Convivial design heuristics for user-facing software (of the programmable kind)

Stephen Kell

S.R.Kell@kent.ac.uk

May 2020

‘Just let me rule you. . . ’

Design heuristics?

Software has many established ‘design heuristics’

- prefer higher-level languages
- hide change-prone information
- prefer to write portable code
- maximise compositionality
- follow these paradigmatic examples: . . .

. . . drawn mostly from a mixture of

- academic research
- textbooks
- folklore

When good ideas go bad. . .

‘A few patients survived longer with transplants of various organs. On the other hand, the total social cost exacted by medicine ceased to be measurable in conventional terms. Society can have no quantitative standards by which to add up the illusion, social control, prolonged suffering, loneliness, genetic deterioration, and frustration produced by medical treatment.’

—from ‘Tools for Conviviality’ (1973)

The basic premise of this talk:

> The conventional 'design heuristics' of programming languages/systems, and much infrastructure built with them, have proven unconvivial.

So what design heuristics might be better?

Recurring themes:

- lessen the extent of *dominance*
- . . . even if it lessens overall 'power'

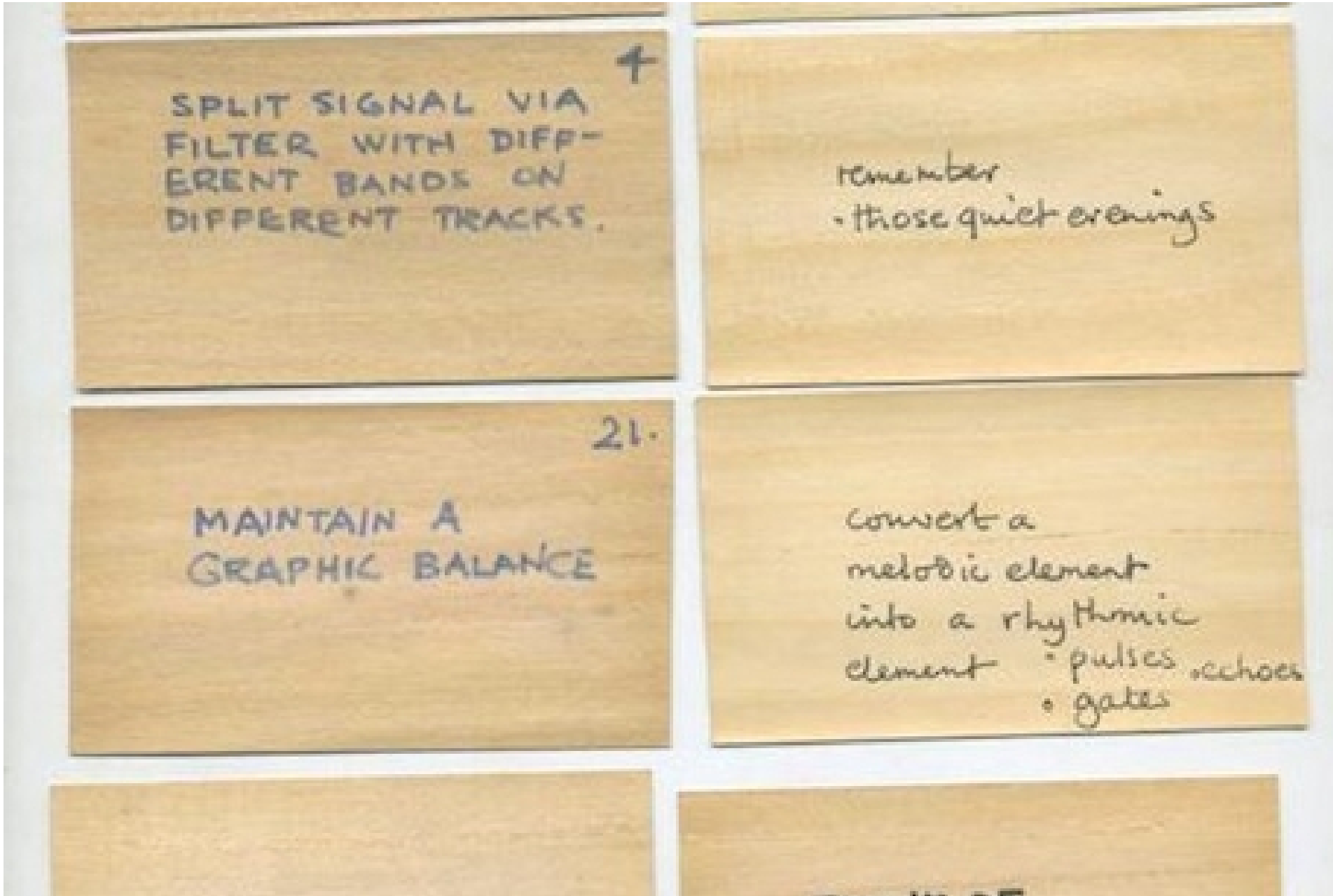
Also: how can we *establish* these heuristics in practice?

