A Catalog of Refactorings to Discipline Preprocessor-Based Annotations

Márcio Ribeiro
http://www.ic.ufal.br/marcio
marcio@ic.ufal.br
@marciomribeiro

```
this.dragon = Resources.getImage(Resources.IMG_DRAGON);
```

64kb, without flip **4Mb, without flip** **100kb, with flip**

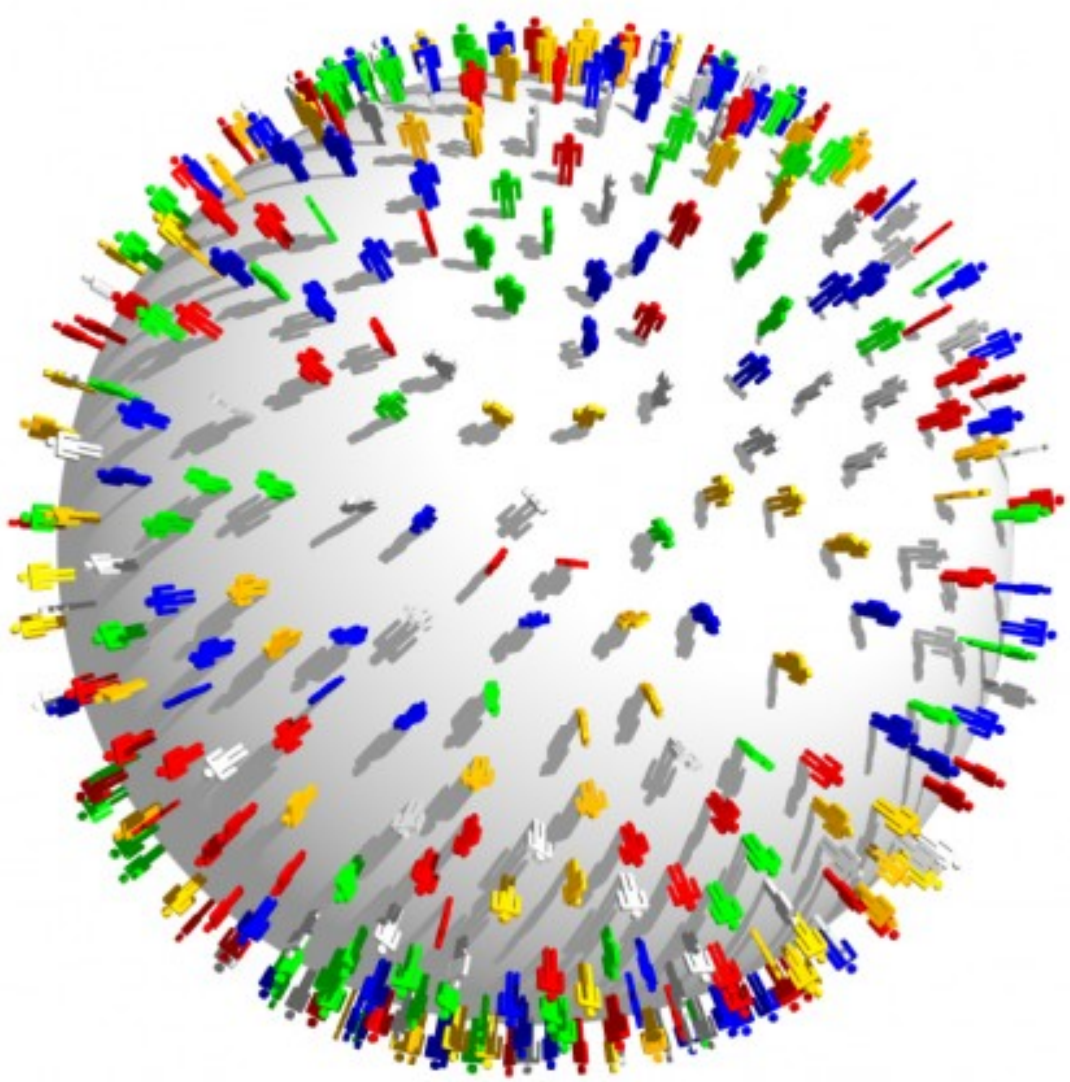
```
private int[] xClouds = new int[]{20, 80, 40, 60};
```

```
private int[] xClouds = new int[]{40, 140, 80, 110};
```

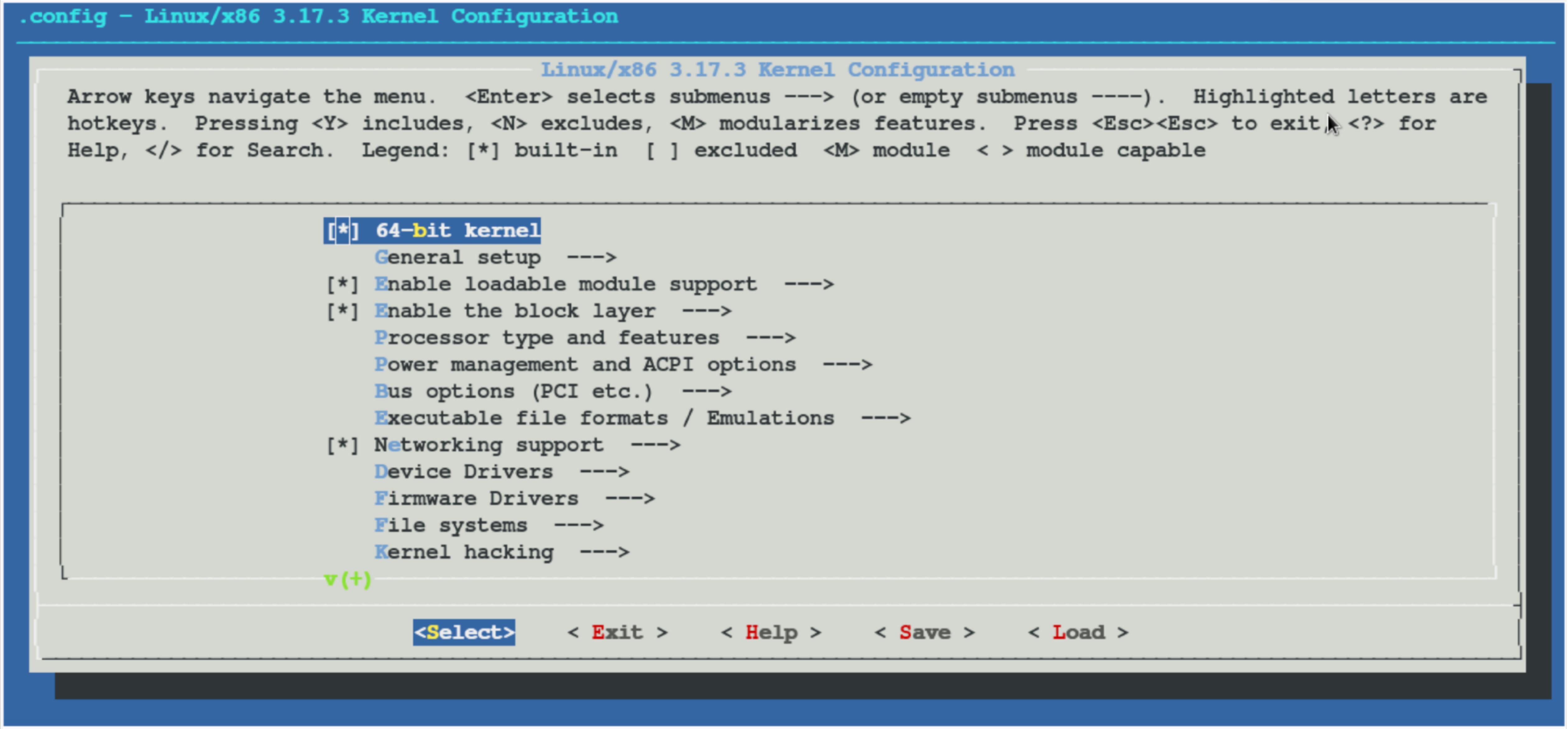
$$2^n$$

```
n = {CLOUD, RAIN}
Products = [{}, {CLOUD}, {RAIN}, {CLOUD, RAIN}]
```

