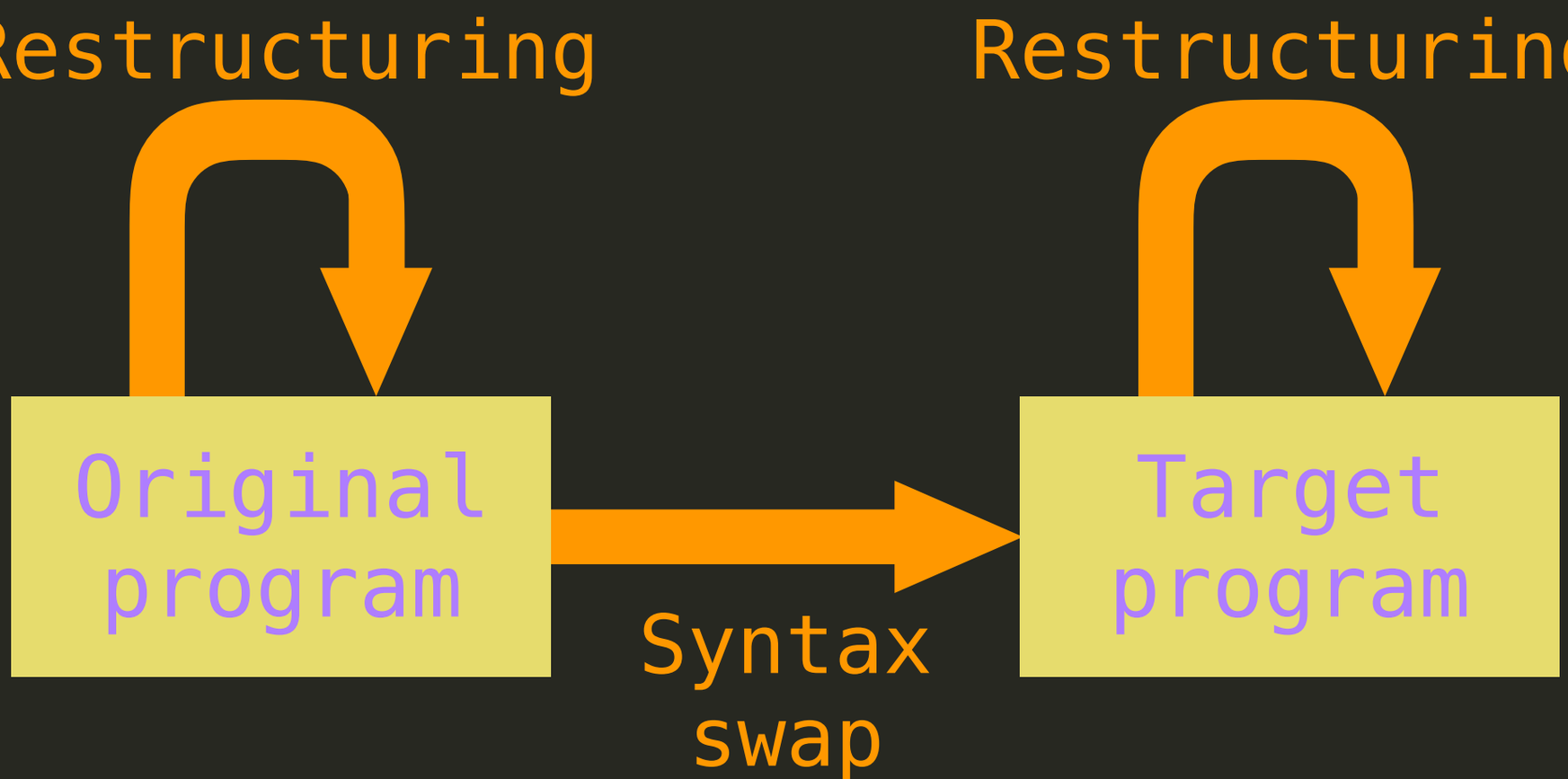
	SimScan	SDD	Simian	CloneDigger	CnP	CSeR	CloneBoard	SHINOBI	CloneDR	Zibran and Roy	JClone	CloneTracker	CeDAR
Integrated Detection Engine	■	■	■	■	■	■	■	■	■	■	■	*	*
Copy-paste Detection	■	■	■	■	■	■	■	■	■	■	■	■	■
Type-1 Clone Detection	■	■	■	■	■	■	■	■	■	■	■	■	■
Type-2 Clone Detection	■	■	■	■	■	■	■	■	■	■	■	■	■
Type-3 Clone Detection	■	■	■	■	■	■	■	■	■	■	■	■	■
On-the-fly/Focused Detection	■	■	■	■	■	■	■	■	■	■	■	■	■
Clone Visualization	■	■	■	■	■	■	■	■	■	■	■	■	■
Code Change History View	■	■	■	■	■	■	■	■	■	■	■	■	■
Clone Genealogy View	■	■	■	■	■	■	■	■	■	■	■	■	■
Clone Tracking	■	■	■	■	■	■	■	■	■	■	■	■	■
Clone Refactoring	■	■	■	■	■	■	■	■	■	■	■	■	■

Rainer Koschke, The Vision of Software Clone Management, CSMR-WCRE 2014 keynote.