Inverting the structure. . .

‘The crisis can be solved only if we learn to invert the present deep structure of tools; if we give people tools that guarantee their right to work with high, independent efficiency . . . eliminating the need for either slaves or masters . . . People need new tools to work with rather than tools that ‘work’ for them.’

—from ‘Tools for Conviviality’ (1973)

Status of this work: oblique strategies



1. ‘Like I could. . . take on the world!’

World domination is ‘expected’

. . . especially by programming language designers.

Currently: becoming useful requires becoming dominant.

Why? Libraries, mostly.

Libraries must exist *inside* the language.

Standard rant: consider parser generators. . .

Instead: what if. . . ?

. . . design languages to be useful in *existing* contexts?

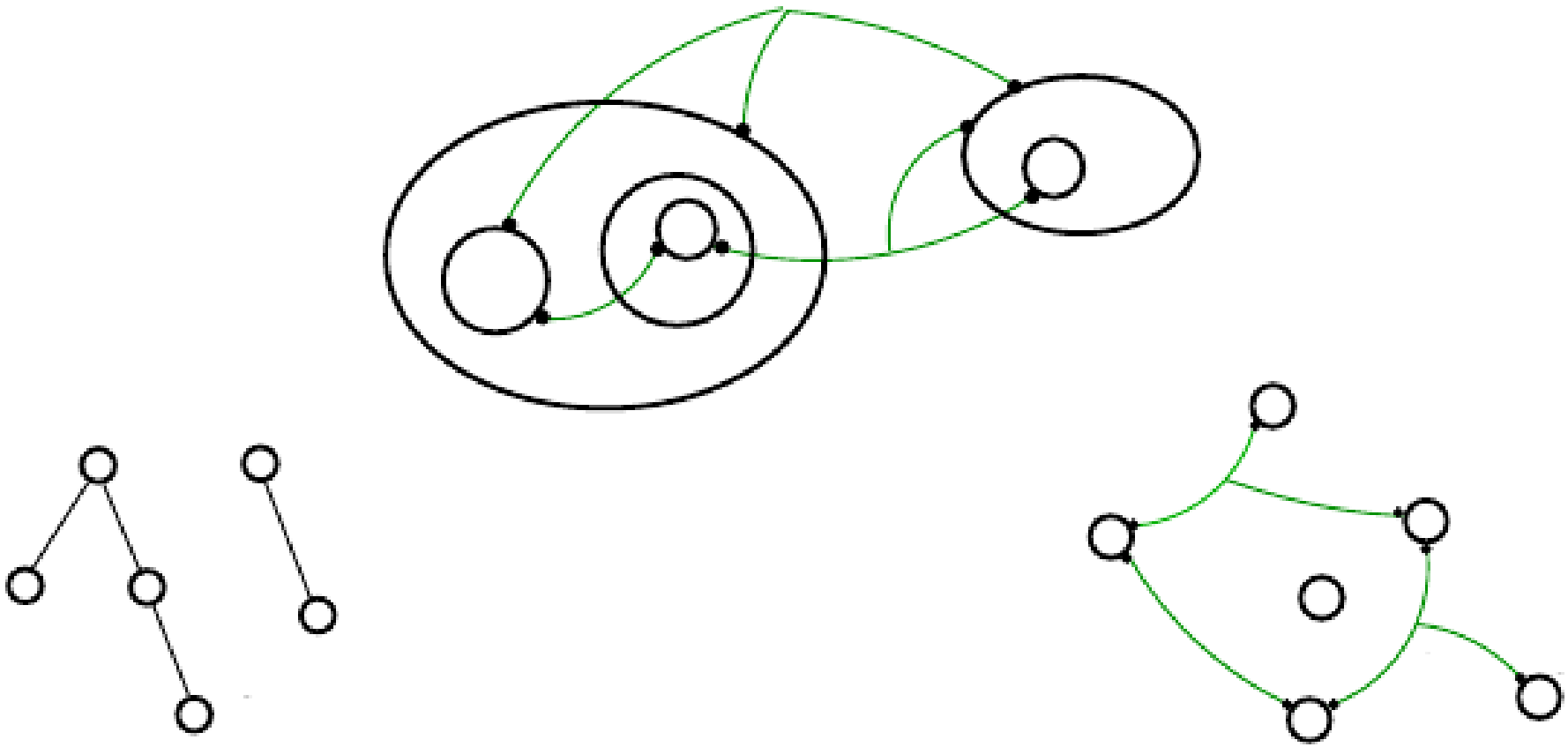
Then we expect libraries not ‘within’ the silo but ‘without’