33 features



One different product to each **person** in the world


.config - Linux/x86 3.17.3 Kernel Configuration
Linux/x86 3.17.3 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable
[*] 64-bit kernel
General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Executable file formats / Emulations --->
[*] Networking support --->
Device Drivers --->
Firmware Drivers --->
File systems --->
Kernel hacking --->
v(+)
<Select> < Exit > < Help > < Save > < Load >

$$2^{12.000}$$

How to Implement?

```
void test() {
    // code 1
    #ifdef CLOUD
      // code 2
    #endif
    #ifdef RAIN
      // code 3
    #endif
    // code 4
}
```

!CLOUD and !RAIN
CLOUD and !RAIN
!CLOUD and RAIN

```
void test() {
    // code 1
    #ifdef CLOUD
      // code 2
    #endif
    #ifdef RAIN
      // code 3
    #endif
    // code 4
}
```

GCC

Lua

Vim

APACHE

ghostscript

ubuntu

FreeBSD

irssi
the client of the future

php

Emacs

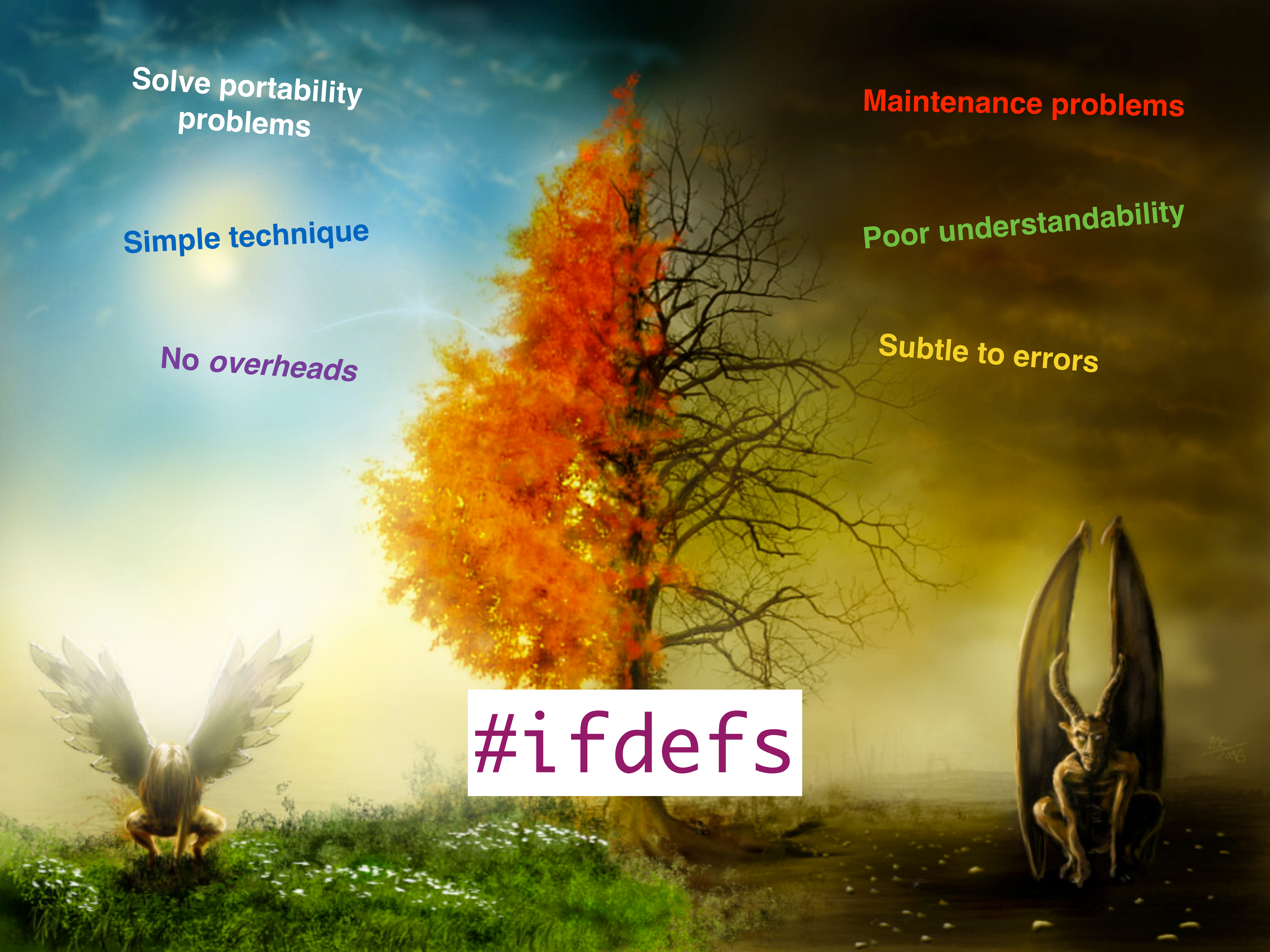
GZIP

CVS

ClamAV

SQLite

python


Solve portability problems
Simple technique
No overheads
Maintenance problems
Poor understandability
Subtle to errors
#ifdefs

Undisciplined vs. Disciplined Annotations

```
#if (WIN32)
if ((ready = select(0, NULL, tp))
#else
if ((ready = select(max_fd + 1, NULL, tp))
#endif
     == -1) {
  ngx_log_error(NGX_LOG_ALERT);
  return NGX_ERROR;
  if (ready == -1) {
    err = ngx_socket_errno;
  } else {
    err = 0;
  }
}
```

```
#if (WIN32)
ready = select(0, NULL, tp);
#else
ready = select(max_fd + 1, NULL, tp);
#endif
if (ready == -1) {
  ngx_log_error(NGX_LOG_ALERT);
  return NGX_ERROR;
  if (ready == -1) {
    err = ngx_socket_errno;
  } else {
    err = 0;
  }
}
```

