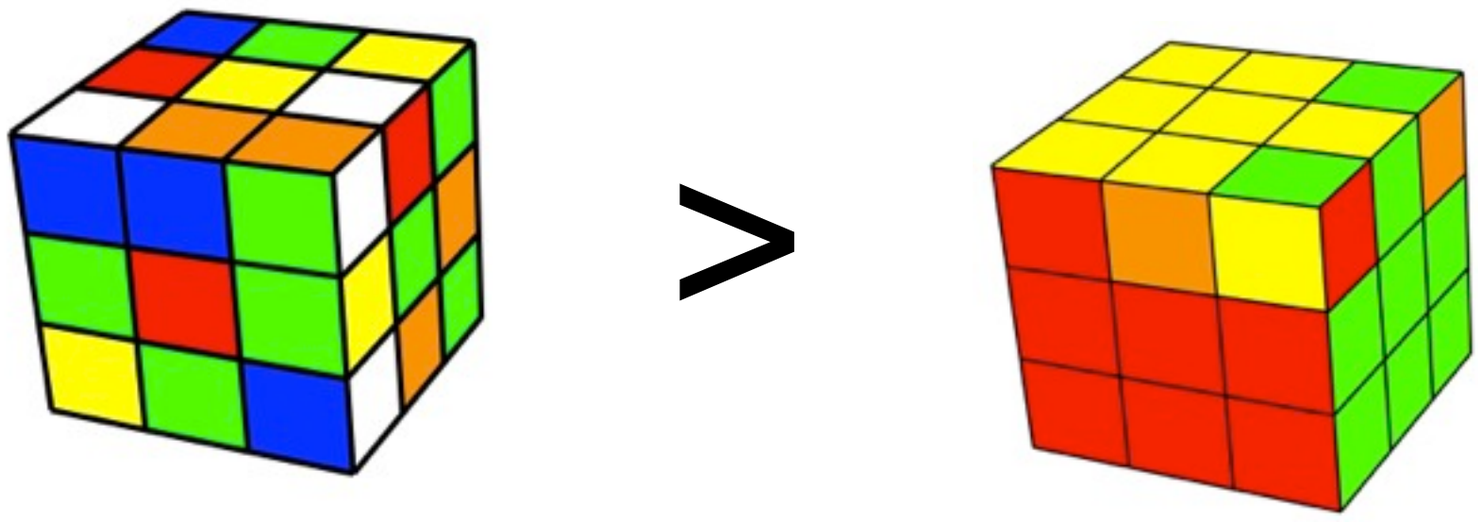
# Code Entropy and Physics of Software

## Jon Jagger & Olve Maudal

While working on the Code Archeology talk, presented at ACCU 2010, we realized that the "Physics of Software", and in particular “Code Entropy,” is a topic worth presenting by itself.

By observing how a codebase changes over time you start to notice there are strong and weak forces that shapes the code, and code can be characterized into having a stable or unstable equilibrium. Maintaining and developing a codebase over time is about pushing it in the right direction, always towards higher grounds.

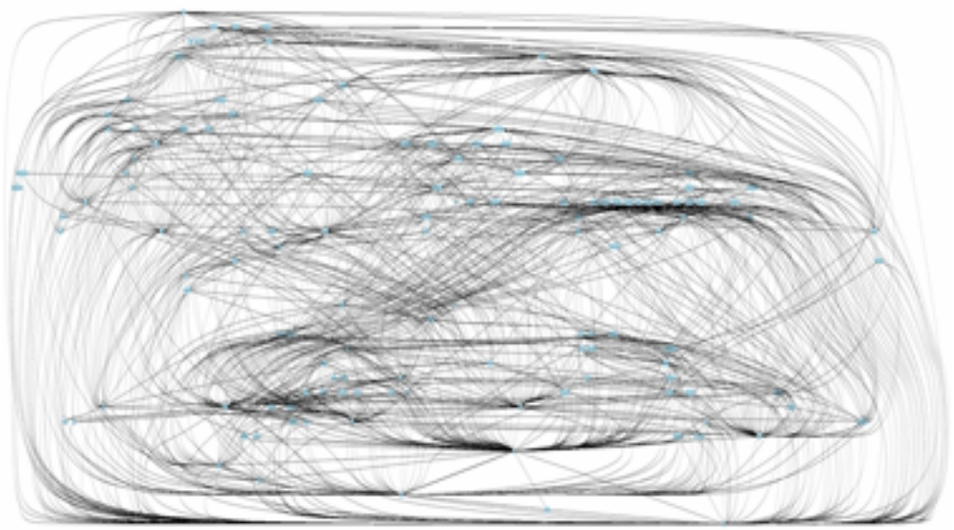
We have studied a large and successful codebase written in C and C++ to find good observations of stable and unstable code, and tried to identify some of the governing forces. We will focus on the small stuff, snippets of code written in C.



A 90 minute session at the ACCU conference
Oxford, April 13-16 2011
(abridged version)

- Introduction
- Results of survey

# Codebase Under Discussion



- C and C++
- a few million lines of code
- processor agnostic (TI, Phillips, PPC, ARM, Intel, and more...)
- currently developed and maintained by ~200 developers
- typically 50-200 commits per day
- very visible traces back to '80s and '90s

# Examples of past products

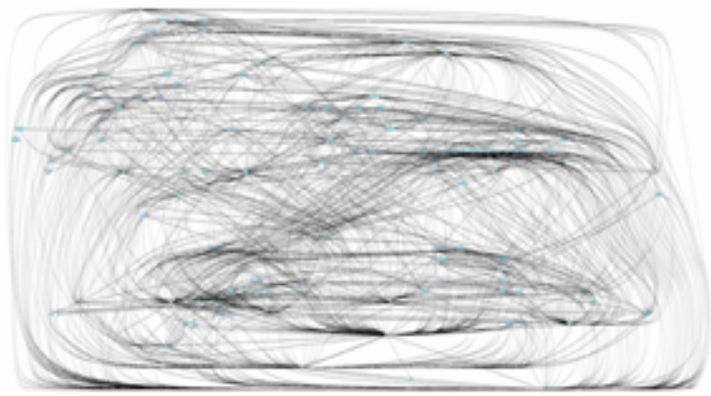
1987

1992

1997

2000

2003



2004

2006

2006

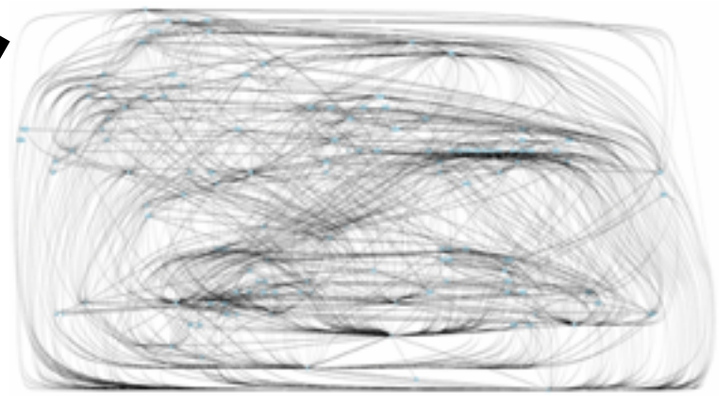
2005

# Examples of current products

2010

2008


2012



2009


2012

2008


2011

2009

This can only be achieved through:

This can only be achieved through:

- visible and strong architecture

This can only be achieved through:

- visible and strong architecture
- well defined development processes

This can only be achieved through:

- visible and strong architecture
- well defined development processes
- proper documentation of codebase

This can only be achieved through:

- visible and strong architecture
- well defined development processes
- proper documentation of codebase
- corporate coding standards

This can only be achieved through:

- visible and strong architecture
- well defined development processes
- proper documentation of codebase
- corporate coding standards

... so we have been told ...

