NPRG075

Heuristic evaluation of programming systems

Tomáš Petříček, 204 (2nd floor)

✉ petricek@d3s.mff.cuni.cz

➔ https://tomasp.net | @tomaspetricek

Lectures: Tuesday 12:20, S6

➔ https://d3s.mff.cuni.cz/teaching/nprg075

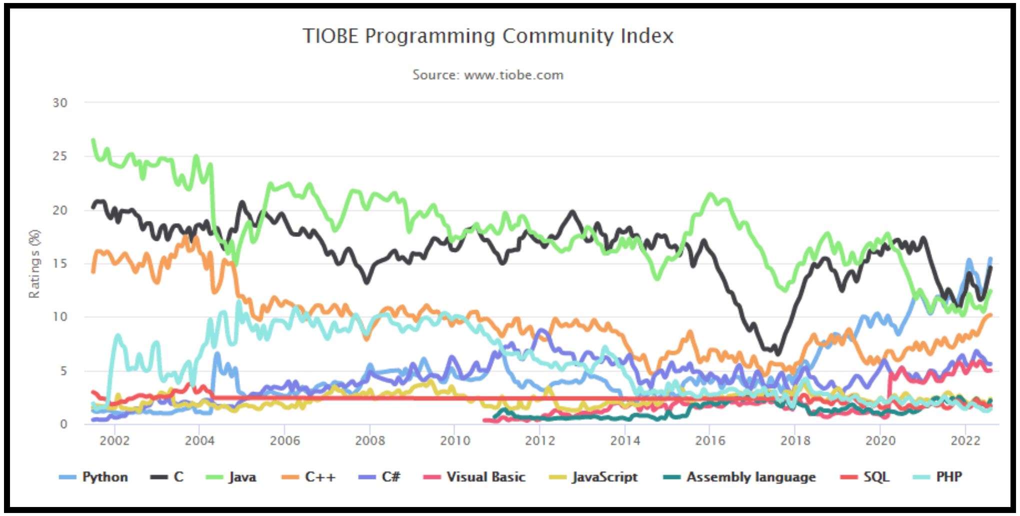
Programming systems

What really matters?

Programming systems

What can we study?

- Formal semantics and type safety
- Learnability for novice programmers
- Socio-technical context of the system
- Principles behind the system design


TIOBE Programming Community Index

Source: www.tiobe.com

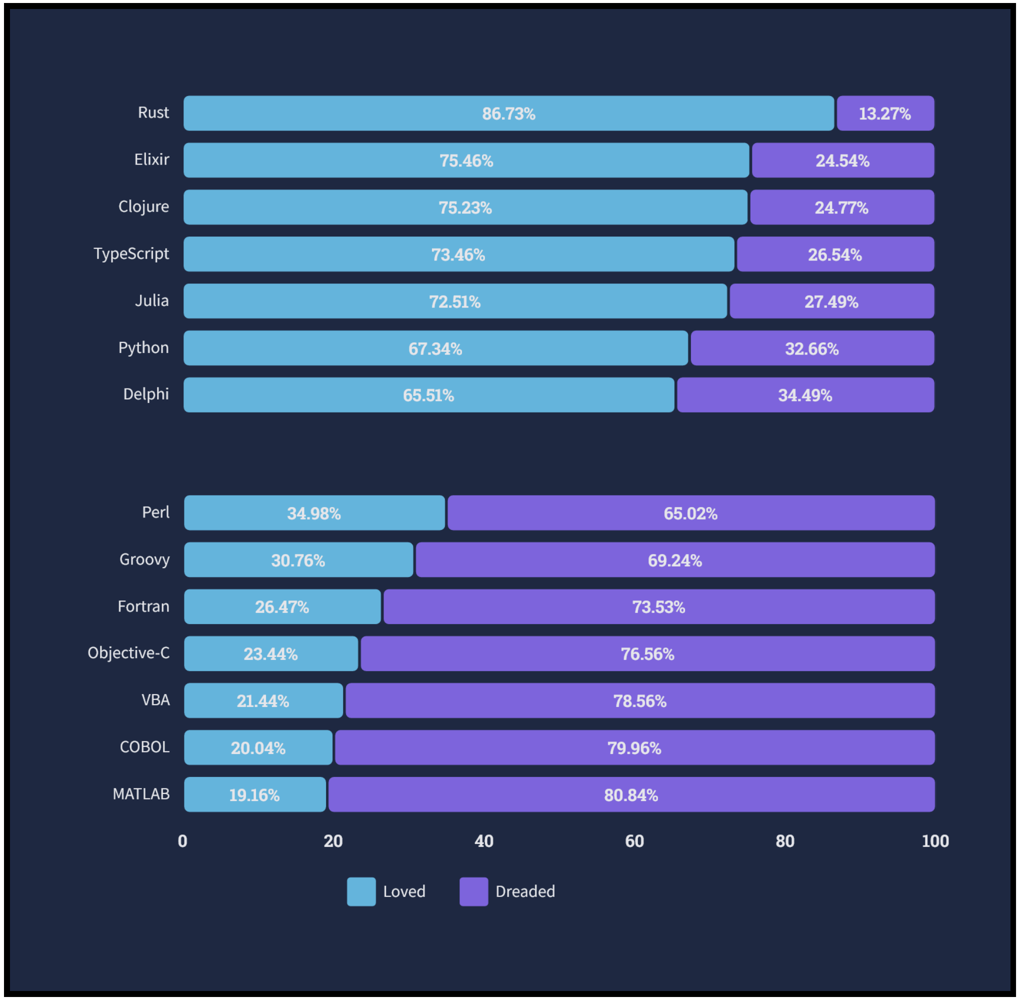
What makes a language popular

None of the things we talked about?

Popular ≠ Good

The index has its flaws

Still, a reason to think!



Most loved or most dreaded?

Enthusiastic community?
Good tooling?
Clean idea?
Practicality?

Need to talk about less exact things!

Analysis of language perceptions

Survey analysis

- Survey of language characteristics
- Feature and language correlations
- tinyurl.com/nprg075-socio

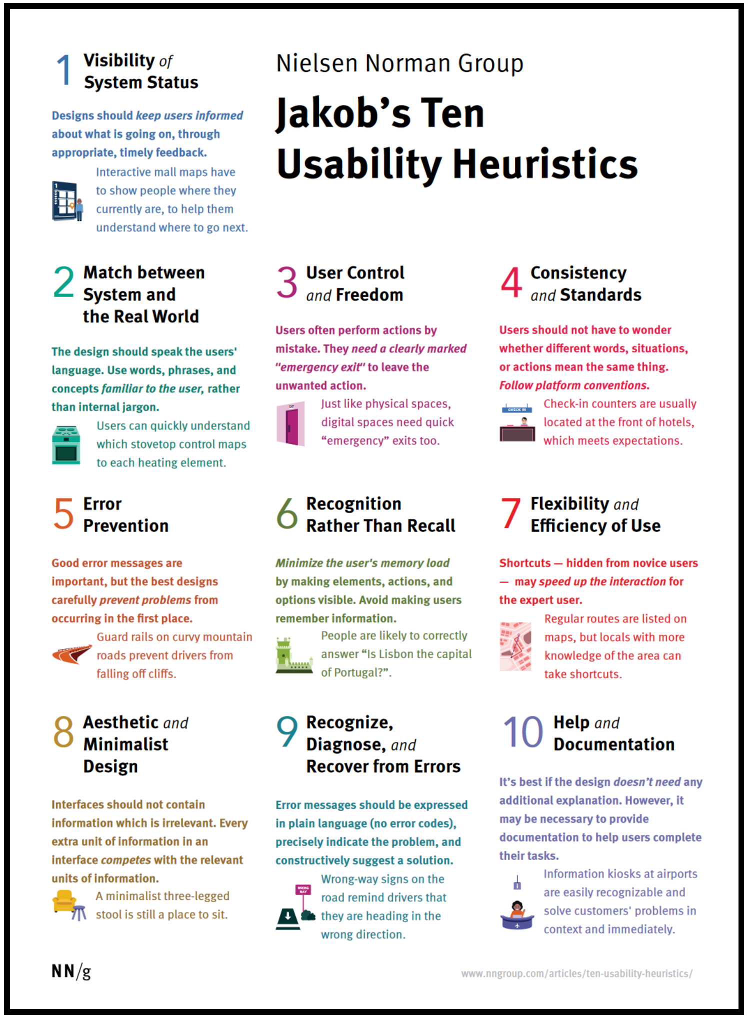
Adoption of languages

- Libraries matter
- Legacy and history matter
- Flexibility more important than correctness