Previous work…

GPCE 2013

ness and response time. Our results indicate that the discipline of annotations has *no influence on program comprehension and maintenance*, neither for correctness nor for performance (in terms of response time). Although we observed some tendencies, they are not supported by our statistical analysis. However, our experiment

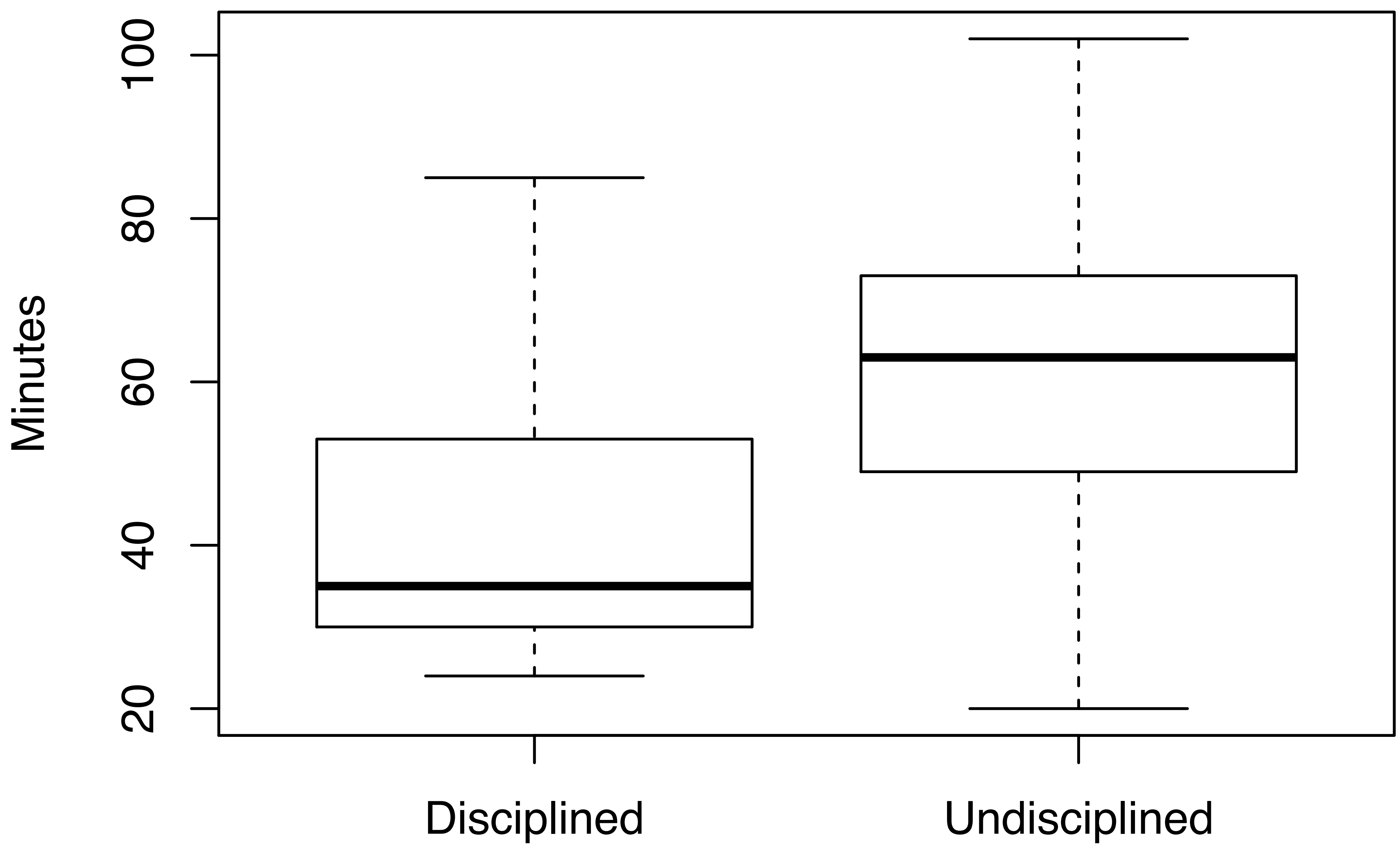
```c
#if defined(FEAT_XCLIPBOARD) || defined(USE_XSMP) || defined(FEAT_MZSCHEME)
    static int busy = FALSE;

#if defined(HAVE_GETTIMEOFDAY) && defined(HAVE_SYS_TIME_H)
    if (msec > 0 && (
#ifdef FEAT_XCLIPBOARD
        xterm_Shell != (Widget)0
#if defined(USE_XSMP) || defined(FEAT_MZSCHEME)
        ||
#endif
#endif
#ifdef USE_XSMP
        xsmp_icefd != -1
#ifdef FEAT_MZSCHEME
        ||
#endif
#endif
#ifdef FEAT_MZSCHEME
    (mzthreads_allowed() && p_mzq > 0)
#endif
        ))
    gettimeofday(&start_tv, NULL);
#endif
    if (busy)
      return 0;
#endif
```