This can only be achieved through:

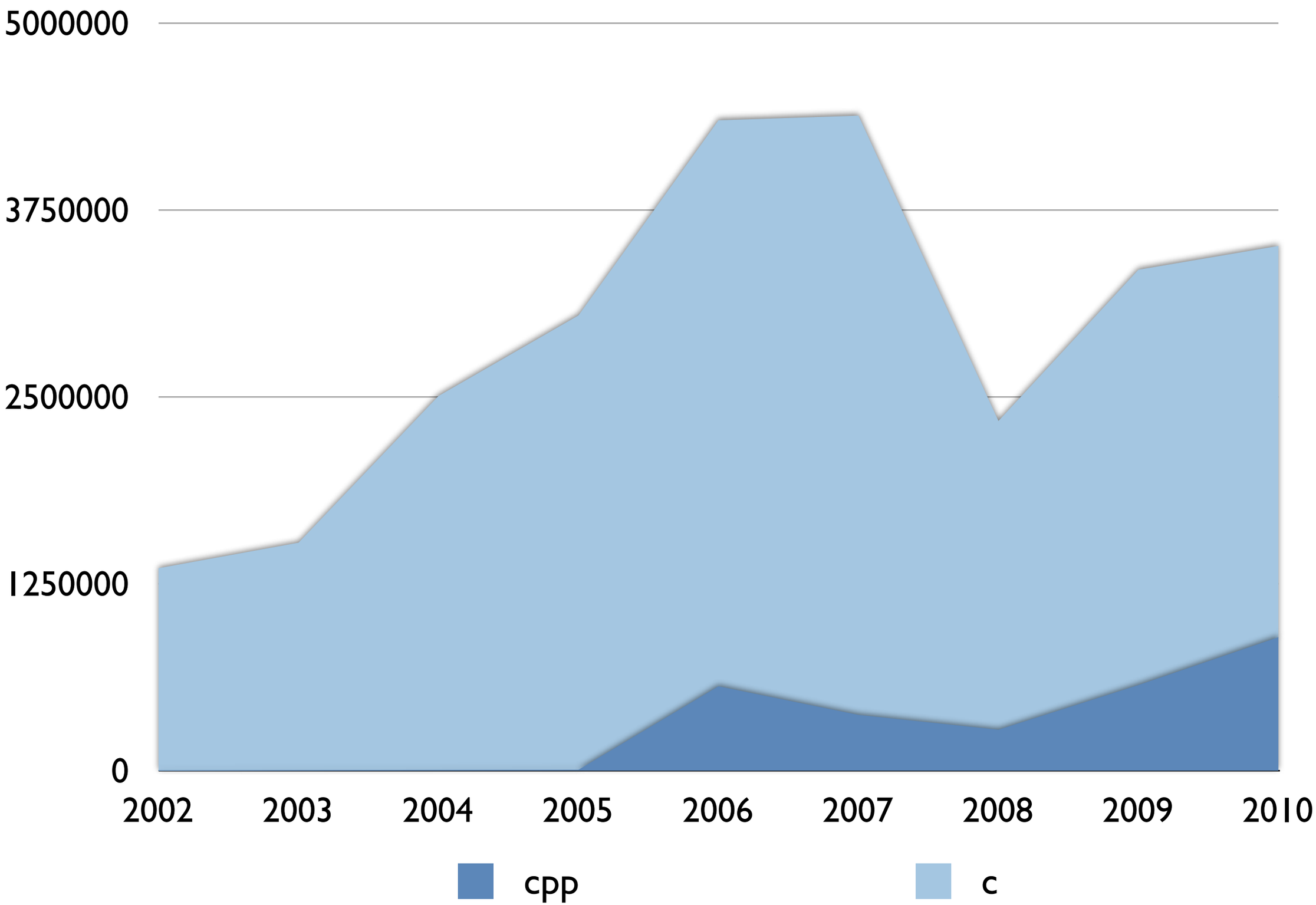
- visible and strong architecture
- well defined development processes
- proper documentation of codebase
- corporate coding standards

... so we have been told ...

There are only two kinds of codebases: the ones people complain about and the ones nobody cares about anymore...

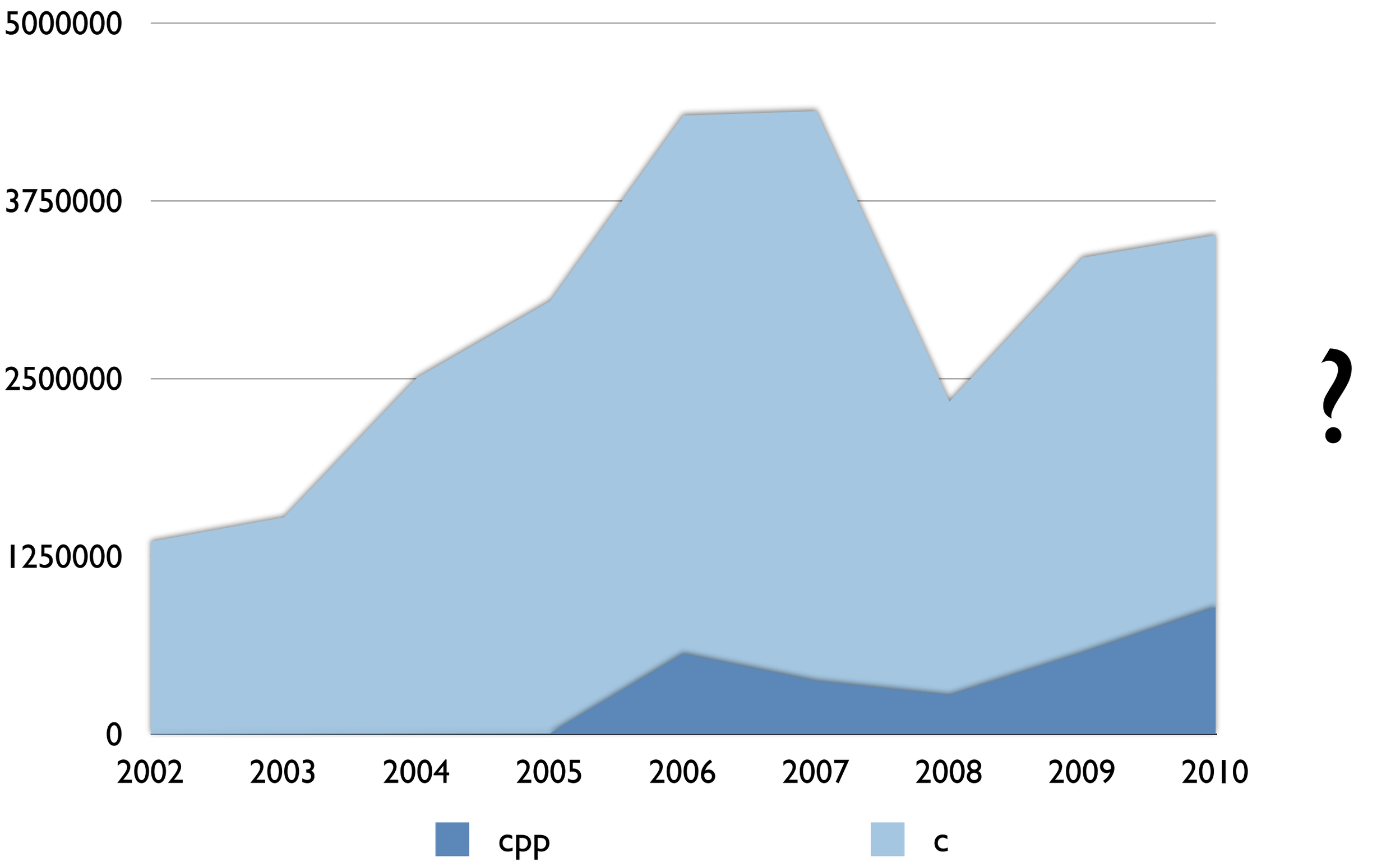
(inspired by a similar quote by Bjarne Stroustrup)