Thought experiment: a language that ‘has’ *no* library?

Ditto: debuggers, package managers, . . .

(the focuses of much of my technical work)

2. Value *linking* over containment



Hierarchy versus heterarchy

Preference for hierarchical abstractions is widespread.

It's appealing {mathematically, bureaucratically, . . . }

It's also often a limitation in expressiveness

. . . but not only that; also in how it 'goes bad'.

What lies within. . . should be minimised

If we force hierarchy ill-fittingly, we get *copies*!

Example: filesystems without symlinks.

Example: libraries ‘for’ language X!

Example: replicated state $\longrightarrow$ synchronization problems

Thought experiment

Imagine an environment with no way to model ‘has-a’

. . . only ‘refers-to-a’

Consequence: *names* for everything

Designs around linking within a shared [name]space:

- convivial; promotes parsimony

Designs around containment within an ownership domain

- gambles on dominance; promotes replication

3. Porous, not portable

Portable *specifications* are really useful. (Think: JVM.)

. . . portable implementations? Problematic. (Think: JVM.)

An instance of 'too many standards? define one more!'

A design heuristic doggedly pursued.

On your desktop right now: how many layers of window management?

Being porous means: work with what you find

Don't plan to build a 'sealed' abstraction

. . . around a 'lower-level' 'common denom.' interface

Don't say 'lower-level'; it's more 'found' or 'varying'.

. . . workable if narrow enough (printing example)

. . . workable if tools can recover *abstract similarities*

(my PhD topic from many years ago; still mostly cold)

(a not-so-shining 'real' example: GNU autoconf)

‘Is the dark side stronger?’

‘Is the dark side stronger?’

Maybe non-convivial tools will always be more powerful?

Not just technically, but socially?

How can the convivial ‘light side’ prevail?

- it’s slower, more difficult, less seductive. . .

‘Is the dark side stronger?’

Tragedy: societies become dependent on over-powerful tools.

- powerful transportation
- medical dependency amid an unhealthy environment
- $\rightarrow$ Illich’s notion of ‘radical monopoly’

Solution? Perhaps: be ‘movement’, ‘culture’, ‘tribe’.

- the evolutionary means of inducing
- . . . sacrifice to the greater good!
- via enforced norms, taboos
- (danger? another form of ‘fighting fire with fire’)

Don’t just be free; be tractable

The ‘free software’ movement has some overlap

- GPL is convivial!

But is very vulnerable to non-convivial outcomes

- ‘free’ projects need not be tractable to individuals

Differentiators needed:

- convivial → slow rate of change
- convivial → learnable
- convivial → user-respecting (Linux)

The end

All disciplines have their dogmas.

Programming needs new ones!

Maybe the preceding heuristics can help. . . ?