Programming systems

Important but hard to study

- Expressivity of the programming notation
- Unifying conceptual model ("everything is ...")
- Style of interaction with the system
- Extensibility and flexibility of the language



Nielsen Norman Group

Jakob's Ten Usability Heuristics

1 Visibility *of* System Status

Designs should *keep users informed* about what is going on, through appropriate, timely feedback.

Interactive mall maps have to show people where they currently are, to help them understand where to go next.

2 Match between System and the Real World

The design should speak the users' language. Use words, phrases, and concepts *familiar to the user*, rather than internal jargon.

Users can quickly understand which stovetop control maps to each heating element.

3 User Control *and* Freedom

Users often perform actions by mistake. They *need a clearly marked "emergency exit"* to leave the unwanted action.

Just like physical spaces, digital spaces need quick "emergency" exits too.

4 Consistency *and* Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. *Follow platform conventions.*

Check-in counters are usually located at the front of hotels, which meets expectations.

5 Error Prevention

Good error messages are important, but the best designs carefully *prevent problems* from occurring in the first place.

Guard rails on curvy mountain roads prevent drivers from falling off cliffs.

6 Recognition Rather Than Recall

Minimize the user's memory load by making elements, actions, and options visible. Avoid making users remember information.

People are likely to correctly answer "Is Lisbon the capital of Portugal?".

7 Flexibility *and* Efficiency of Use

Shortcuts — hidden from novice users — may *speed up the interaction* for the expert user.

Regular routes are listed on maps, but locals with more knowledge of the area can take shortcuts.

8 Aesthetic *and* Minimalist Design

Interfaces should not contain information which is irrelevant. Every extra unit of information in an interface *competes* with the relevant units of information.

A minimalist three-legged stool is still a place to sit.

9 Recognize, Diagnose, *and* Recover from Errors

Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.

Wrong-way signs on the road remind drivers that they are heading in the wrong direction.

10 Help *and* Documentation

It's best if the design *doesn't need* any additional explanation. However, it may be necessary to provide documentation to help users complete their tasks.

Information kiosks at airports are easily recognizable and solve customers' problems in context and immediately.

NN/g

www.nngroup.com/articles/ten-usability-heuristics/

Heuristic analysis

High-level rules, characteristics or principles

Developed by experts, based on reviews and experience

Useful for evaluation, classifying, analysis, new design

Programming systems

Heuristic frameworks

- Levels of liveness of programming systems
- Memory models of programming languages
- Cognitive dimensions of notation
- Technical dimensions of programming systems

Programming systems

Liveness and memory models

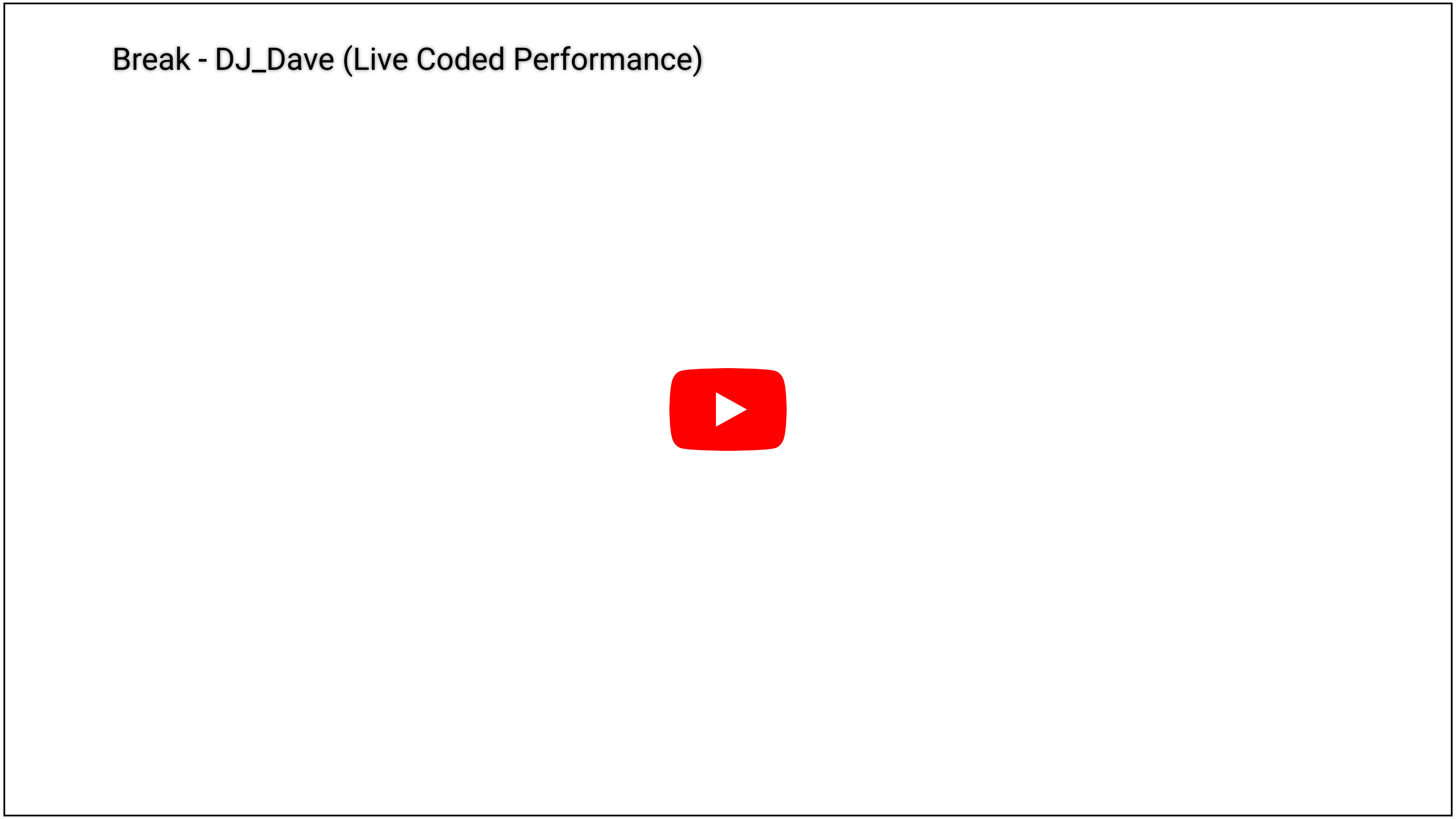
From batch processing ...

Coding at the computer prohibitively expensive

Write program, punch on cards, submit & wait

A few day feedback cycle!