# Maintrunk 2002-2010



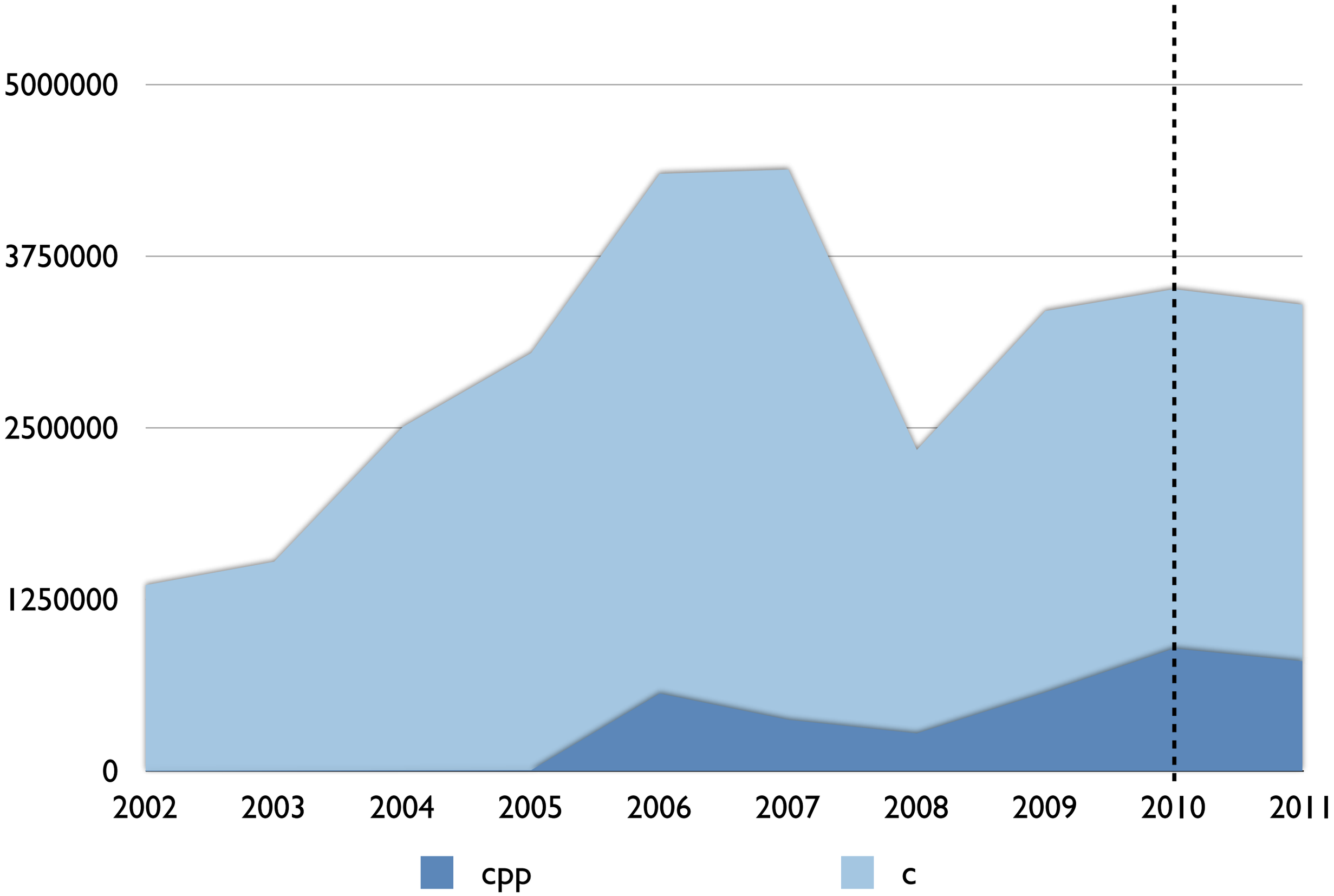

generated using David A. Wheeler's 'SLOCCount'

# Maintrunk 2002-2010




generated using David A. Wheeler's 'SLOCCount'