SCAM

* Partial evaluation
* Generative programming
* Staging and morphing
* Optimisation
* Folding and unfolding
* Superoptimisation

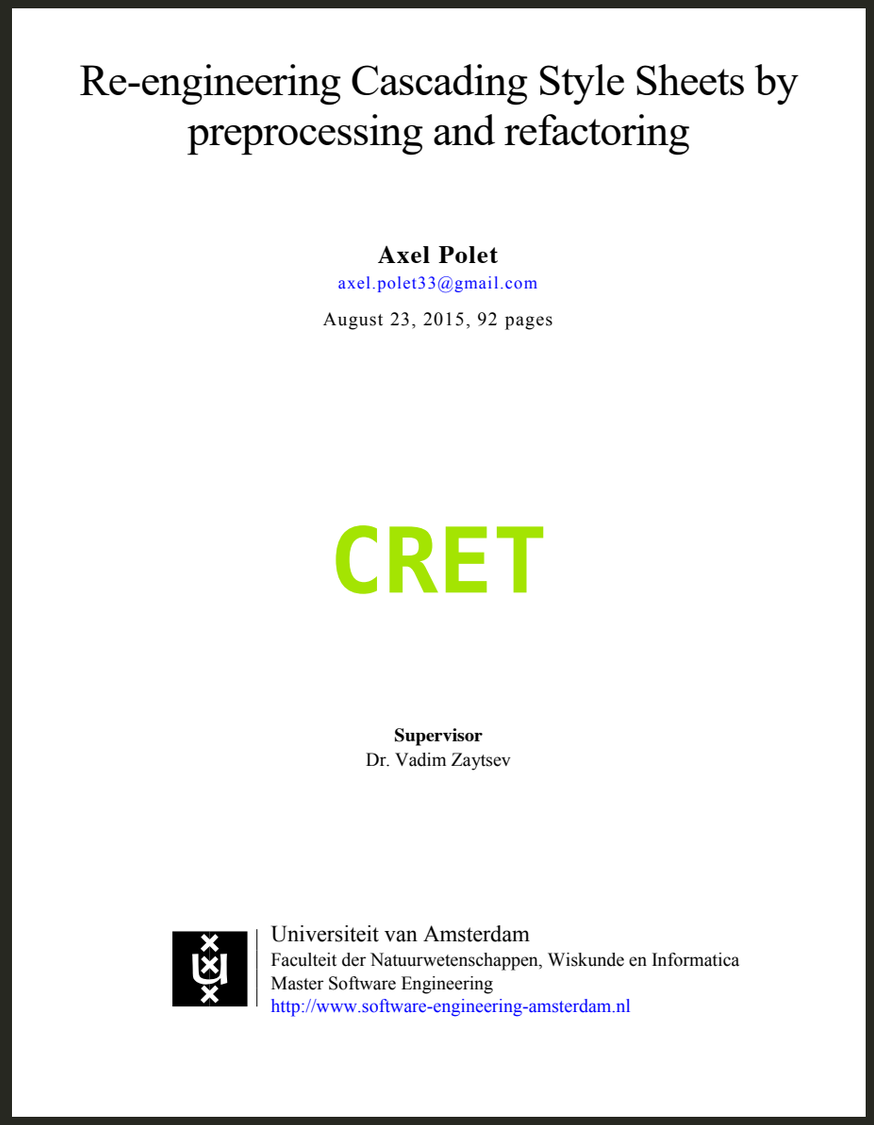
Language Conversion



A.A.Terekhov, C.Verhoef, The Realities of Language Conversions, IEEE Software 2000.

Up-compilation

* CSS to SASS
* ~70% less code
* ~5% less padding
* ~10% in mixins
* ~8% to children
* ~2 CSS decls per SASS var



Re-engineering Cascading Style Sheets by preprocessing and refactoring

Axel Polet
axel.polet33@gmail.com

August 23, 2015, 92 pages

CRET

Supervisor
Dr. Vadim Zaytsev

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Stay tuned:

guest lecture

next week