The Mythical Matched Modules

Or: overcoming the tyranny of inflexible software construction.

Stephen Kell

Stephen.Kell@cl.cam.ac.uk

Computer Laboratory

UNIVERSITY OF CAMBRIDGE

There’s something about software. . .

Software is expensive and *inflexible*.

Tools assume:

- ground-up
- perfect fit
- don’t change
- never replaced

Reality: none of the above!

What it means in practice

Tasks dealing with change are notoriously hard work.

- *porting*
- *wrapping*

Language innovations are notoriously slow to be adopted

- each language is its own silo

Usual solution is “don’t do that!”

- rewrite from scratch
- single-language programming

The idea in one slide

Parnas pioneered *information hiding*.

- interface changes are painful, so . . .
- keep interfaces *minimal*, to avoid change

But interfaces *do* change. What can be done?

- radically separate integration: *interface hiding*
  - Error: “import” statement deprecated
- specialized tool support for integration
  - complementary languages (Cake is one example)

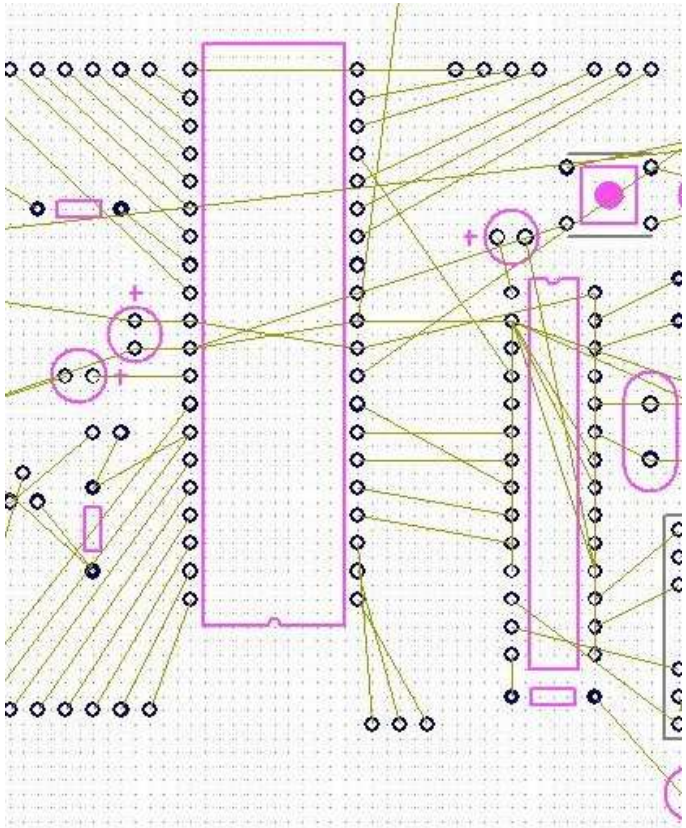
Dry stone software: complexity inheritance (1)

```
decoder = mpeg2_init ();
info = mpeg2_info (decoder);
do { state = mpeg2_parse (decoder);
     switch ( state ) {
      case STATE_BUFFER:
       size = fread ( buffer , /* ... */);
       mpeg2_buffer (decoder, buffer , /* ... */);
       break;
      case STATE_SLICE: case /* ... */:
       for (i = 0; i < seq->luma_height; i++)
        fwrite (info->display_fbuf-> /* ... */ );
       break;
    } while ( size );
mpeg2_close (decoder);
```

Dry stone software: complexity inheritance (2)

```
avcodec_init (); av_register_all ();
codec = avcodec_find_decoder(c->codec_id);
ap->time_base= (AVRational){1, 25};
ap->pix_fmt = PIX_FMT_NONE;
err = av_find_stream_info (ic);
for(i = 0; i < ic->nb_streams; i++) {
 AVCodecContext *enc = ic->streams[i]->codec;
 if (enc->codec_type == CODEC_TYPE_VIDEO)
  video_index = i; }
c = ic->streams[video_index]->codec;
for (;;) { while (pkt->size > 0) {
  picture = avcodec_alloc_frame ();
  len = avcodec_decode_video2(c, picture, &got_picture, pkt);
  if (len >= 0 && got_picture) {
   for(i=0; i < c->height; i++) /* for each row */
```

Separate integration



Hardware (and other domains)

- keep components simple
- . . . and composable
- . . . and cheap

By separating integration. . .

- *physically* (modules)
- *notationally* (languages)

Cake: an integration domain

Just *one* example of an integration domain: Cake.

- declarative language of *interface correspondences*
- complements general-purpose programming languages

```
derive /* ... */ WholeProgram = link[someClient.o, my_library.so ]
{
  // ...
  someClient.o ↔ my_library.so {
    manipulate_foo(i, d, c_h, data)
      → make_foobaz(i, d, c_h, data, null, {});
    // more correspondences ...
  }
  // more pairwise blocks...
};
```

Conclusions

Interface hiding is a radical “next step” in modularity.