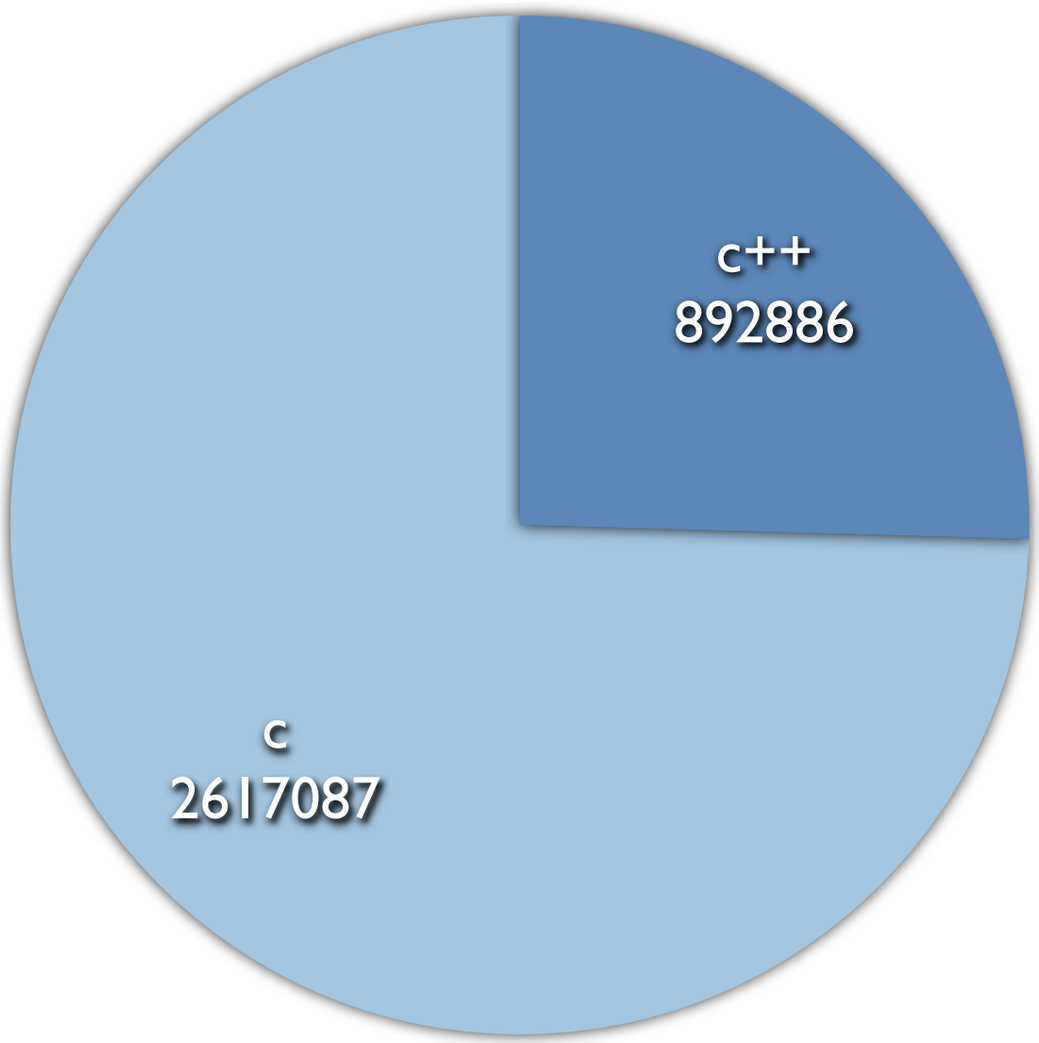
# Maintrunk 2002-2011



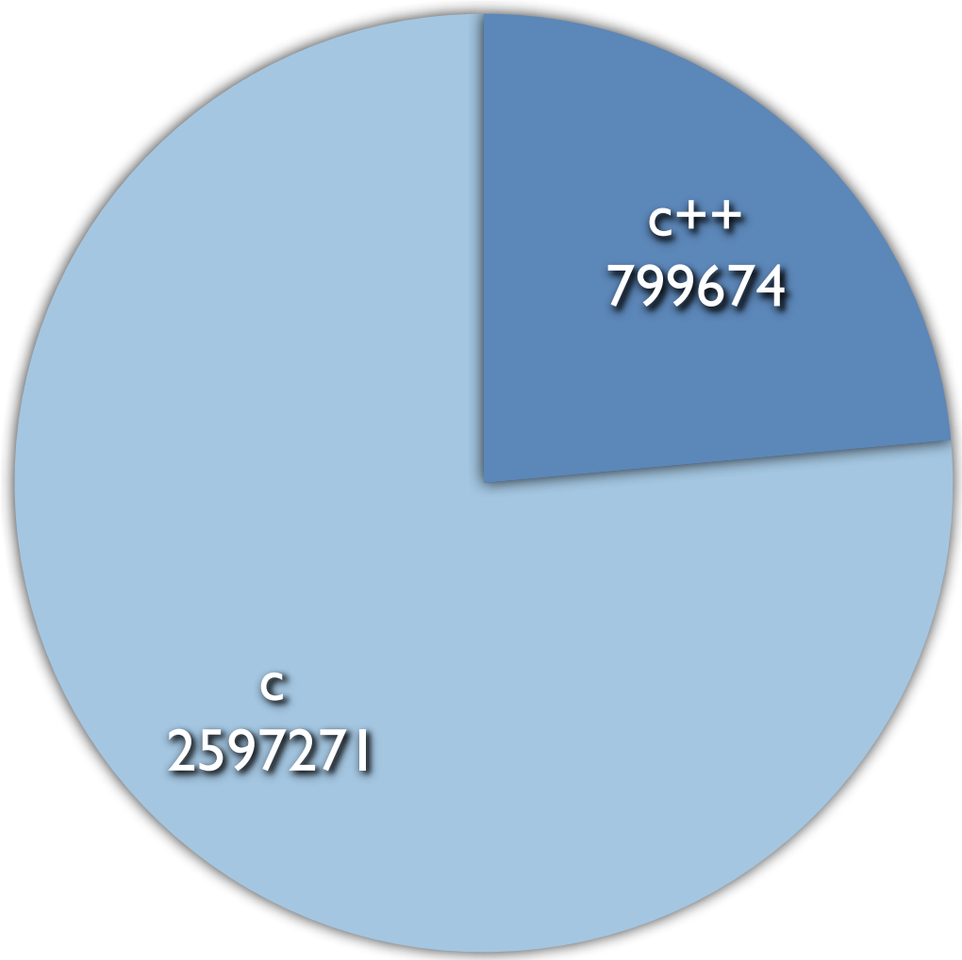

generated using David A. Wheeler's 'SLOCCount'

# Maintrunk 2010-2011

2010

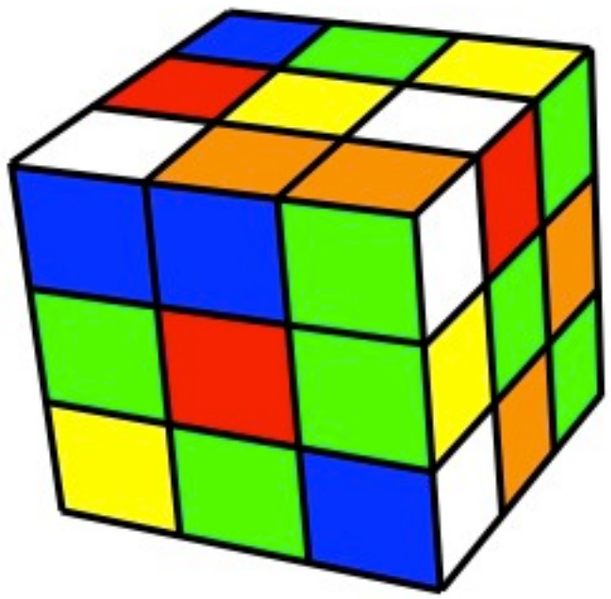



2011



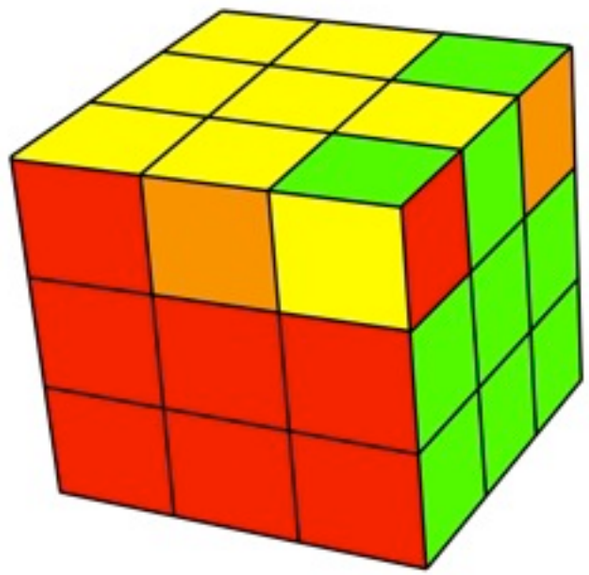

Commits between 2010-04-01 and 2011-04-01:
22660 commits (avg 61 per day)
150+ committers (60 devs with more than 100 commits)

# Entropy



high entropy

>



low entropy

Entropy is a measure of how organized or disorganized a system is

(wikipedia)

# Code Entropy

# Code Entropy

*Consider two semantically similar code snippets, A and B.*

# Code Entropy

*Consider two semantically similar code snippets, A and B.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A

# Code Entropy

*Consider two semantically similar code snippets, A and B.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A

```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

B

# Code Entropy

*Consider two semantically similar code snippets, A and B.*
*If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A

```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

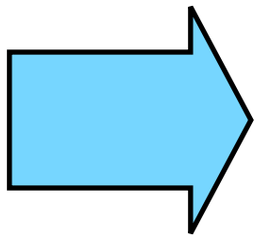
B

# Code Entropy

*Consider two semantically similar code snippets, A and B.*
*If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A



```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

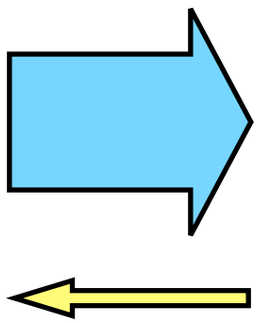
B

# Code Entropy

*Consider two semantically similar code snippets, A and B.*
*If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A



```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

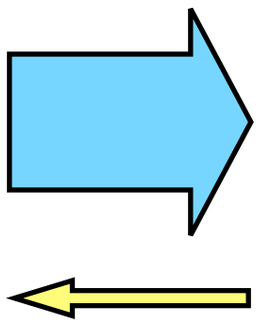
B

# Code Entropy

*Consider two semantically similar code snippets, A and B.*
*If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.*
*Hence, A has higher entropy than B.*

```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

A



```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

B

# Code Entropy

*Consider two semantically similar code snippets, A and B.*
*If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.*
*Hence, A has higher entropy than B.*

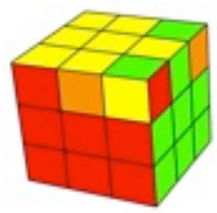
```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

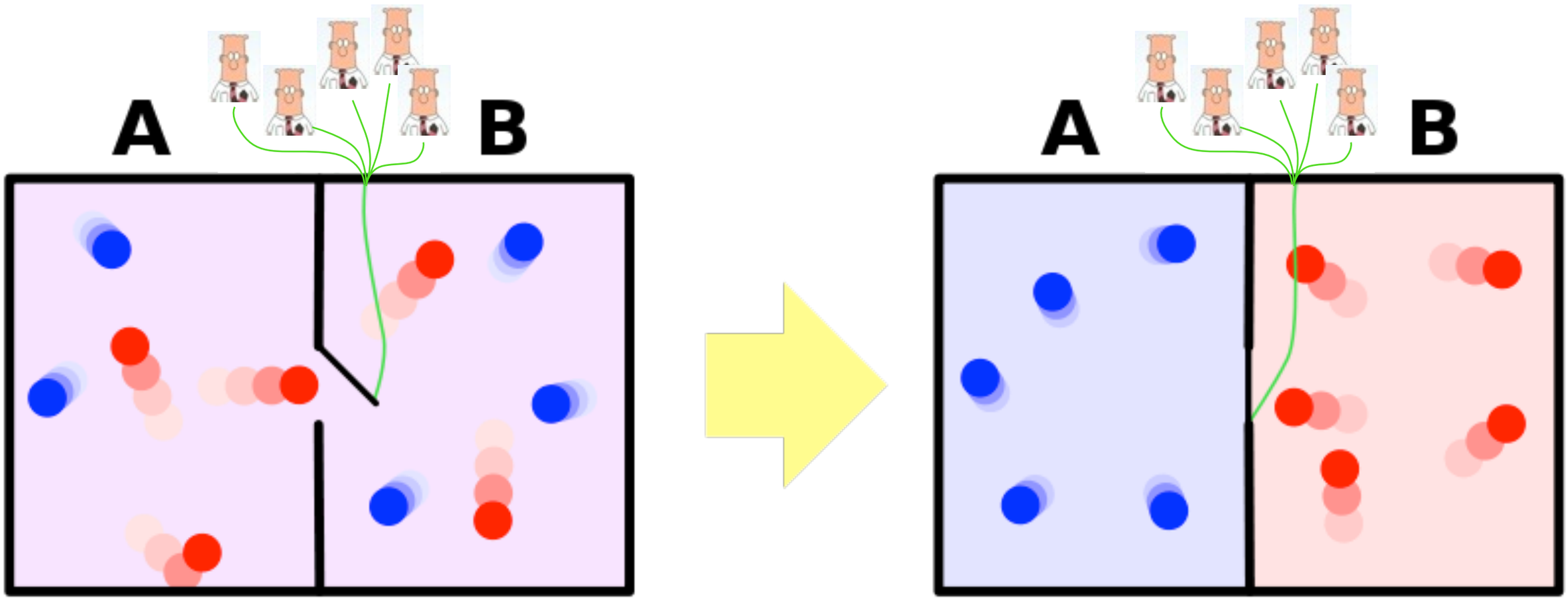
```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