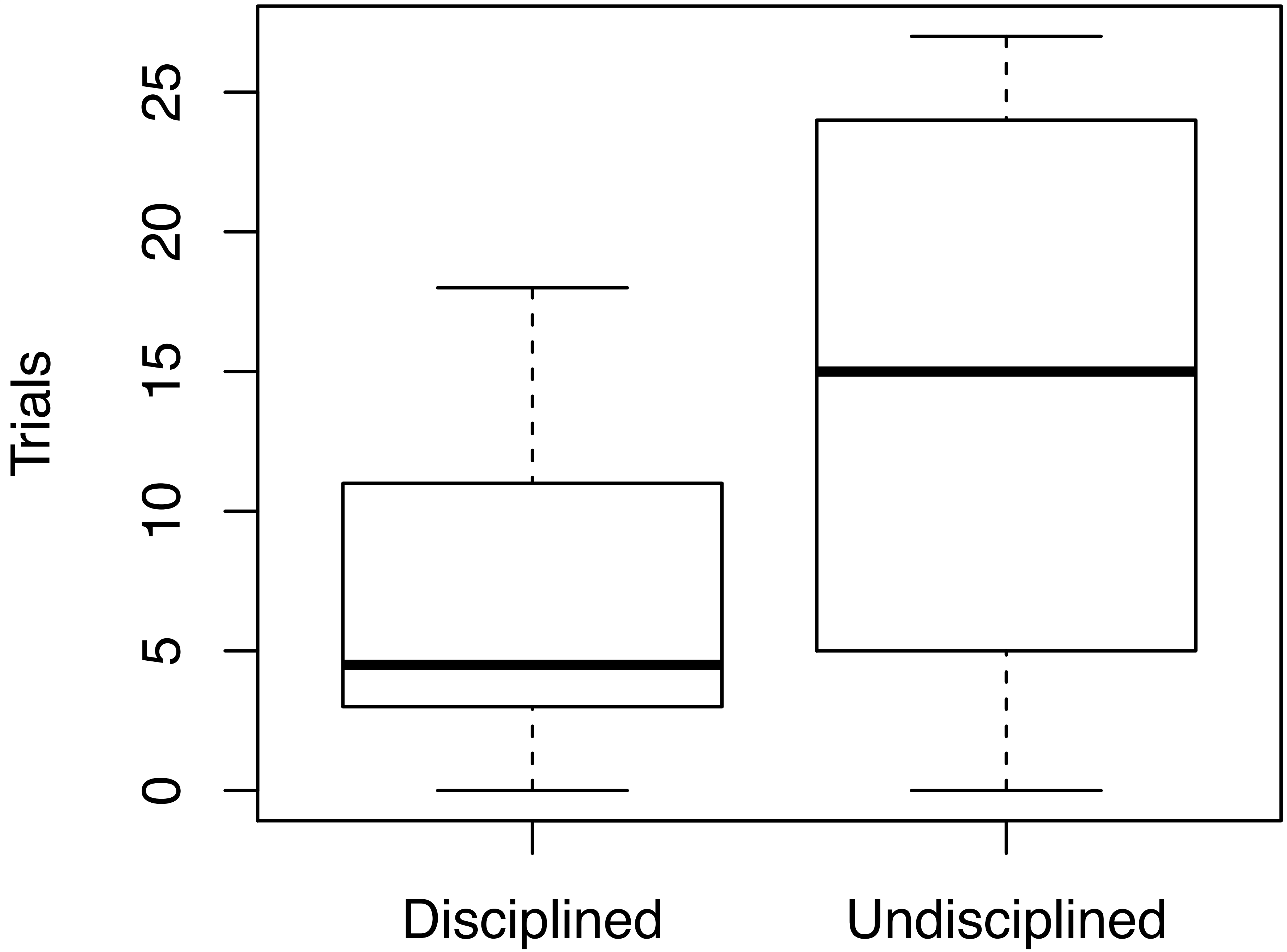
Linux Guidelines

*"Prefer to **compile out entire functions**, rather than portions of functions or **portions of expressions.**"*

We found the opposite



Significant difference!



Significant difference!

Refactoring Undisciplined Annotations

☑ Existing Refactorings...

```c
#if defined (USE_ISPTS_FLAG)
  if (result) {
#endif
    result = open("/dev/ptmx");
    if (!result)
      strcpy(ttydev);
#ifdef USE_ISPTS_FLAG
  }
#endif
```

```c
#if defined (USE_ISPTS_FLAG)
  if (result) {
    result = open("/dev/ptmx");
    if (!result)
      strcpy(ttydev);
  }
#else
    result = open("/dev/ptmx");
    if (!result)
      strcpy(ttydev);
#endif
```

✗ Code clone

✗ Lines of code

Catalogue of Refactorings

Bad Smells	Refactoring
Else inside directives	Else if wrappers
Incomplete if and while statements	Conditional statement wrappers
Case inside directives	Case wrappers
Alternative choice	Alternative statement wrappers
Incomplete else	If statements ending with an else
Incomplete conditions	Conditions
Incomplete commands	Returns and expressions
Incomplete arrays and enums	Data structure definitions
Incomplete function definitions	Function definitions
• • •	• • •

14 refactorings

```
if (condition1
#ifdef expression1
    && condition2
#endif
) {
    // Statements..
}
```

```
bool test = condition1;
#ifdef expression1
    test = test && condition2;
#endif
if (test) {
    // Statements..
}
```

Precondition: original code is not using the variable test in this scope

✓ **Code clone**

✓ **Lines of code**

```
#ifdef expression_1
    if (condition_1) {
#else
    if (condition_2) {
#endif
        // Lines of code..
}
```

```
bool test;
#ifdef expression_1
    test = condition_1;
#else
    test = condition_2;
#endif
if (test) {
    // Lines of code..
}
```

Precondition: original code is not using the variable test in this scope

✓ **Code clone**

✓ **Lines of code** **+2**

```
type function(
#ifdef expression1
    type param_id
#endif
) {
    // Statements..
}
```

```
#ifdef expression1
  #define PARAM type param_id
#else
  #define PARAM ""
#endif
type function(PARAM) {
   // Statements..
}
```

Precondition: original code does not define the macro PARAM in this scope

- ✓ Code clone
- ✓ Lines of code
- ? Code understanding

Evaluating the Refactorings

Frequency of application possibilities

Opinion of developers

Behavior preservation

Code cloning

63

Program Families

1972	905	357	178	270
73	87	23	31	19

7K application possibilities

A

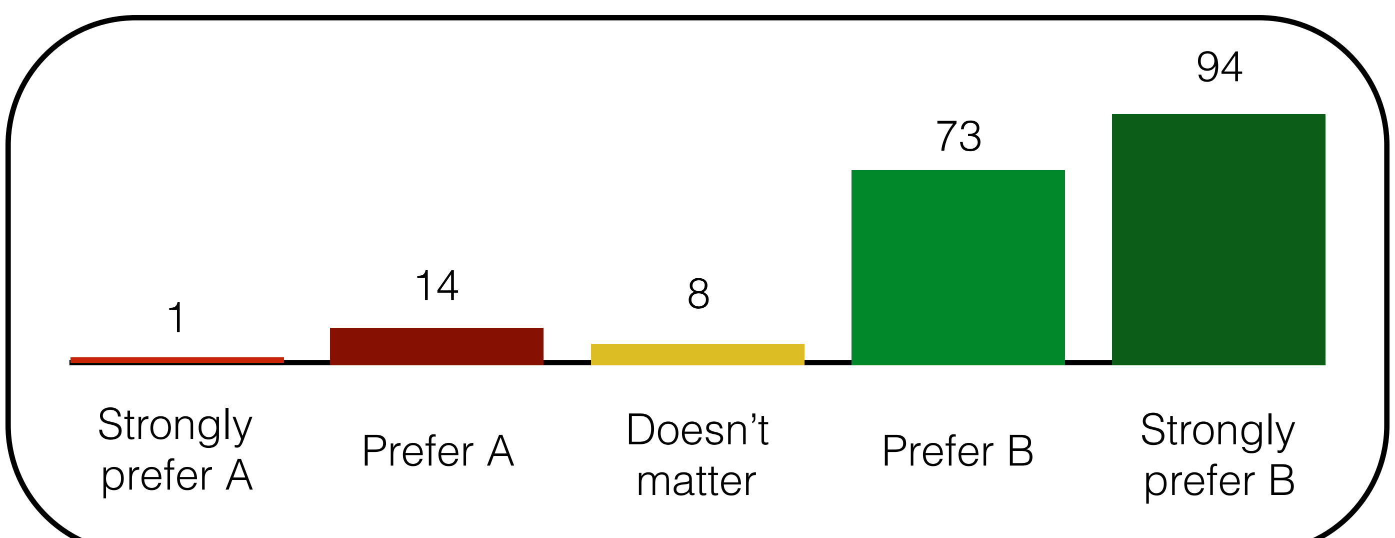
```
mfp = open(mf
#ifdef TS
   , (mode_t) 0600
#else
   , IR | IW
#endif
);
```