… to live coded music performance



Break - DJ_Dave (Live Coded Performance)

Visual programming

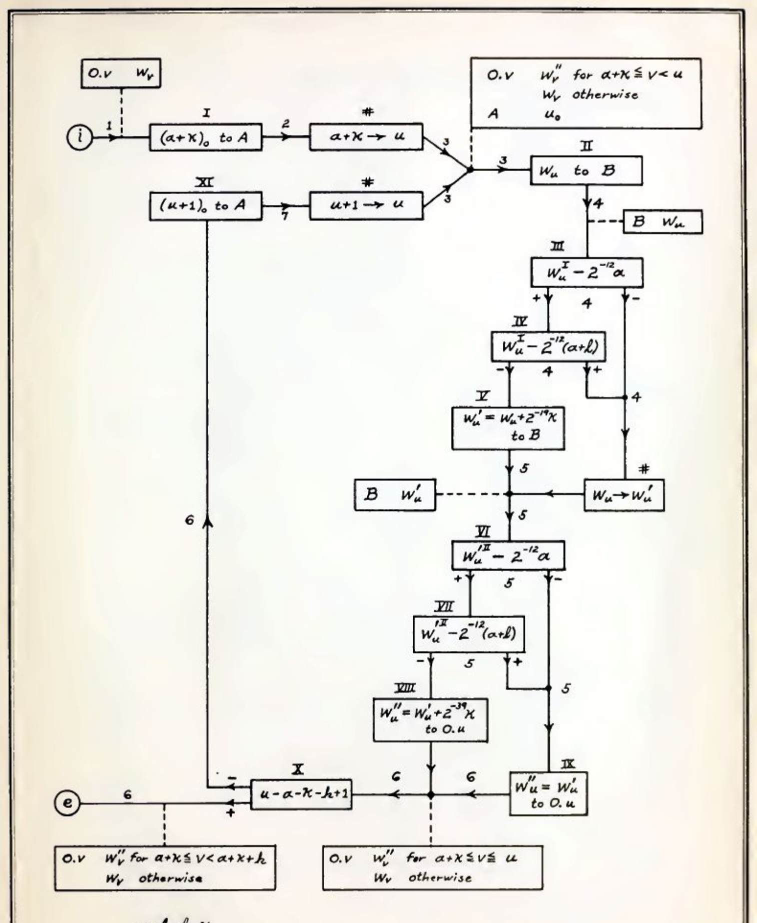


FIGURE 12.1

Planning and coding of problems for an electronic computing instrument (Goldstine, von Neumann, 1948)

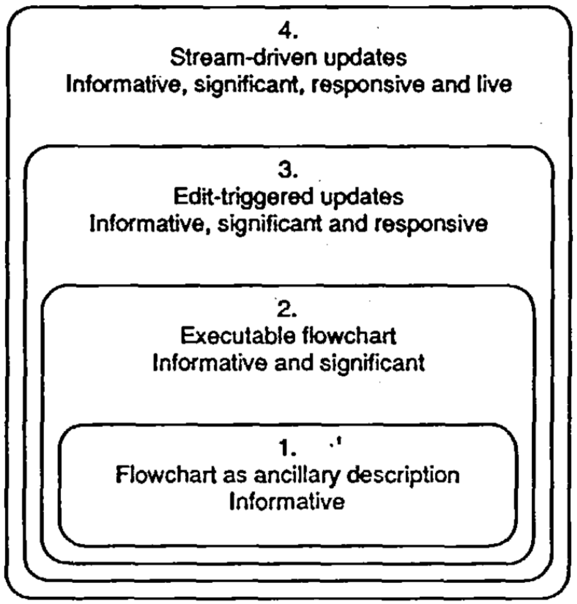

4.
Stream-driven updates
Informative, significant, responsive and live
3.
Edit-triggered updates
Informative, significant and responsive
2.
Executable flowchart
Informative and significant
1.
Flowchart as ancillary description
Informative

Figure 2. Levels of 'liveness' in visual programming systems

Liveness levels

(Tanimoto, 1990)

Level 1

Flowchart that exists independently of a program

Level 4

Continuous processing with immediate dynamic change of behaviour

Liveness levels

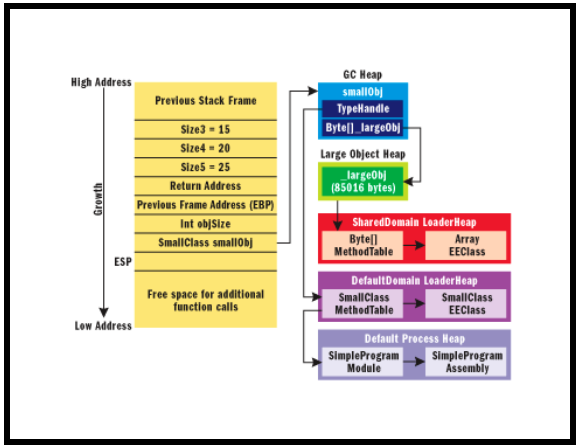
Programming system heuristic

- Single property of specific systems
- Can be used for comparing systems
- Imagines step beyond the state-of-the-art
- Can be used for designing new systems

Memory models of systems

Primary representation

- How things are represented
- Defines what can be done
- Defines how to think!



Six major conceptualizations

- COBOL, LISP and FORTRAN
- SQL, UNIX and tape storage
- In reality, it's always a mix!

Language memory models

- COBOL - Memory is a nested record (tax form)
  No need for pointers, but no sharing allowed
- LISP - Memory is an object graph (symbol list)
  Flexible, but serialization & efficiency tricky
- FORTRAN - Memory is a bunch of arrays (vector)
  Close to the metal, but no semantic checking

Storage memory models

PIPES - Magnetic tape model (I/O streams)
Specific, but great for some problems (MapReduce)

MULTICS - Tree with blob leaves (file system)
Legible, allows separation; rarely used in full

SQL - Memory is a set of relations (tables)
Expressive query language, c.f. Prolog and similar

Memory models

Programming system heuristic

- Single property of any programming system
- Categorical rather than ordinal
- Sheds light on what exists
- Open to questioning, e.g., is that all there is?

Notations

Cognitive dimensions

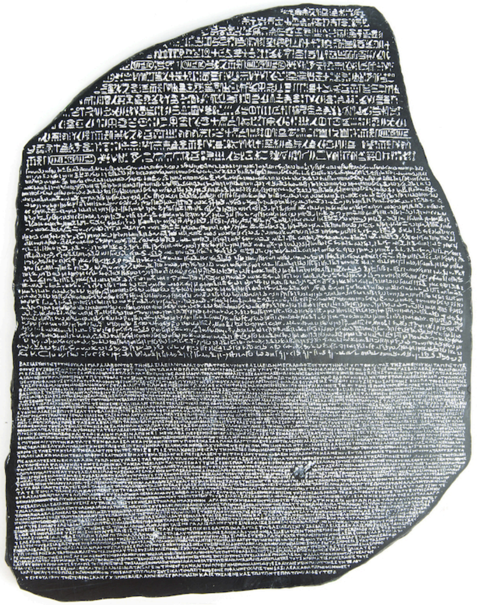
Notations and humans

Notations in computing

- Programming languages
- Markup and config files
- Rule and macro editors

User experience questions

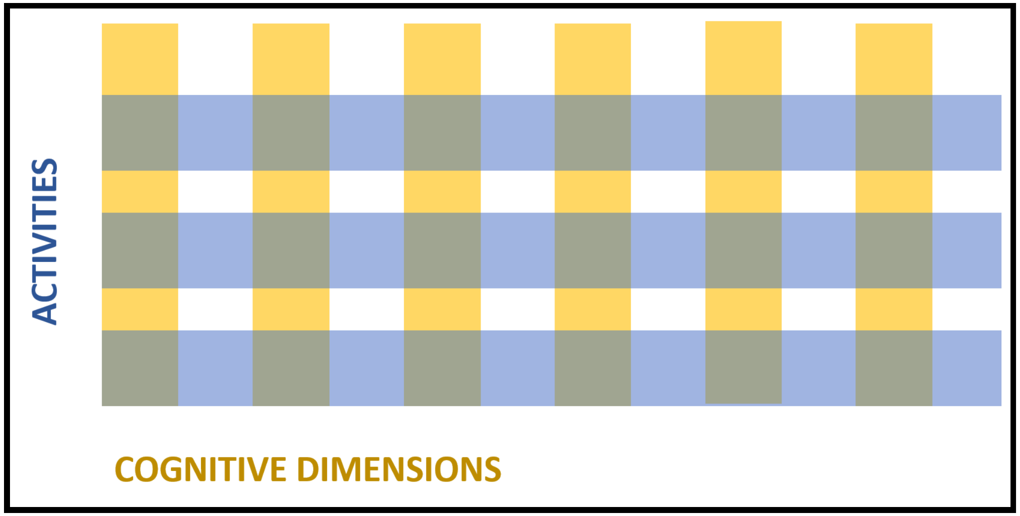
- Does the notation structure support activities of the user?
- Is one notation the best?