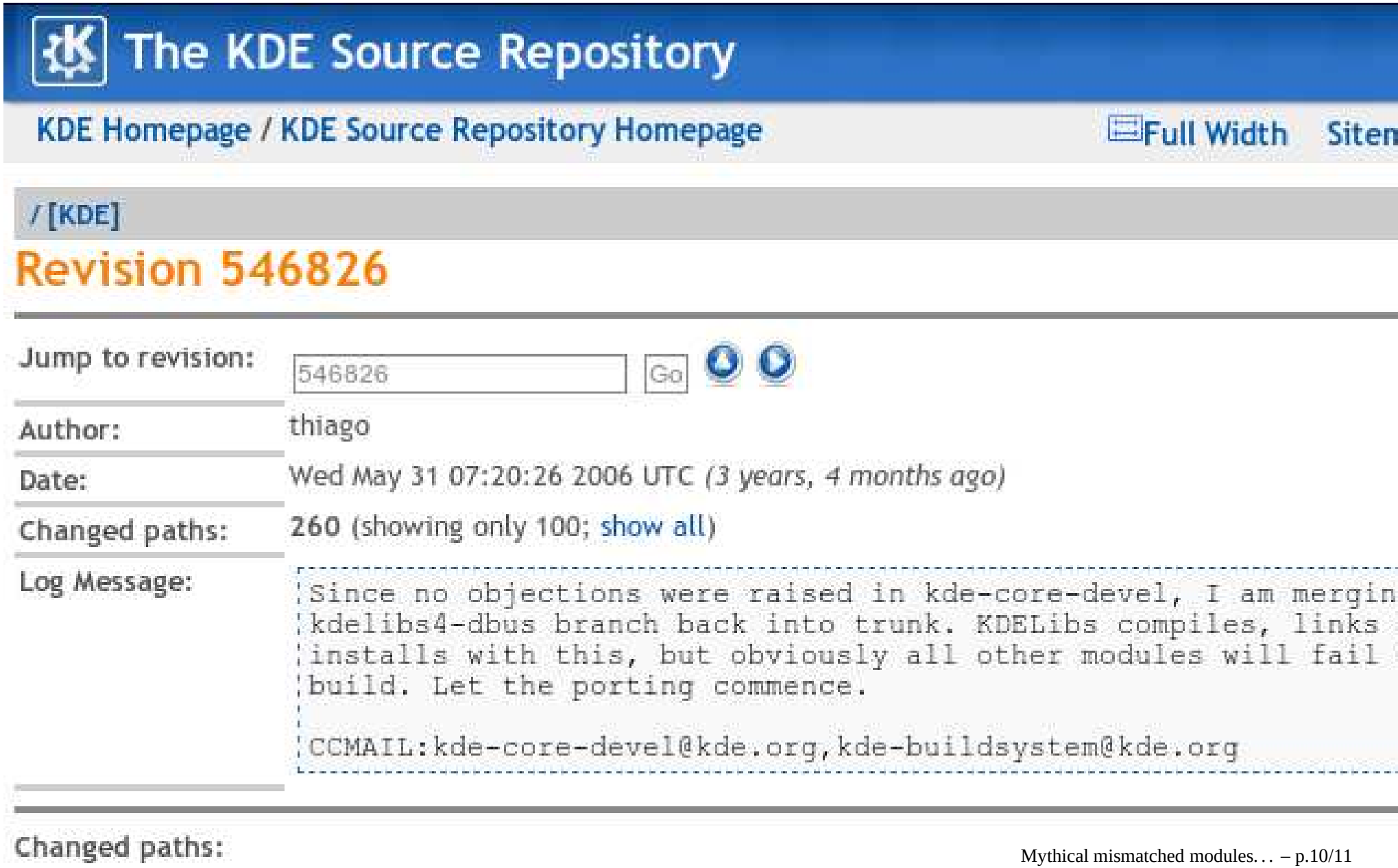
- reduce component complexity
- minimises coupling
- retain composability with *many libraries*

Integration domains abstract integration

- better notational abstractions
- absorb change; absorb language mismatch; …
- *incremental adoption is feasible*

Thanks for your attention. Any questions?

Exhibit K



The KDE Source Repository

KDE Homepage / KDE Source Repository Homepage Full Width Siten

/ [KDE]

Revision 546826

Jump to revision: 546826 Go

Author: thiago

Date: Wed May 31 07:20:26 2006 UTC *(3 years, 4 months ago)*

Changed paths: **260** (showing only 100; show all)

Log Message:

```
Since no objections were raised in kde-core-devel, I am mergin
kdelibs4-dbus branch back into trunk. KDELibs compiles, links
installs with this, but obviously all other modules will fail
build. Let the porting commence.

CCMAIL:kde-core-devel@kde.org,kde-buildsystem@kde.org
```

Changed paths:

Table X

	mpeg2	ffmpeg	db	sqlite
API function count	24	151	208	214
mean signature size	3.3	4.3	2.8	4.3
API structures count	10	26	25	12
mean structure size	5.6	19	17	8.0
slice function count	5	14	5	12
mean signature size	2.8	3.1	6	4
slice structures count	3	6	2	4
client size (LoC)	71	93	51	75
client API calls	5	14	5	13
helper calls	2	1	6	3
mean call size	3.4	3.2	4.9	3.9