7%

B

```
#ifdef TS
mfp = open(mf, (mode_t) 0600);
#else
mfp = open(mf_name, IR | IW);
#endif
```

87%



A

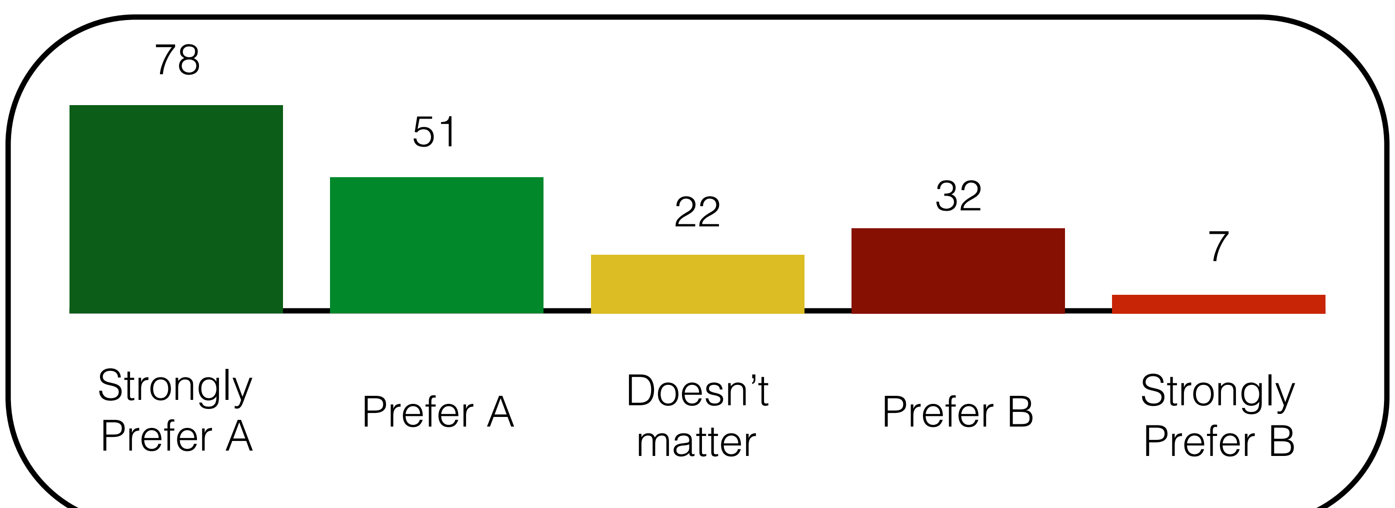
```
bool exec = (bit < 8);
#ifdef TS
exec = exec && (r != NULL);
#endif
if (exec) {
   // STMTS
}
```

B

```
if (bit < 8
#ifdef TS
    && r != NULL
#endif
) {
    // STMTS
}
```

67%

20%



A

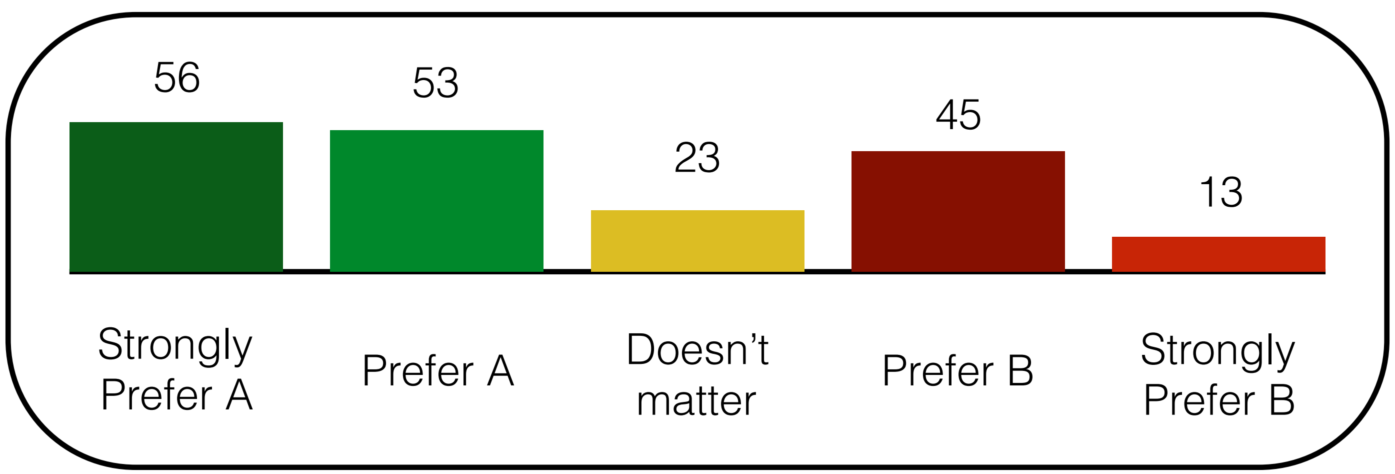
```
#ifdef TS
#define PARAM Ct server,
#else
#define PARAM
#endif
void msgNetbeans(PARAM XT client) {
    // STMTS
}
```

57%

B

```
void msgNetbeans(
#ifdef TS
Ct server,
#endif
XT client) {
    // STMTS
}
```

30%



We received positive feedback from developers when submitting pull requests to discipline undisciplined annotations

```c
#ifdef expression_1
    if (condition_1)
#endif
{
    // Lines of code..
}
```

```c
bool test = 1;
#ifdef expression_1
    test = condition_1;
#endif
if (test) {
    // Lines of code..
}
```

```c
if ( condition_1
#ifdef expression_1
     && condition_2
#endif
){
    // Lines of code..
}
```