A

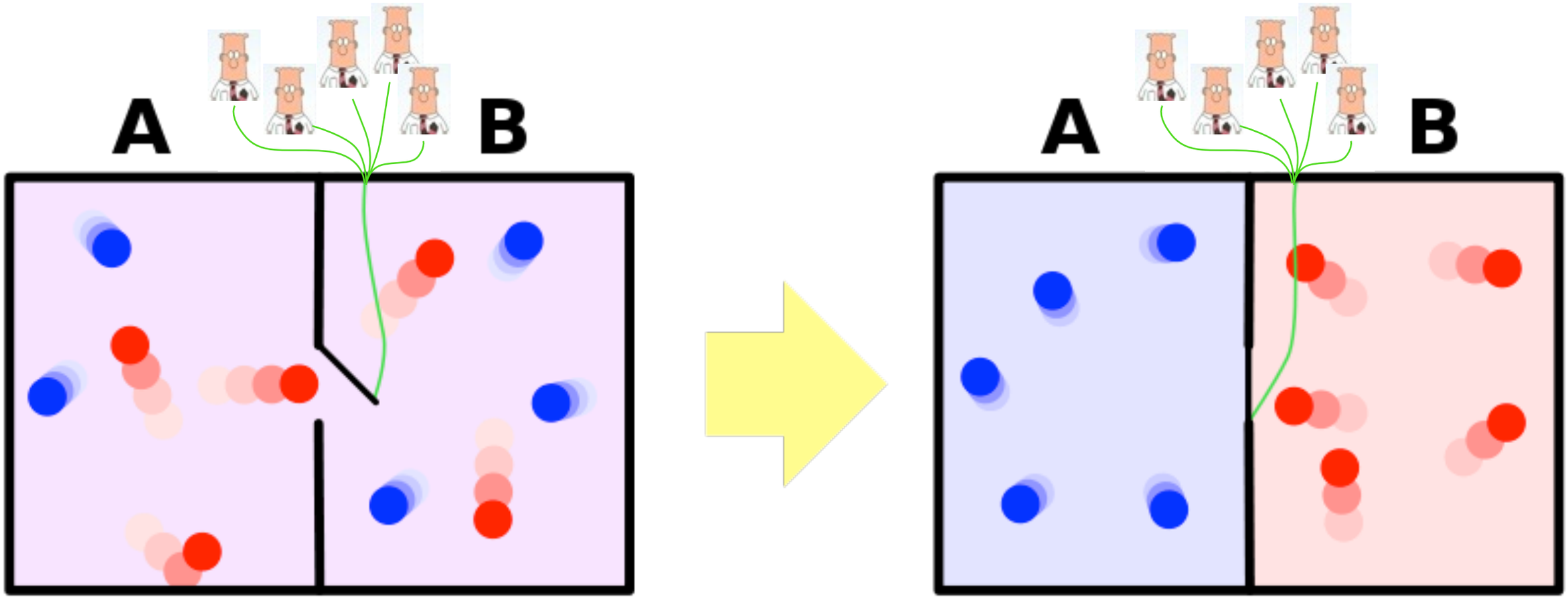
>

B



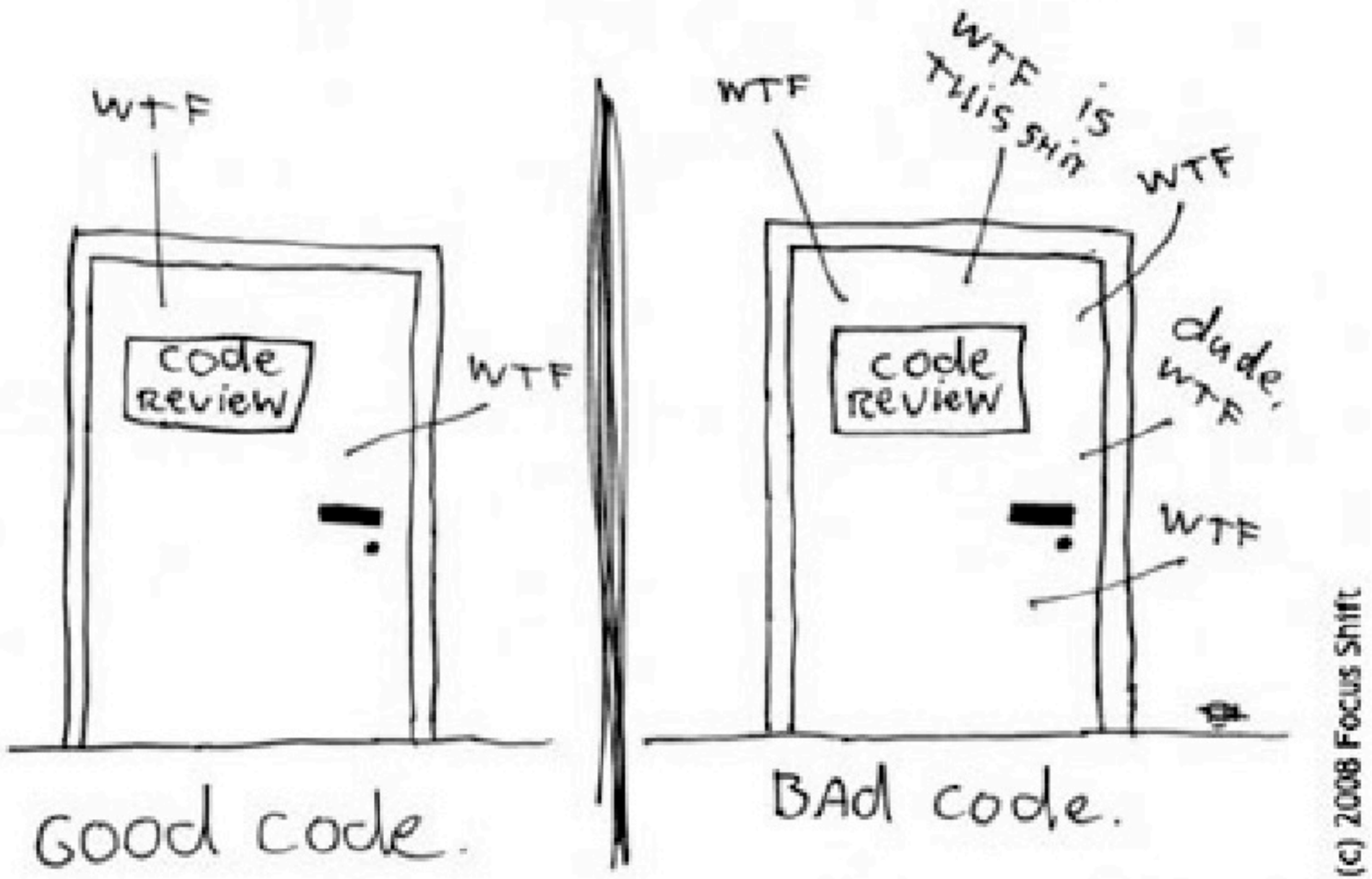

A
B
A
B

# Dilbert's demon



Codebases are changing over time

# The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE




(reproduced with kind permission of Thom Holwerda)

The health of a codebase is best understood by studying how it changes over time.