Cognitive dimensions

Programming system heuristic

- Comprehensible broad-brush evaluation
- Understandable for non-specialists
- Distinguish different user needs
- Prompt designers to see more choices



Dimensions × Activities

Variety of dimensions
For a given activity

Activities

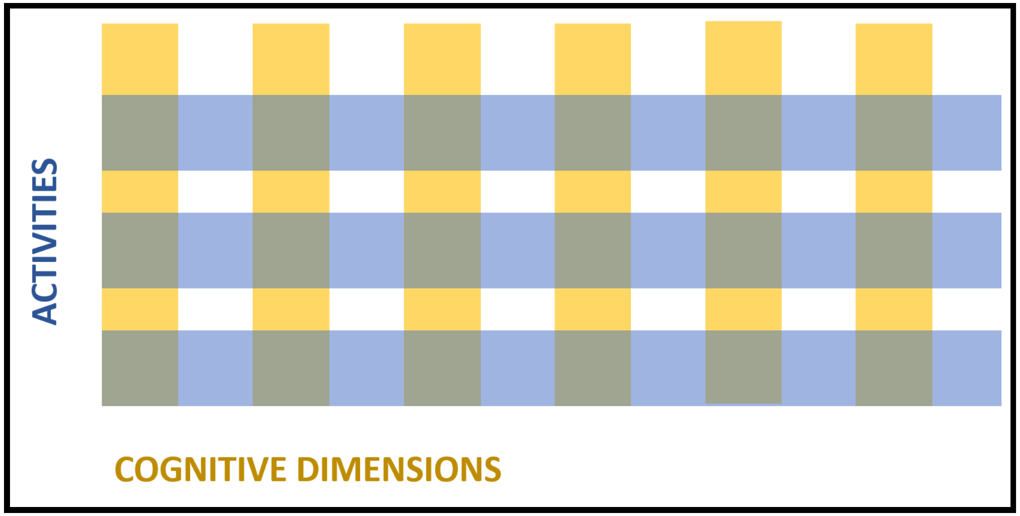
Generic activities involving notations

Each has different notational needs

Activities with different needs

- Incrementation - adding formulas to spreadsheet
- Transcription - copying data from paper
- Modification - changing formula in a spreadsheet
- Exploratory design - designing software structure
- Searching - finding uses of a function
- Exploratory understanding - understanding code

Dimensions × Activities



Variety of dimensions
For a given activity

Dimensions

Characteristic
of the notation

Human-computer
interaction analysis
perspective

Example cognitive dimensions (1/2)

- Viscosity - Resistance to change
- Visibility - Ability to view components easily
- Premature commitment - Need to decide too early
- Hidden dependencies - Important links not visible
- Role-expressiveness - Purpose of an entity is clear

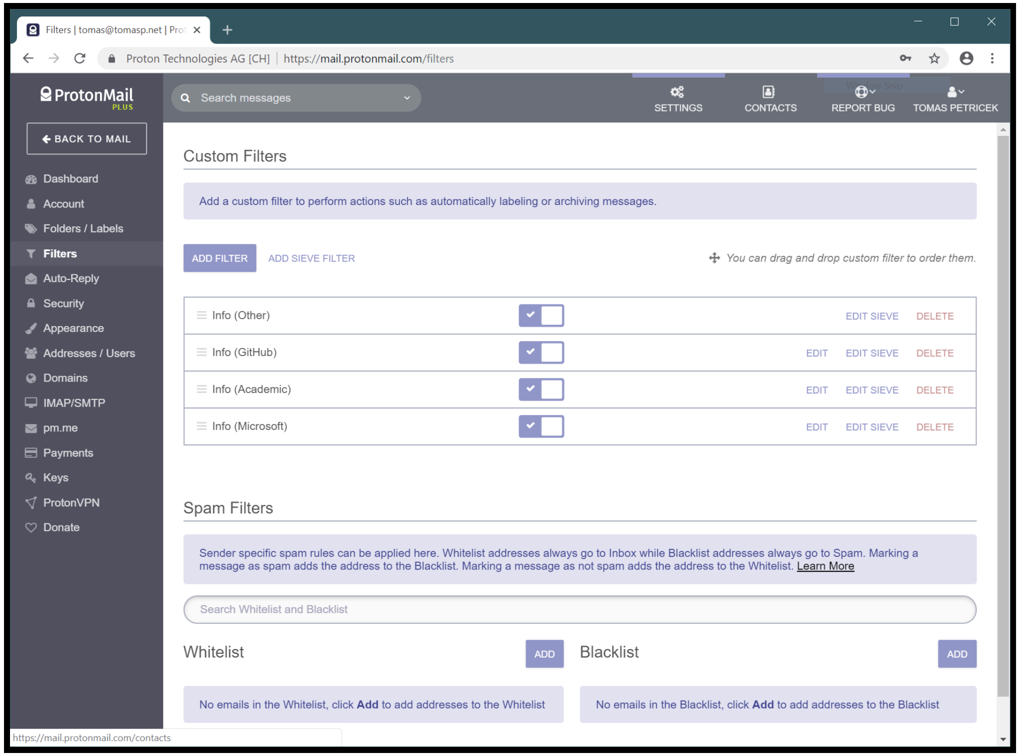
Example cognitive dimensions (2/2)

- Error-proneness - Notation invites mistakes
- Abstraction - Types and availability of mechanisms
- Consistency - Similar syntax has similar semantics
- Diffuseness - Verbosity of language
- Hard mental operations - High cognitive demand

Case study

Two ways of specifying email filters

Visual rule editor vs. scripting language



Custom Filters

Add a custom filter to perform actions such as automatically labeling or archiving messages.

You can drag and drop custom filter to order them.

Info (Other)	☑	EDIT SIEVE DELETE
Info (GitHub)	☑	EDIT EDIT SIEVE DELETE
Info (Academic)	☑	EDIT EDIT SIEVE DELETE
Info (Microsoft)	☑	EDIT EDIT SIEVE DELETE

Spam Filters

Sender specific spam rules can be applied here. Whitelist addresses always go to Inbox while Blacklist addresses always go to Spam. Marking a message as spam adds the address to the Blacklist. Marking a message as not spam adds the address to the Whitelist. Learn More