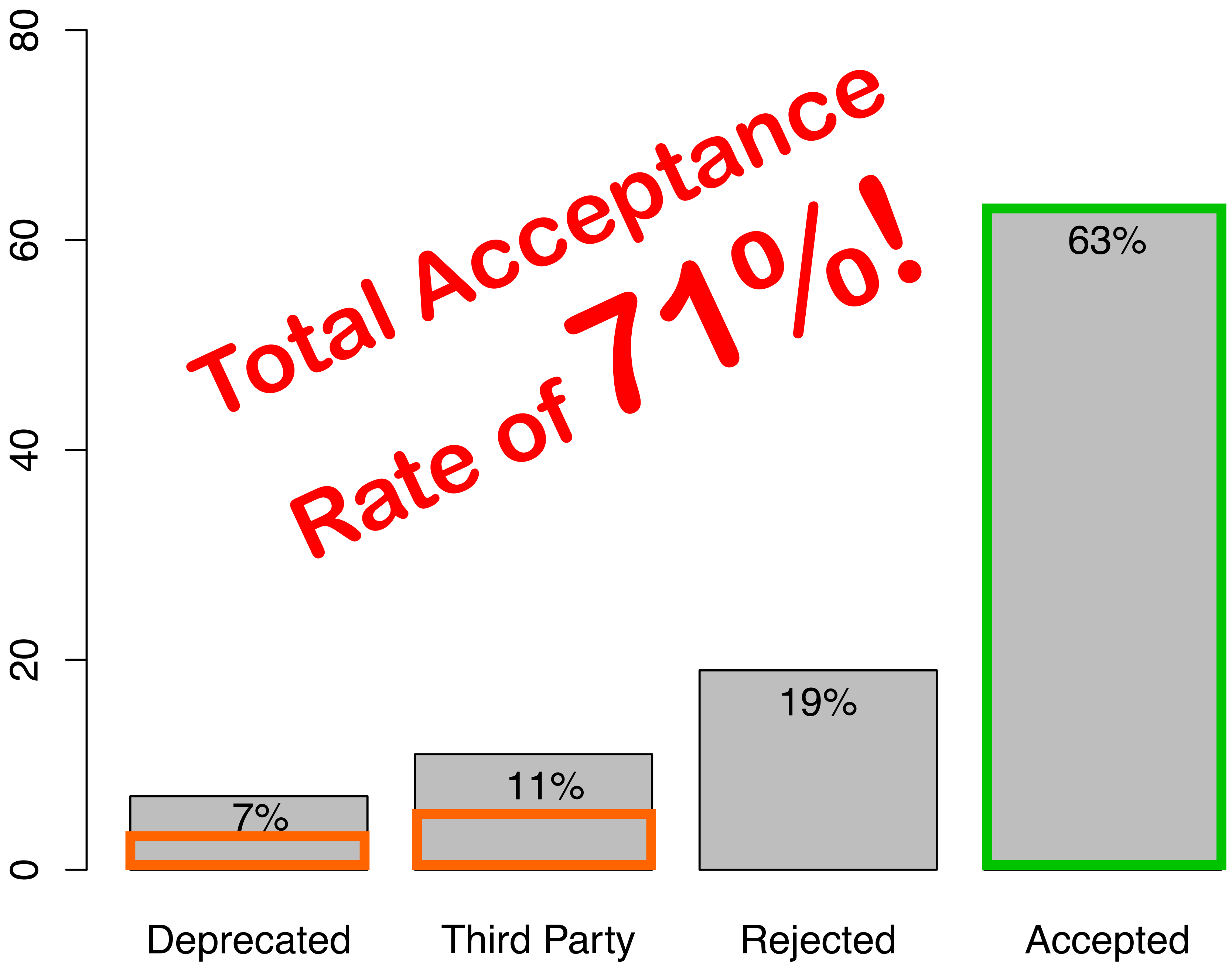
```c
bool test = condition_1;
#ifdef expression_1
    test = test && condition_2;
#endif
if (test) {
    // Lines of code..
}
```

```c
#ifdef expression_1
    if (condition_1){
#else
    if (condition_2){
#endif
    // Lines of code..
}
```

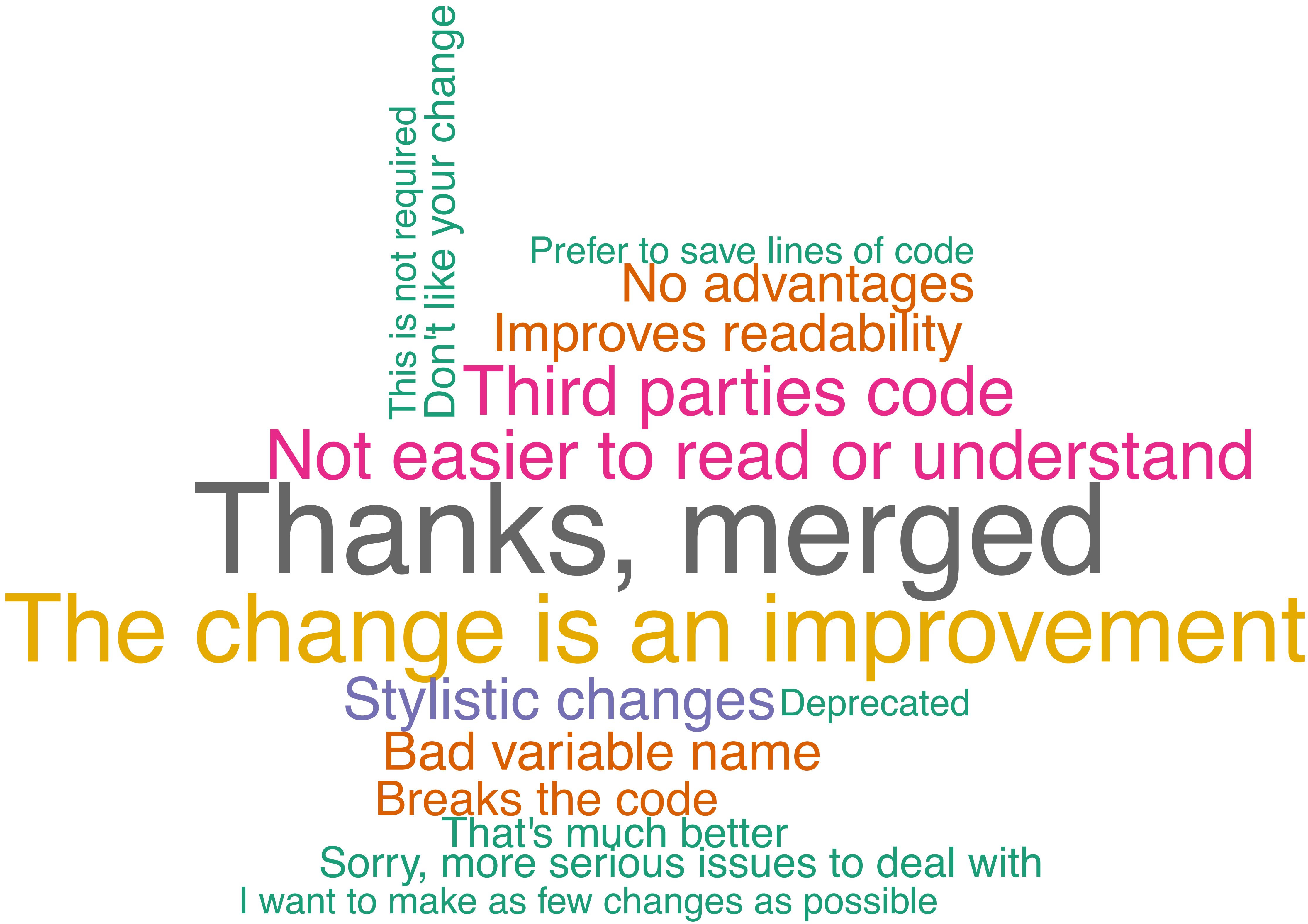
```c
#ifdef expression_1
    bool test = condition_1;
#else
    bool test = condition_2;
#endif
if (test) {
    // Lines of code..
}
```

110 submitted pull requests. **99** decided!