Is it deteriorating or improving?

The health of a codebase at a moment in time is less useful.

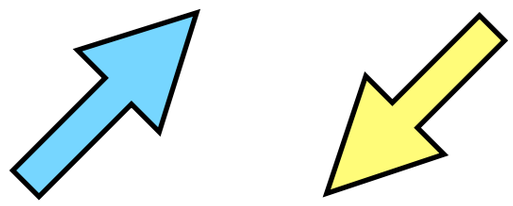
# Survey...

After the introduction, Jon and I showed a collection of code snippet selected from change sets in a real codebase. We asked the audience to vote on which code snippet they believe a group of experts would prefer. Ie, which code snippet is more stable and has lower entropy.

# Results of survey

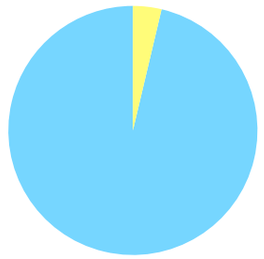
# #1 - negation

```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```

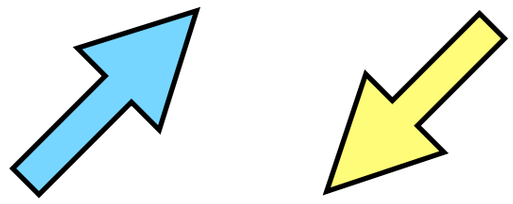


```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

# #1 - negation



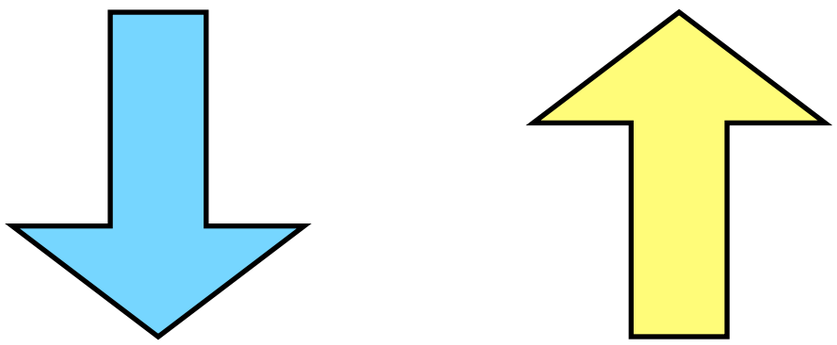
```
  ...
  if (is_open(socket))
    return true;
  else
    return false;
}
```



```
  ...
  if (!is_open(socket))
    return false;
  else
    return true;
}
```

# #2 - braces

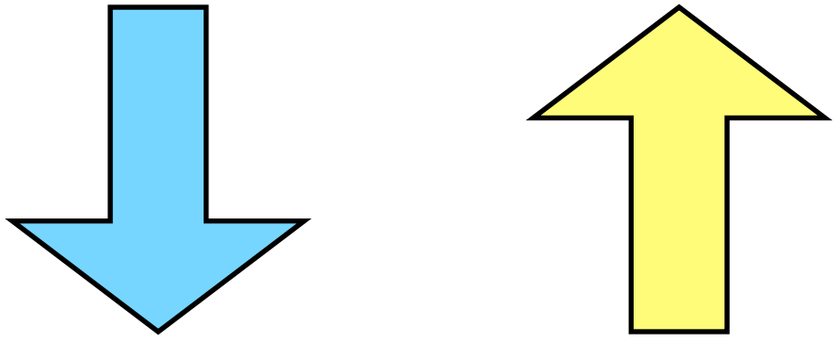
```
void eventlogputs(const char * string) {
    ...
}
```



```
void eventlogputs(const char * string)
{
    ...
}
```

# #2 - braces

```
void eventlogputs(const char * string) {
    ...
}
```



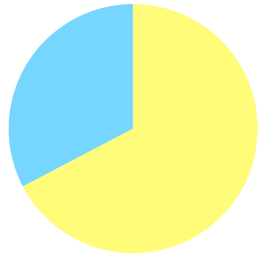
```
void eventlogputs(const char * string)
{
    ...
}
```

# #3 - horizontal spacing

```
NetAddr_initAsIPv6(&na.addr, &tests[0].addr, 80, 0);
ret = NetAddr_toString(&na.addr, buf, sizeof(buf), true);
ASSERT(ret && strcmp(buf, "[::1]:80") == 0);
```

```
NetAddr_initAsIPv6( &na.addr, &tests[0].addr, 80, 0 );
ret = NetAddr_toString( &na.addr, buf, sizeof(buf), true );
ASSERT( ret && strcmp( buf, "[::1]:80" ) == 0 );
```

# #3 - horizontal spacing



```
NetAddr_initAsIPv6(&na.addr, &tests[0].addr, 80, 0);
ret = NetAddr_toString(&na.addr, buf, sizeof(buf), true);
ASSERT(ret && strcmp(buf, "[::1]:80") == 0);
```

```
NetAddr_initAsIPv6( &na.addr, &tests[0].addr, 80, 0 );
ret = NetAddr_toString( &na.addr, buf, sizeof(buf), true );
ASSERT( ret && strcmp( buf, "[::1]:80" ) == 0 );
```

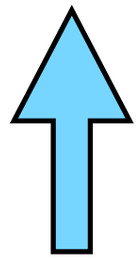
# #4 - use of temporaries

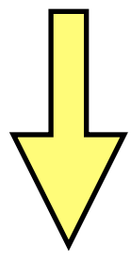
```
/* According to CSL API, size must be multiple of 4 (However,
   looking at CSL implementation, it handles non-multiple of
   4. Best to follow doc...) */
size = ((size + 3) / 4) * 4;
```

```
/* According to CSL API, size must be multiple of 4 (However,
   looking at CSL implementation, it handles non-multiple of
   4. Best to follow doc...) */
tmp = size & 0x3;
if (tmp)
   size += (4 - tmp);
```

# #4 - use of temporaries

```
/* According to CSL API, size must be multiple of 4 (However,
   looking at CSL implementation, it handles non-multiple of
   4. Best to follow doc...) */
size = ((size + 3) / 4) * 4;
```





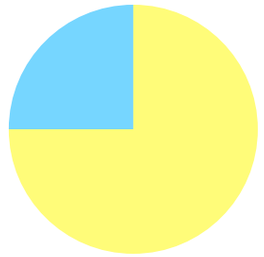
```
/* According to CSL API, size must be multiple of 4 (However,
   looking at CSL implementation, it handles non-multiple of
   4. Best to follow doc...) */
tmp = size & 0x3;
if (tmp)
   size += (4 - tmp);
```

# #5 - to embrace or not to embrace

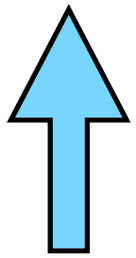
```
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164)
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
else
    strcpy(verify, " ");
```

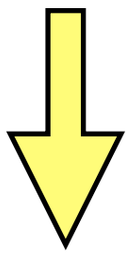
```
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164) {
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
} else {
    strcpy(verify, " ");
}
```

# #5 - to embrace or not to embrace



```
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164)
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
else
    strcpy(verify, " ");
```





```
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164) {
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
} else {
    strcpy(verify, " ");
}
```

# #6 - sizeof

```
SIP_bytes2hexstr(bytes, sizeof bytes, cnonce, len);
```

```
SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);
```

# #6 - sizeof

```
SIP_bytes2hexstr(bytes, sizeof bytes, cnonce, len);
```

```
SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);
```

# #7 - casting

```
assert(len < (int)sizeof(cmdname));
```

```
assert((size_t)len < sizeof(cmdname));
```

# #7 - casting

```
assert(len < (int)sizeof(cmdname));
```

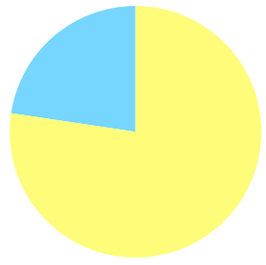
```
assert((size_t)len < sizeof(cmdname));
```

# #8 - typedef structs or not

```
pPriv->FBufsize = sizeof(struct AUD_STREAM_DATA_STRUCT);
```

```
pPriv->FBufsize = sizeof(AUD_STREAM_DATA_STRUCT);
```