Search Whitelist and Blacklist

Whitelist

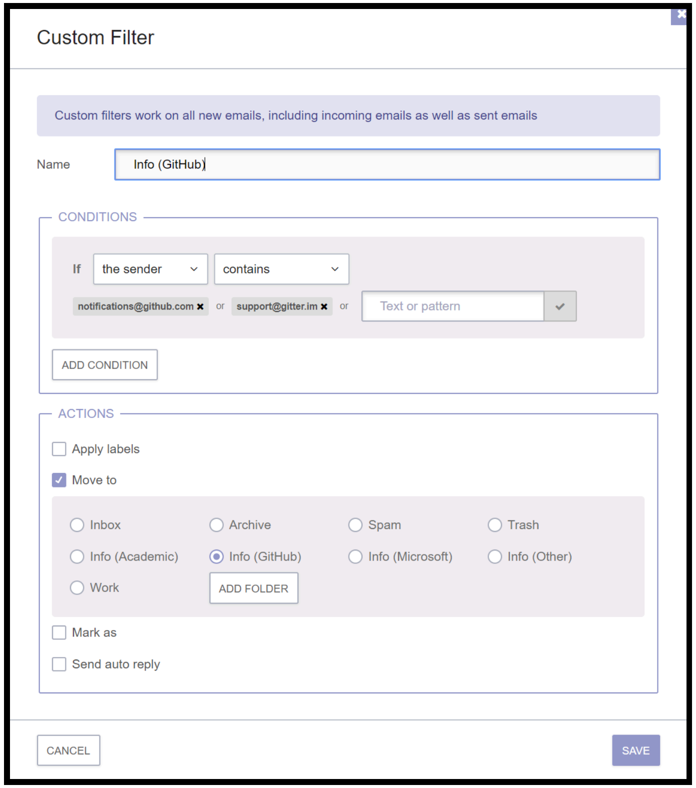
No emails in the Whitelist, click **Add** to add addresses to the Whitelist

Blacklist

No emails in the Blacklist, click **Add** to add addresses to the Blacklist

Two ways of specifying email filters

Visual editor



Custom Filter

Custom filters work on all new emails, including incoming emails as well as sent emails

Name: Info (GitHub)

CONDITIONS

If the sender contains

notifications@github.com ✖ or support@gitter.im ✖ or

ACTIONS

□ Apply labels

☑ Move to

○ Inbox ○ Archive ○ Spam ○ Trash

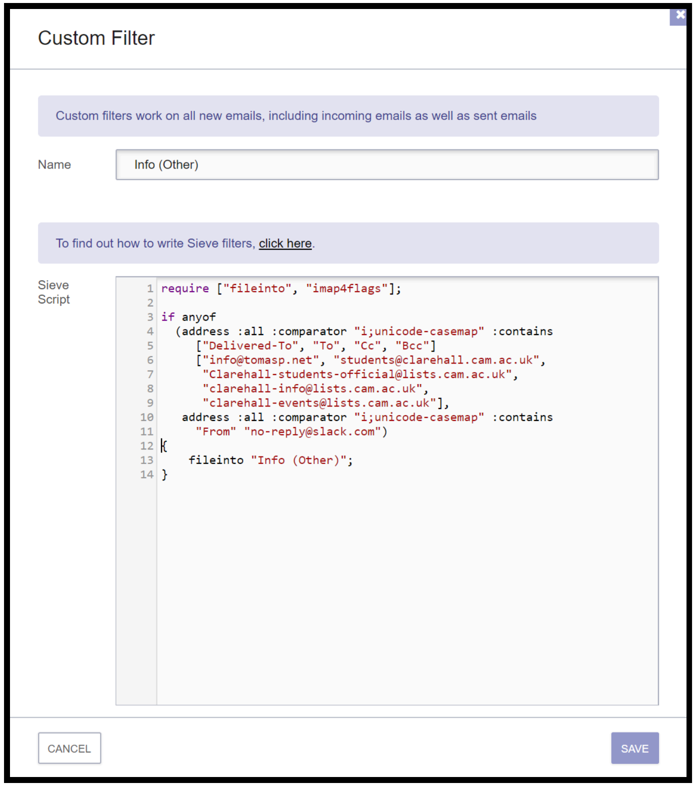
○ Info (Academic) ● Info (GitHub) ○ Info (Microsoft) ○ Info (Other)

○ Work

□ Mark as

□ Send auto reply

Scripting language



Custom Filter

Custom filters work on all new emails, including incoming emails as well as sent emails

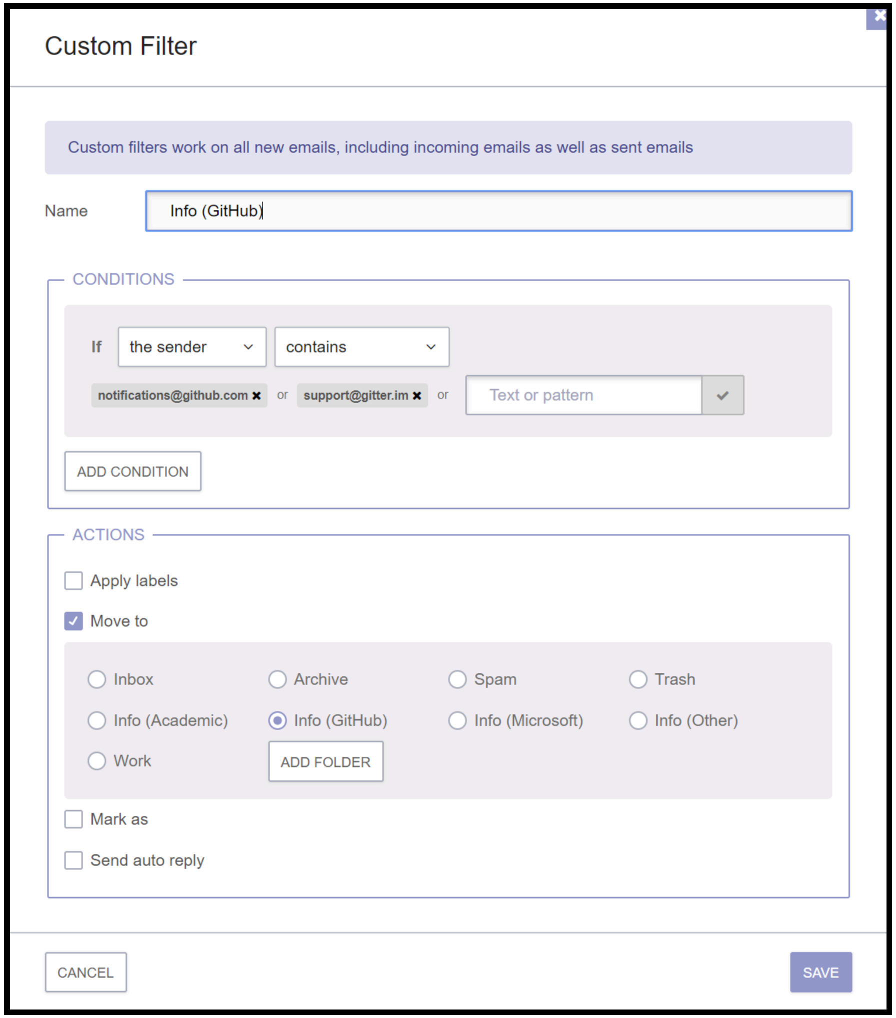
Name: Info (Other)

To find out how to write Sieve filters, click here.