110 different systems! (one per system!)



Might Accept + Accept


This is not required
Don't like your change
Prefer to save lines of code
No advantages
Improves readability
Third parties code
Not easier to read or understand
Thanks, merged
The change is an improvement
Stylistic changes
Deprecated
Bad variable name
Breaks the code
That's much better
Sorry, more serious issues to deal with
I want to make as few changes as possible

Token sequence

```
call (p1
#ifdef EXP
    , p2
#endif
);
```

↦

```
#ifdef EXP
call (p1, p2);
#else
call (p1);
#endif
```

```
#define EXP
call <> p1 <> p2
```

=

```
#define EXP
call <> p1 <> p2
```

```
#undef EXP
 call <> p1
```

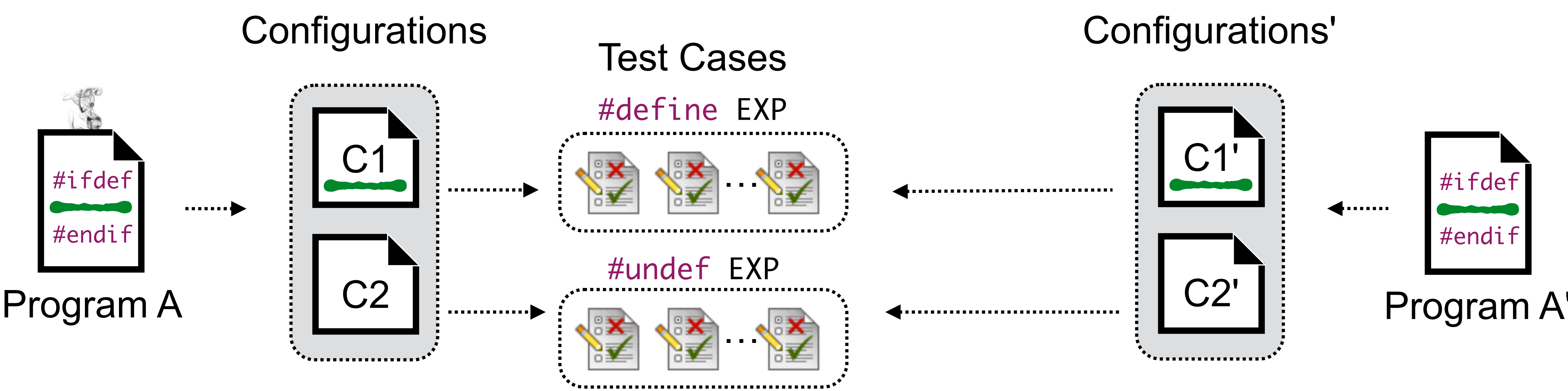
=

```
#undef EXP
 call <> p1
```

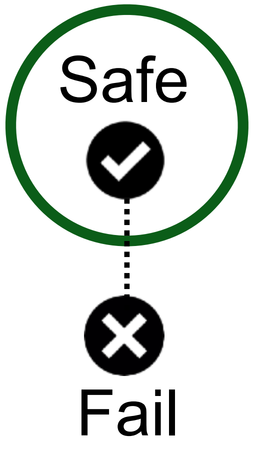
Safe

Fail

Test results



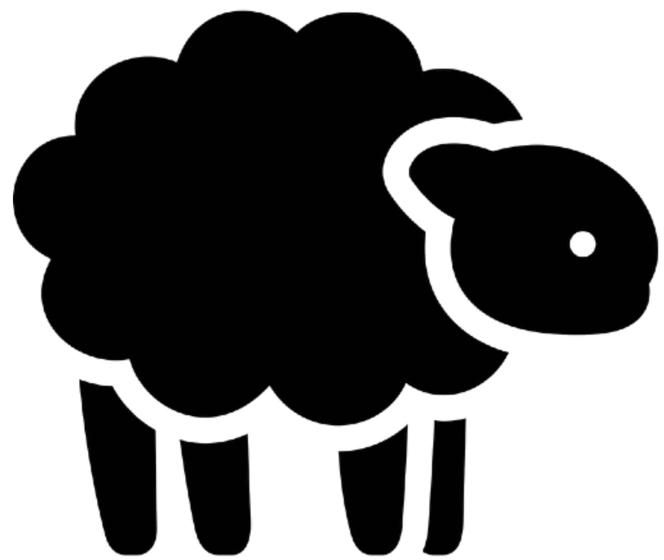
	Config	Input	Output
A	!EXP	10	20
A'	!EXP	10	20

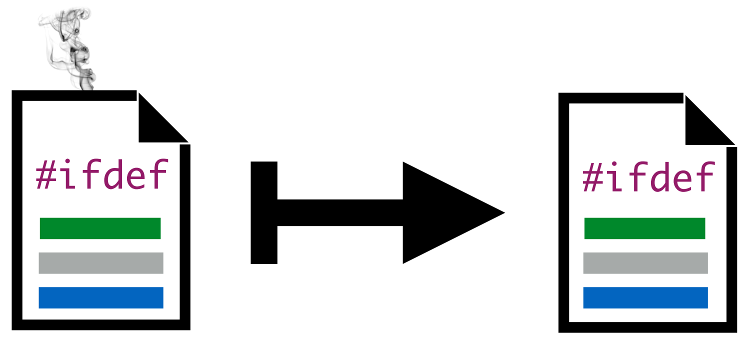


We found and fixed a behavioral change in one refactoring of the catalogue

	R2	R3	R4	R5
Valid programs	7746	7723	14448	6700
Invalid programs	2254	2277	5452	2300
Valid refactorings	7746	7723	14448	6700
Behavioral changes	5	1	5	2

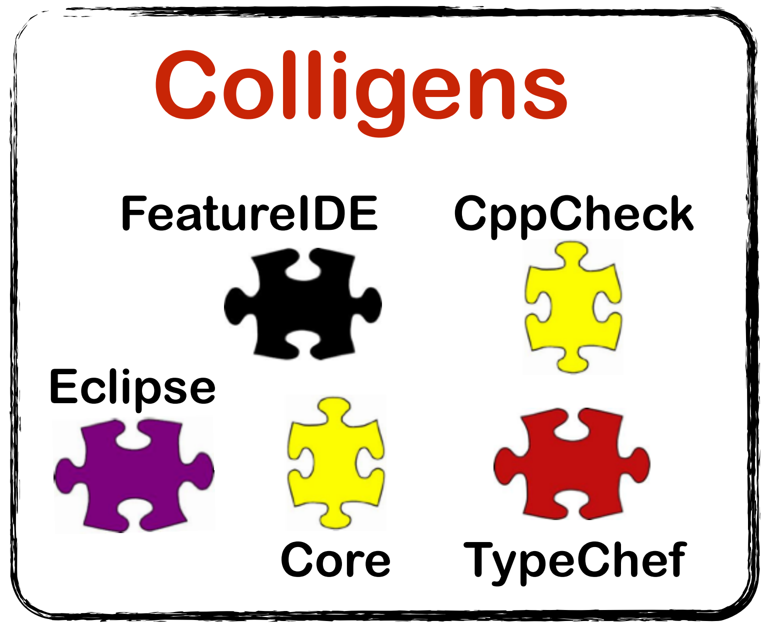
We found and fixed some behavioral changes in the implementation of refactorings