# #8 - typedef structs or not

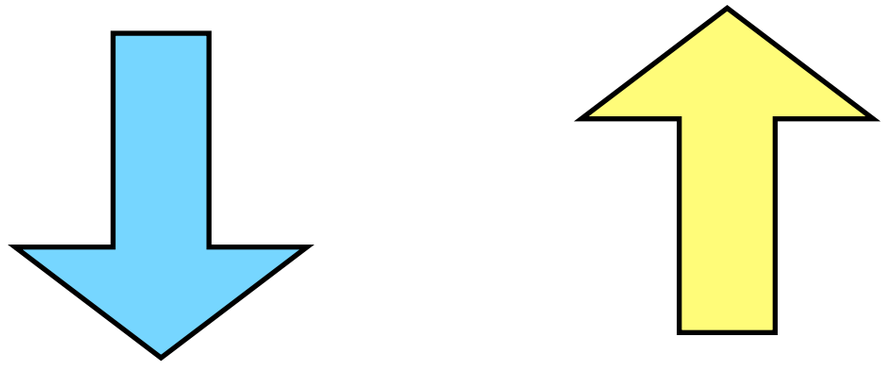


```
pPriv->FBufsize = sizeof(struct AUD_STREAM_DATA_STRUCT);
```

```
pPriv->FBufsize = sizeof(AUD_STREAM_DATA_STRUCT);
```

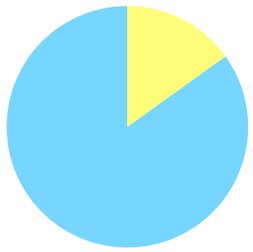
# #9 - parenthesis

```
if (priv->vcfsm.Spec != SYSTEM && priv->vcfsm.Spec != NO_SPECIE)
{
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
```

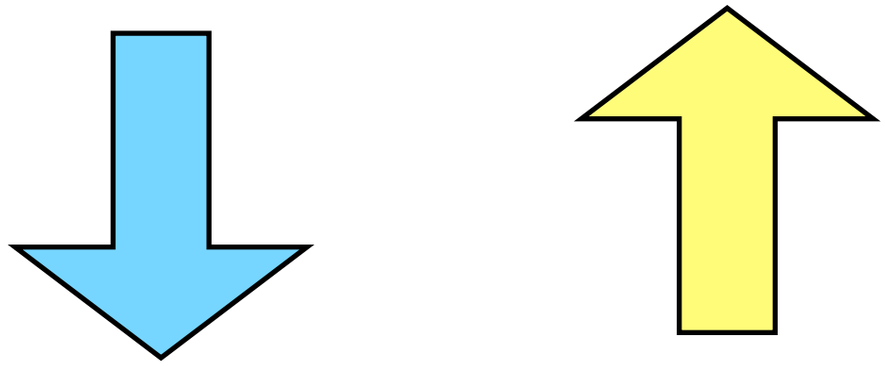


```
if ((priv->vcfsm.Spec != SYSTEM) && (priv->vcfsm.Spec != NO_SPECIE))
{
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
```

# #9 - parenthesis



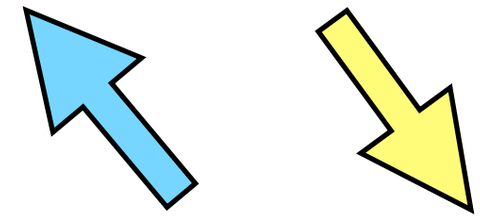
```
if (priv->vcfsm.Spec != SYSTEM && priv->vcfsm.Spec != NO_SPECIE)
{
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
```



```
if ((priv->vcfsm.Spec != SYSTEM) && (priv->vcfsm.Spec != NO_SPECIE))
{
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
```

# #10 - naming

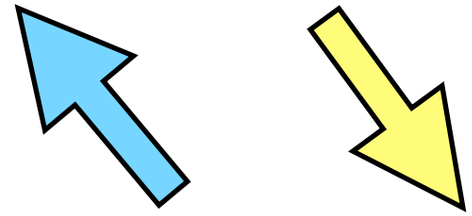
```
static void html_select_option(
    char * dest,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
   size_t destsize)
{
```



```
static void html_select_option(
    char * buffer,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t buffer_size)
{
```

# #10 - naming

```
static void html_select_option(
    char * dest,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
   size_t destsize)
{
```



```
static void html_select_option(
    char * buffer,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t buffer_size)
{
```

# #11 - strcpy vs sprintf

```
strcpy(tmp, gctx->digest.algorithm);
```

```
sprintf(tmp, "%s", gctx->digest.algorithm);
```

# #11 - strcpy vs sprintf

```
strcpy(tmp, gctx->digest.algorithm);
```

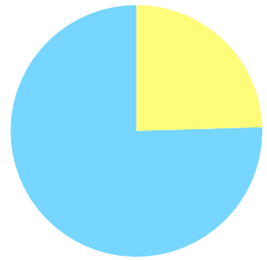
```
sprintf(tmp, "%s", gctx->digest.algorithm);
```

# #12 - needle and haystack

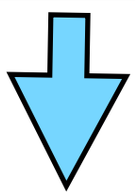
```
static bool isin(const char * needle,
                 char * const * haystack)
{
    while (*haystack)
        if (strcmp(needle, *haystack++) == 0)
            return true;

    return false;
}
```

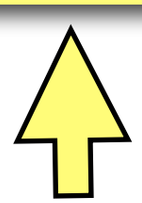
```
static bool isin(const char * needle,
                 char * const * haystack)
{
    while (*haystack) {
        if (strcmp(needle, *haystack) == 0) {
            return true;
        }
        ++haystack;
    }

    return false;
}
```

# #12 - needle and haystack



```
static bool isin(const char * needle,
                 char * const * haystack)
{
    while (*haystack)
        if (strcmp(needle, *haystack++) == 0)
            return true;

    return false;
}
```

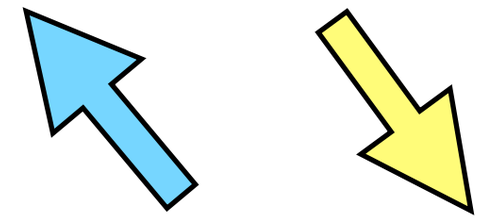




```
static bool isin(const char * needle,
                 char * const * haystack)
{
    while (*haystack) {
        if (strcmp(needle, *haystack) == 0) {
            return true;
        }
        ++haystack;
    }

    return false;
}
```

# #13 - small change

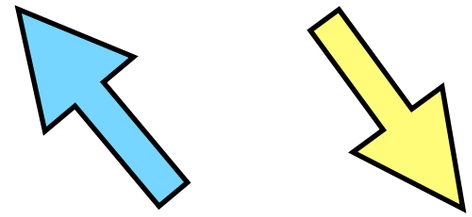
```
static void cleanup(char ** strings)
{
    char ** r = strings;

    while (*r) {
        free(*r);
        ++r;
    }

    free(strings);
}
```



```
static void cleanup(char ** strings)
{
    char ** r = strings;

    while (*r)
        free(*r++);

    free(strings);
}
```

# #13 - small change

```
static void cleanup(char ** strings)
{
    char ** r = strings;

    while (*r) {
        free(*r);
        ++r;
    }

    free(strings);
}
```



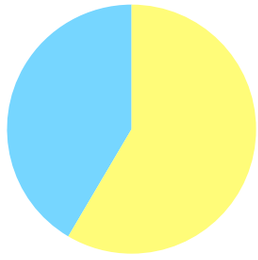
```
static void cleanup(char ** strings)
{
    char ** r = strings;

    while (*r)
        free(*r++);

    free(strings);
}
```

# #14 - variables

```
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
    parg_t *pa;
    FSMADDR fsm;
    int gate_id = GATE_UNDEFINED;
```

```
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
    parg_t *pa;
    FSMADDR fsm;
    int gate_id = GATE_UNDEFINED;
```

# #14 - variables



```
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
    parg_t *pa;
    FSMADDR fsm;
    int gate_id = GATE_UNDEFINED;
```

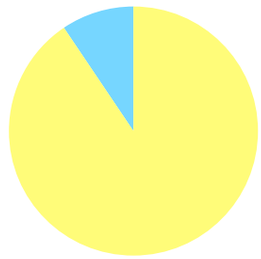
```
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
    parg_t *pa;
    FSMADDR fsm;
    int gate_id = GATE_UNDEFINED;
```