Sieve Script

```
require ["fileinto", "imap4flags"];

if anyof
  (address :all :comparator "i;unicode-casemap" :contains
     ["Delivered-To", "To", "Cc", "Bcc"]
     ["info@tomasp.net", "students@clarehall.cam.ac.uk",
      "Clarehall-students-official@lists.cam.ac.uk",
      "clarehall-info@lists.cam.ac.uk",
      "clarehall-events@lists.cam.ac.uk"],
   address :all :comparator "i;unicode-casemap" :contains
     "From" "no-reply@slack.com")
{
    fileinto "Info (Other)";
}
```



Custom Filter

Custom filters work on all new emails, including incoming emails as well as sent emails

Name: Info (GitHub)

CONDITIONS

If the sender contains

notifications@github.com ✖ or support@gitter.im ✖ or Text or pattern

ACTIONS

- □ Apply labels
- ☑ Move to
  - ○ Inbox ○ Archive ○ Spam ○ Trash
  - ○ Info (Academic) ● Info (GitHub) ○ Info (Microsoft) ○ Info (Other)
  - ○ Work
- □ Mark as
- □ Send auto reply

Incrementation

Adding new condition

Viscosity

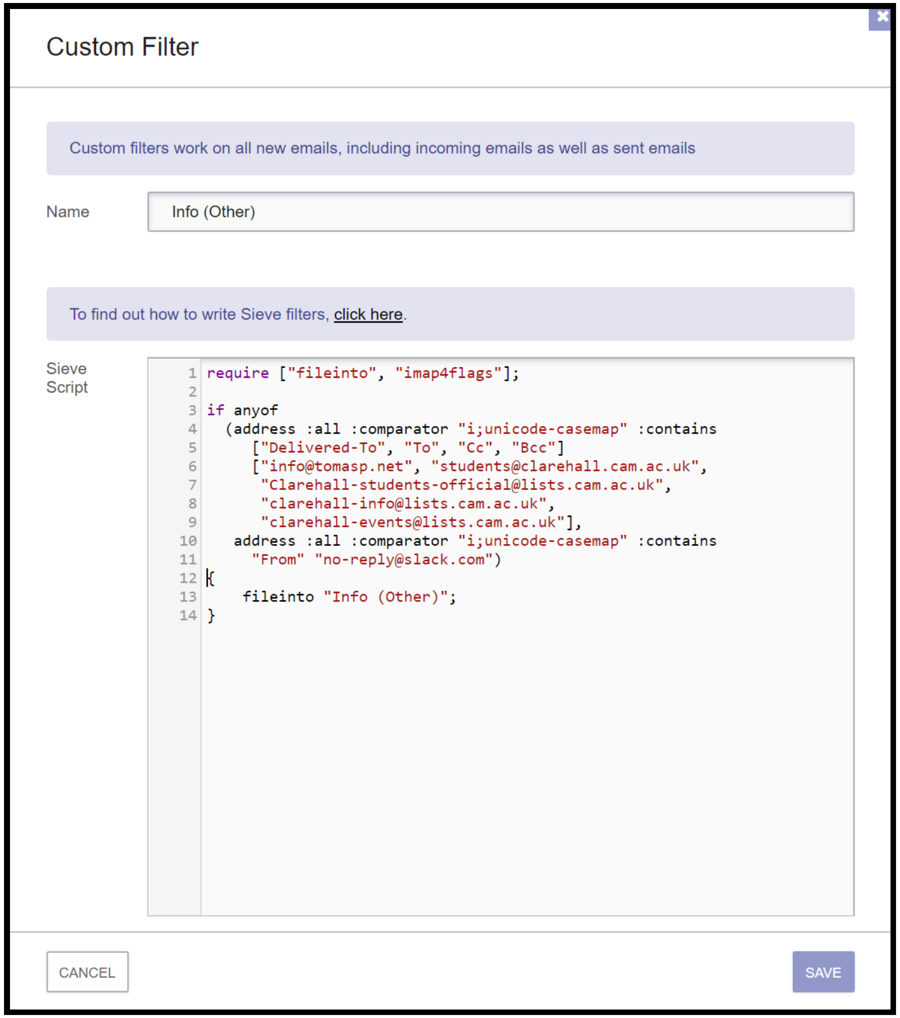
Not all additions possible

Abstraction

Condition format is fixed

Hard mental operations

Everything is simple & clear



Custom Filter

Custom filters work on all new emails, including incoming emails as well as sent emails

Name: Info (Other)

To find out how to write Sieve filters, click here.

Sieve Script

```
require ["fileinto", "imap4flags"];

if anyof
  (address :all :comparator "i;unicode-casemap" :contains
     ["Delivered-To", "To", "Cc", "Bcc"]
     ["info@tomasp.net", "students@clarehall.cam.ac.uk",
      "Clarehall-students-official@lists.cam.ac.uk",
      "clarehall-info@lists.cam.ac.uk",
      "clarehall-events@lists.cam.ac.uk"],
   address :all :comparator "i;unicode-casemap" :contains
     "From" "no-reply@slack.com")
{
    fileinto "Info (Other)";
}
```

Incrementation

Adding new condition

Viscosity

Edit text for any change

Abstraction

Possible via a script

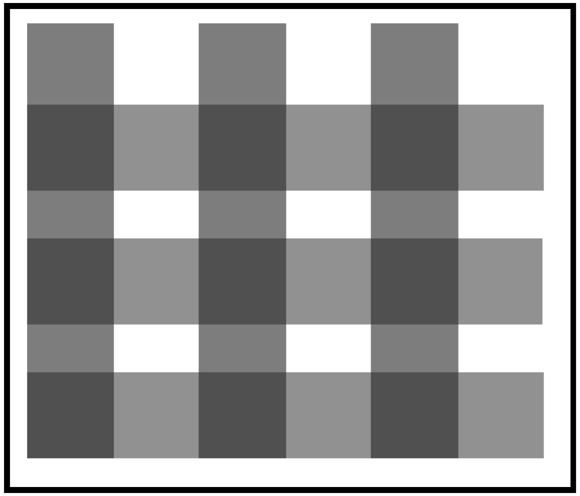
Hard mental operations

Understanding code is hard

Two ways of specifying filters

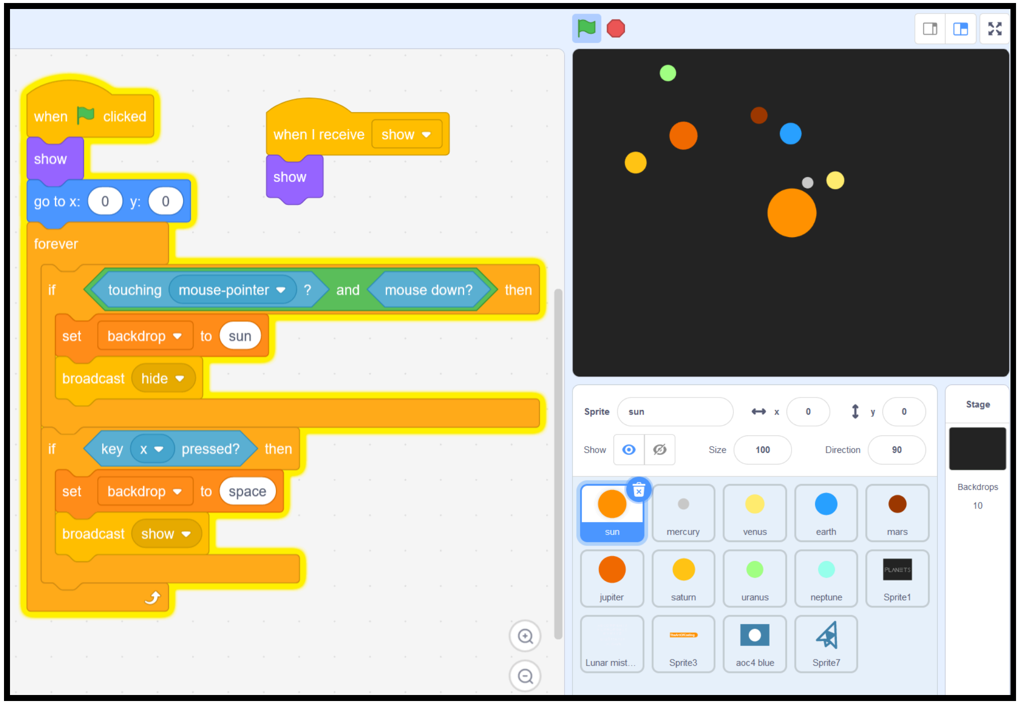
Cognitive dimensions

- Used for evaluation
- Consider activities & dimensions
- Clear lists to use



What is a better notation?