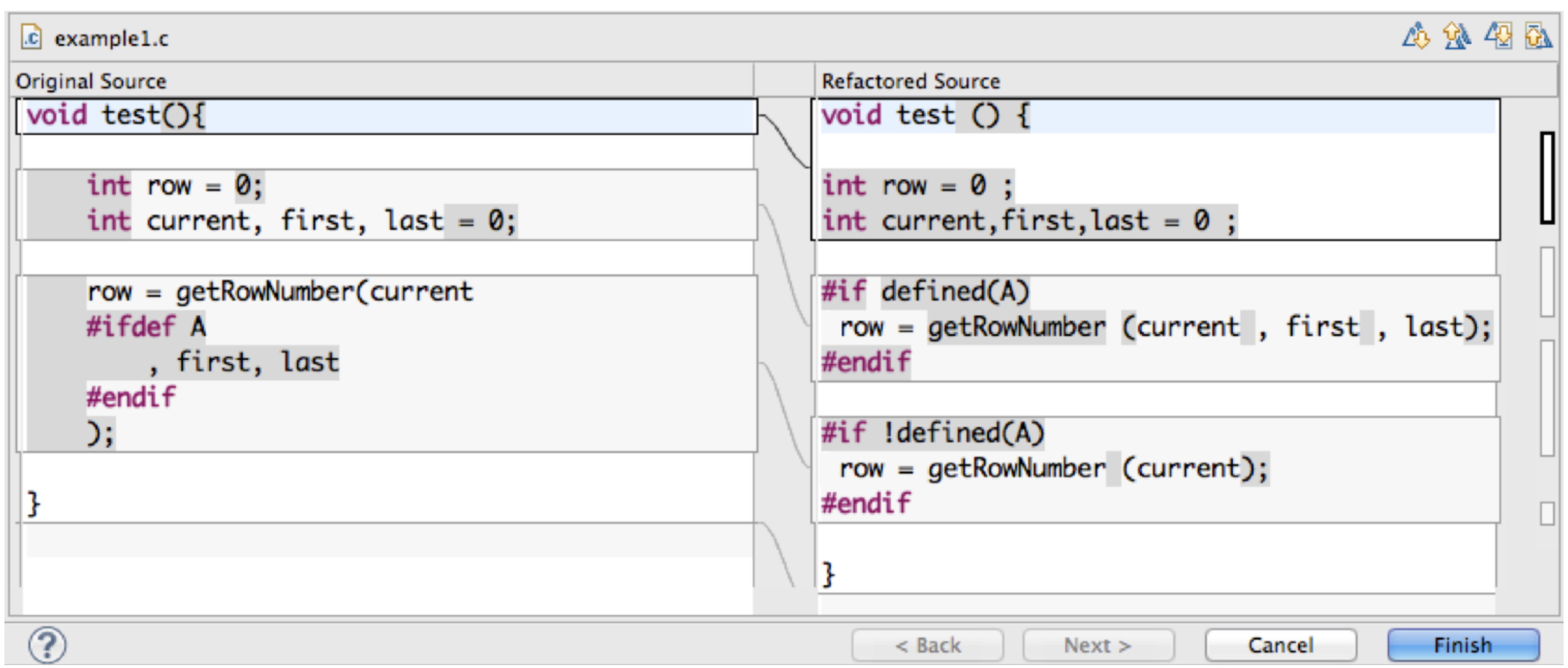




Project	Undisciplined
apache	178
bc	6
dia	31
expat	31
flex	16
fvwm	61
ghostscript	87
gnuchess	2
gzip	19
lighttpd	23
lua	6
mptris	17
Total	477

Clone	LOC	Directives
0	0,18%	2,21%
0	0,12%	0%
0	0,31%	4,06%
0	0,44%	3,87%
0	0,09%	0%
0	0,11%	3,35%
0	0,01%	0,95%
0	0,02%	0%
0	0,64%	4,03%
0	0,08%	1,18%
0	0,12%	3,11%
0	0,78%	3,05%
0	0,04%	2,1%


Colligens
FeatureIDE
CppCheck
Eclipse
Core
TypeChef


example1.c
Original Source
void test(){
int row = 0;
int current, first, last = 0;
row = getRowNumber(current
#ifdef A
, first, last
#endif
);
}
Refactored Source
void test () {
int row = 0 ;
int current,first,last = 0 ;
#if defined(A)
row = getRowNumber (current , first , last);
#endif
#if !defined(A)
row = getRowNumber (current);
#endif
}
< Back
Next >
Cancel
Finish

Ongoing work…

"If my memory serves correctly, it was just a refactoring to simplify the code and make it a little more understandable."

libpng

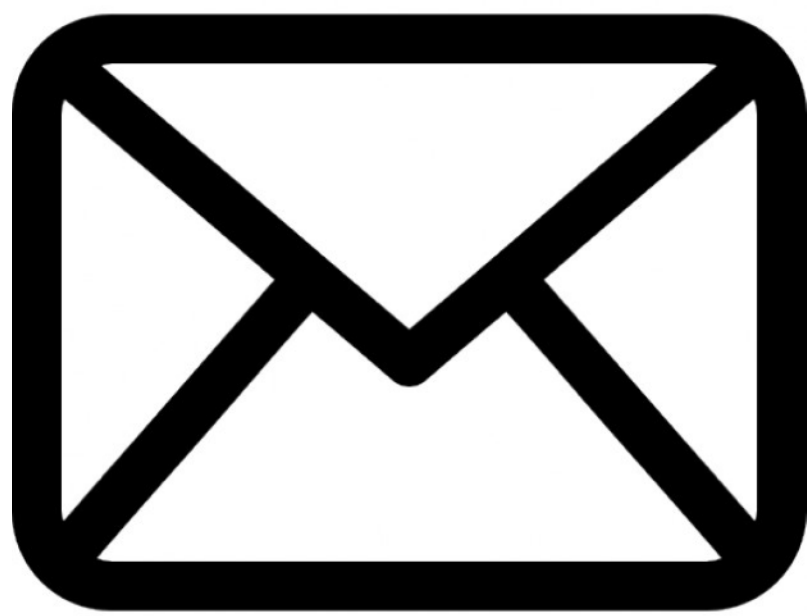
"Because using conditional for only part of one high level instruction makes it less understandable and is likely to raise issue over long time maintainance."

Libxml2

The Discipline of Preprocessor-Based Annotations

Does `#ifdef` `TAG` n’t `#endif` Matter

and let TAG be disabled!




Maceió, Alagoas, Brasil
Takk!

A Catalog of Refactorings to Discipline Preprocessor-Based Annotations

Márcio Ribeiro
http://www.ic.ufal.br/marcio
marcio@ic.ufal.br
@marciomribeiro