# #15 - fall through

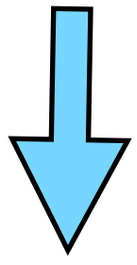
```
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
    // intentional fall through
case H261:
    if (!ttvenc->add_payload_headers) {
        skip += 4;
```

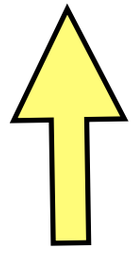
```
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
case H261:
    if (!ttvenc->add_payload_headers) {
        skip += 4;
```

# #15 - fall through



```
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
    // intentional fall through
case H261:
    if (!ttvenc->add_payload_headers) {
        skip += 4;
```





```
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
case H261:
    if (!ttvenc->add_payload_headers) {
        skip += 4;
```

# #16 - sizeof

```
globData->Time = malloc(sizeof *globData->Time);
```

```
globData->Time = malloc(sizeof(TIME_STR));
```

# #16 - sizeof

```
globData->Time = malloc(sizeof *globData->Time);
```

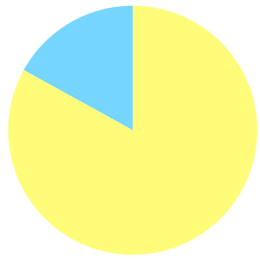
```
globData->Time = malloc(sizeof(TIME_STR));
```

# #17 - typedef

```
typedef struct {
    FSMADDR  portHandler;  /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
} CAM_Init_Req_Struct;
```

```
struct CAM_Init_Req_Struct {
    FSMADDR  portHandler;  /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
};
```

# #17 - typedef



```
typedef struct {
    FSMADDR  portHandler;  /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
} CAM_Init_Req_Struct;
```

```
struct CAM_Init_Req_Struct {
    FSMADDR  portHandler;  /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
};
```

# #18 - parenthesis

```
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return PARAMETER_ERROR;
}
```

```
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return(PARAMETER_ERROR);
}
```

# #18 - parenthesis

```
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return PARAMETER_ERROR;
}
```

```
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return(PARAMETER_ERROR);
}
```

# #19 - size_t and naming

```
char* SIP_bytes2hexstr(const unsigned char* input, int len, char* output, int maxlen);
```

```
char* SIP_bytes2hexstr(const unsigned char* src, size_t srclen, char* dest, size_t destsize);
```

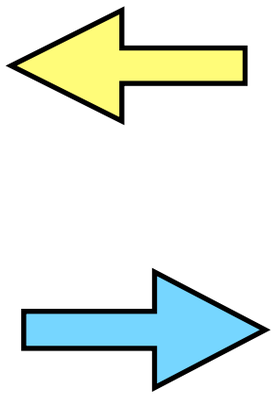
# #19 - size_t and naming

```
char* SIP_bytes2hexstr(const unsigned char* input, int len, char* output, int maxlen);
```

```
char* SIP_bytes2hexstr(const unsigned char* src, size_t srclen, char* dest, size_t destsize);
```

# #20 - include

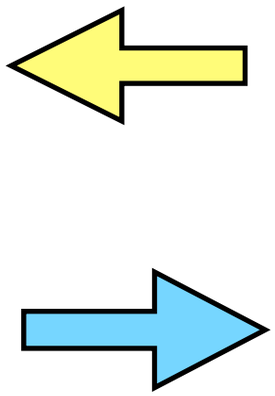
```
#include "ccf.h"

#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
```



```
#include "ccf.h"

#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <malloc.h>
#include <sys/ioctl.h>
```

# #20 - include

```
#include "ccf.h"

#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
```





```
#include "ccf.h"

#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <malloc.h>
#include <sys/ioctl.h>
```

# Appendix

# Significant events (recently)

Major directory reorganization (2008)

genbuild2 (2005)

Warning free (since 2007)

Proper C and C++ training (since 2007)

Automatic Continuous Integration (2005)

C++ introduced (~2006)

Unit Tests (2005)

Module Tests (2007)

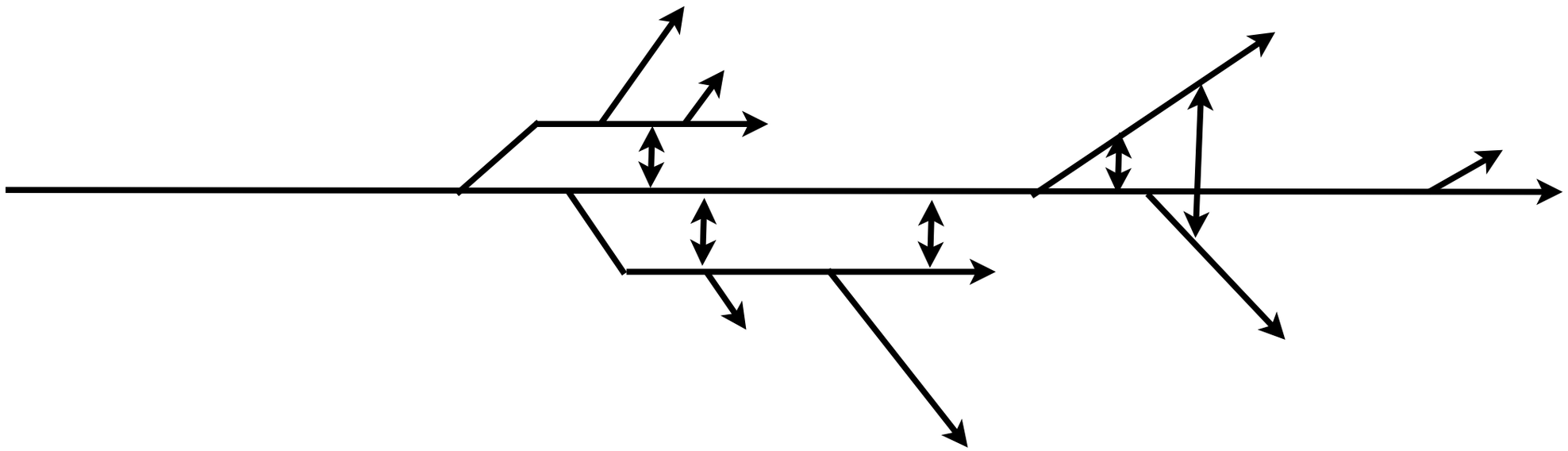
RCS to CVS (2002)

Stopped using code generating tools (2007)

CVS to SVN (2006)

Automatic system testing (since ~2007)

# Maintrunk and branching strategy



# Key observations from the Lysaker codebase 2010-2011

- readability
- compilation time
- dependency analysis / control / management
- less use of inline functions
- dependency injection
- more use of const, size_t and assert
- more use of forward declarations
- typedef struct deprecated
- moving away from corporate libraries
- using new language featuers, eg C99
- focus on standard C
- prefer C over C++, only use C++ where necessary
- proper casting, eg, `len < (int)sizeof x` **vs** `(size_t)len < sizeof x`
- macros gradually replaced, less use of preprocessor
- peer reviews / pair programming / patches mailing list
- collective ownership
- towards idioms

# and some more observations

- change functions to return void callers du not care about the return value
- `xx_assert()` replaced with `assert()`
- reducing scope of variables
- abbreviations replaced with more descriptive names
- explanation variables
- initialization of variables
- `for (int i=0; i<42; i++)` vs `for (int i=0; i!=42; ++i)`
- reduce use of fix ints (`uint32_t` -> `int`)
- order of include files
- early returns, less nesting
- intention revealing typenames (eg, `WORD` -> `bool`)
- less use of `NULL`
- log messages tend to be removed

# and, even more observations

- dehungarization
- decamelization
- aligned braces for functions, disaligned for if/for/while
- {} removed from single line blocks
- removing parenthesis
- long lines are broken into 80 character lines
- tabs are replaced with spaces
- `char * s` seems to be more stable than K&R and BS style
- In C, post-increment seems to be more stable than pre-increment (`i++` vs `++i`)
- focus on “robust” layout
- increased horizontal and vertical spacing
- indentation 4 spaces
- space around operators
- block comments are removed
- removing comments by improving code