- Wrong question: different trade-offs!
- UI is viscose, less abstract, but simpler
- Script has abstractions, less viscose, but harder



```
when ⚑ clicked
show
go to x: 0 y: 0
forever
    if touching mouse-pointer ? and mouse down? then
        set backdrop to sun
        broadcast hide
    if key x pressed? then
        set backdrop to space
        broadcast show
```

```
when I receive show
show
```

Sprite: sun   x: 0   y: 0

Show   Size: 100   Direction: 90

Block based visual languages

Contrast with text for addition (writing code)

Premature commit

Diffuseness / verbosity

Abstraction

Error-proneness

Programming systems

Technical dimensions

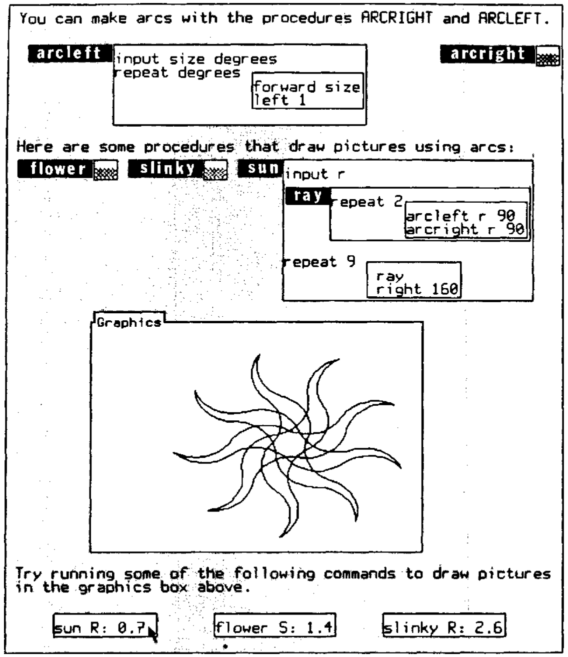
From languages to systems

Programming system is

Integrated and complete set of tools sufficient for creating, modifying, and executing programs

These will include

Notations for structuring programs and data, facilities for running and debugging programs, and interfaces for performing all of these tasks.



You can make arcs with the procedures ARCRIGHT and ARCLEFT.

```
arcleft
input size degrees
repeat degrees
    forward size
    left 1
```

Here are some procedures that draw pictures using arcs:

```
sun
input r
ray
  repeat 2
      arcleft r 90
      arcright r 90
repeat 9
    ray
    right 160
```

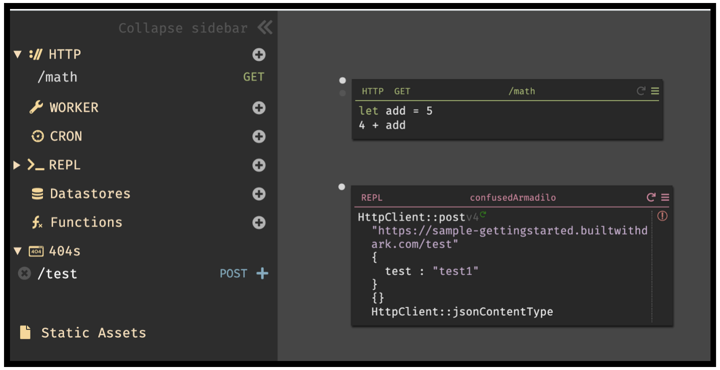
Try running some of the following commands to draw pictures in the graphics box above.

```
sun R: 0.7        flower S: 1.4        slinky R: 2.6
```

Interesting programming systems

Research and industry

- Low-code and no-code startups
- Live & interactive systems
- Interesting code editors

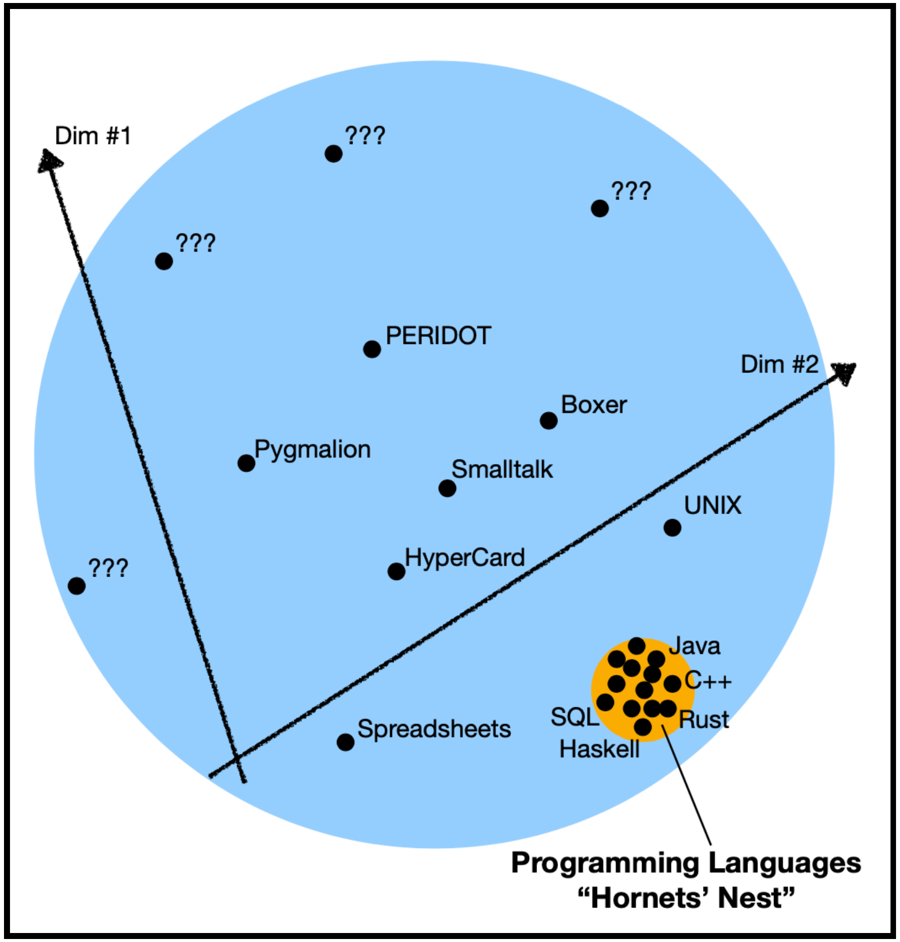


```
let add = 5
4 + add
```

```
HttpClient::postv4
  "https://sample-gettingstarted.builtwithdark.com/test"
  {
    test : "test1"
  }
  {}
  HttpClient::jsonContentType
```

How do we talk about these?

- Difficult to say what is new
- Hard to look beyond the interface
- Programming systems deserve a theory too!



Technical dimensions

Based on analysis of past and modern systems

Capture their key characteristics

Describe a range of possible values

Descriptive, not prescriptive

Technical dimensions catalogue

Interaction

Feedback Loops
Modes of interaction
Abstraction Construction

Notation

Notational Structure
Surface/Internal
Primary/Secondary
Expression Geography
Uniformity

Error Handling

Error Detection
Error Response

Conceptual Structure

Integrity/Openness
Composability
Convenience
Commonality

Customizability

Staging
Externalizability
Additive Authoring
Self-Sustainability

(Others)

Degrees of Automation
Learnability & Sociability

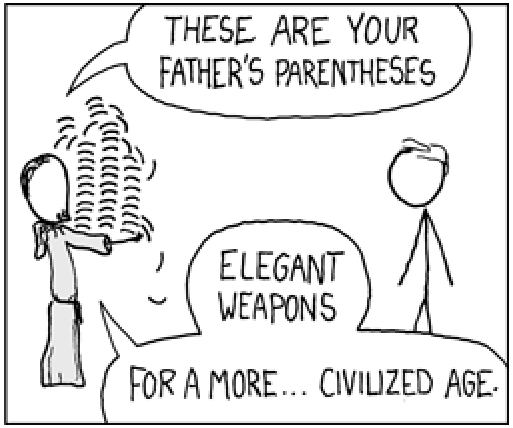
Notational uniformity

Post-modernist

- Variety of different notations
- More to learn, but better problem fit
- Perl language, Web platform

Modernist

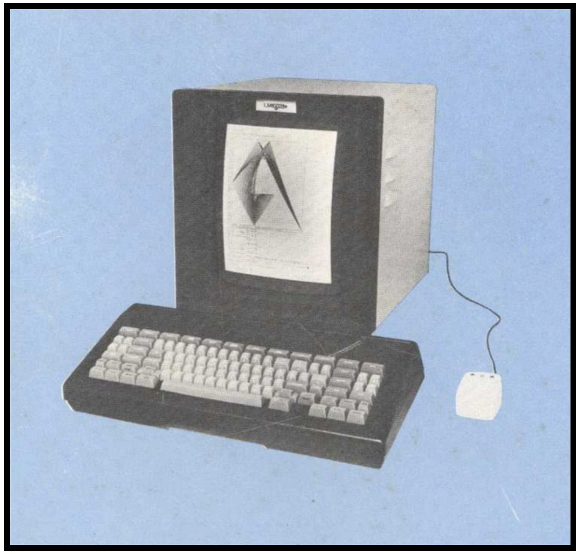
- Small set of uniform primitives
- Not everything fits the notation
- Lisp and (partly) Smalltalk


THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

Self-sustainability

Separate language level

- Implementation vs. user level
- Limited changeability from within
- Java and other languages



Integrated systems design

- Implemented & modifiable in itself
- Often changeable at runtime
- Smalltalk, Lisp Machines

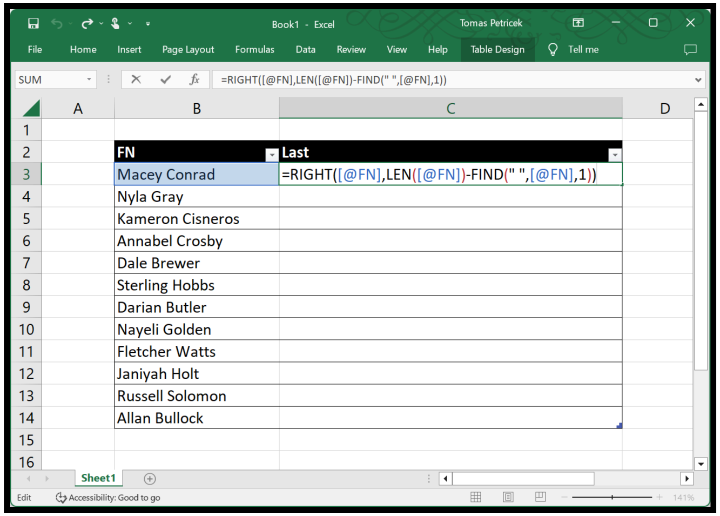
Abstraction construction

From Concrete

- Generalize from examples
- Expanding range in Excel
- Pygmalion system

From Abstract

- Define function first
- Most programming languages
- Coding done without values



```
=RIGHT([@FN],LEN([@FN])-FIND(" ",[@FN],1))
```

	A	B	C	D
1				
2		FN	Last	
3		Macey Conrad	=RIGHT([@FN],LEN([@FN])-FIND(" ",[@FN],1))	
4		Nyla Gray		
5		Kameron Cisneros		
6		Annabel Crosby		
7		Dale Brewer		
8		Sterling Hobbs		
9		Darian Butler		
10		Nayeli Golden		
11		Fletcher Watts		
12		Janiyah Holt		
13		Russell Solomon		
14		Allan Bullock		
15				
16				

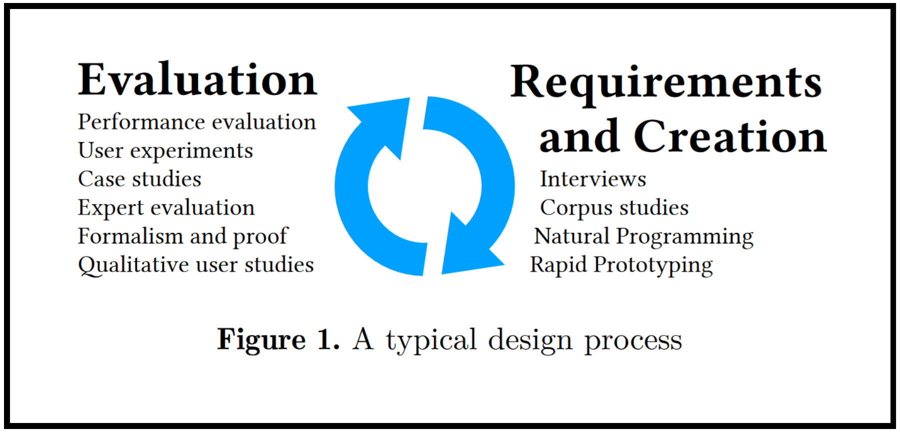
Technical dimensions

Programming system heuristic

- Making sense of different systems
- Broad strokes and high-level
- Useful for making comparisons
- Useful for finding gaps in design space

Conclusions

Heuristic analysis



Evaluation

Performance evaluation
User experiments
Case studies
Expert evaluation
Formalism and proof
Qualitative user studies

Requirements and Creation

Interviews
Corpus studies
Natural Programming
Rapid Prototyping

Figure 1. A typical design process

Heuristic analysis of languages

Both idea generation and evaluation

Depends on the kind of heuristic

Categorical allows questioning

Ordinal allows for degree comparison

Reading

CDs in the real-world!

- A Usability Analysis of Blocks-based Programming Editors using Cognitive Dimensions
- tinyurl.com/nprg075-blocks (SciHub)

Why read this paper

- Example of rigorous analysis
- Based on a user study
- Equally possible with expert assessment

Conclusions

Heuristic evaluation of programming systems

- Memory (categorical) and liveness (ordinal)
- Cognitive and technical dimension frameworks
- Broad-brush map of the design space
- Useful for evaluation and novel design ideas

Tomáš Petříček, 204 (2nd floor)
✉ petricek@d3s.mff.cuni.cz
➜ https://tomasp.net | @tomaspetricek
➜ https://d3s.mff.cuni.cz/teaching/nprg075

References (1/2)

Live visual programming

- Tanimoto, S. L. (1990). VIVA: A visual language for image processing, Journal on Visual Languages
- Tanimoto, S. L. (2013). A Perspective on the Evolution of Live Programming, LIVE

Language adoption & Heuristics

- Meyerovich, L. A., Rabkin, A. S. (2013). Empirical Analysis of Programming Language Adoption, OOPSLA
- Meyerovich, L. A., Rabkin, A. S. (2012). Socio-PLT: Sociological Principles for Programming Language Adoption, Onward!
- Nielsen, J. (1994). 10 Usability Heuristics for User Interface Design. Norman-Nielsen Group

References (2/2)

Cognitive, technical & memory models

- Sitaker, K. J. (2016). The memory models that underlie programming languages, Online
- Jakubovic, J. et al. (2023). Technical Dimensions of Programming Systems, Programming
- Holwerda, R., Hermans, F. (2018). A usability analysis of blocks-based programming editors using cognitive dimensions, VL/HCC
- Blackwell, A., Green, T. (2002). Notational Systems – the Cognitive Dimensions of Notations framework. (Chapter)

A bit of history

- Goldstine, H., von Neumann, J. (1947). Planning and coding of problems for an electronic computing, Princeton