`if`s considered Harmful

Or; how to eliminate 90% of your bugs and 99% of your technical debt in one easy step.

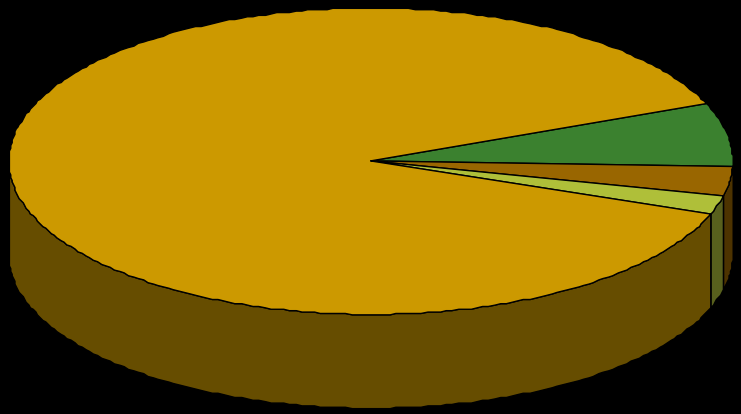
Jules May

JulesMay.co.uk

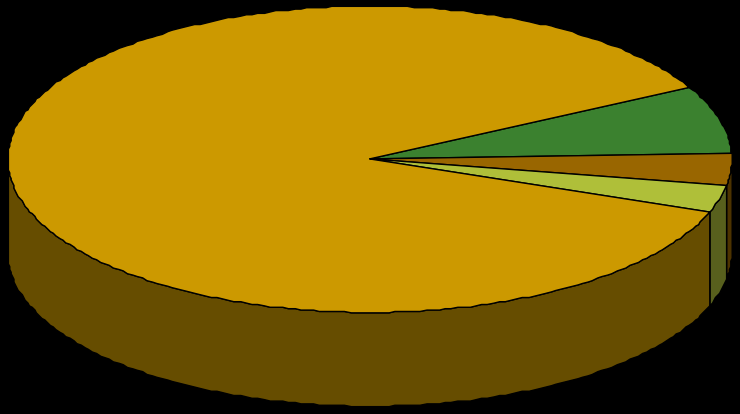
Codebase survey

- ~2.5 million lines
- 5 years development
- 6-25 developers

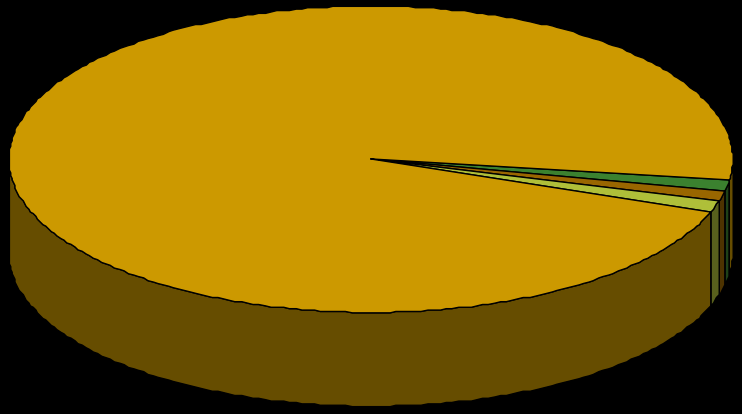
Bugs: all



Bugs: primary



Bugs: secondary



goto considered harmful

Edsger Wybe Dijkstra (1930-2002)

EWD 215: A case against the goto statement (1968)

EWD215 - 0

A Case against the GO TO Statement.

by Edsger W.Dijkstra
Technological University
Eindhoven, The Netherlands

Goto considered harmful

Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to effectuate the desired effect; it is this process that in its dynamic behaviour has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visiulize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost best to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF

"Goto statement considered harmful", Commun. ACM 11 (1968), 3: 147–148.

`goto` considered harmful - *why?*

What a program looks like:	What a program means:
▪ Static map	▪ Dynamic process
▪ Spatial - locations	▪ Temporal - instants

Execution coordinate

Execution coordinate

```
10: a=10;
11: b=20;
12: c=a+b;
13: print (c);
14: d=sqrt (c);
15: end;
```

Execution coordinate

```
10: a=10;
11: b=20;
12: c=a+b;
13: print (c);
14: d=sqrt (c);
15: end;
```

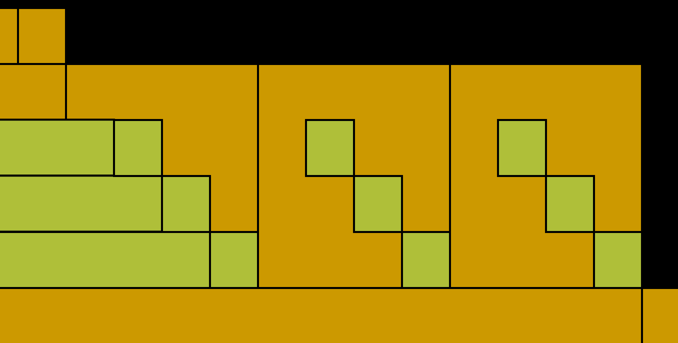
Rt:9180:

Rt:4126:

10:

Execution coordinate

```
10: a=10;
11: for b=(1..3) {
12:     c=a+b;
13:     print (c);
14:     d=sqrt (c);
15: }
```

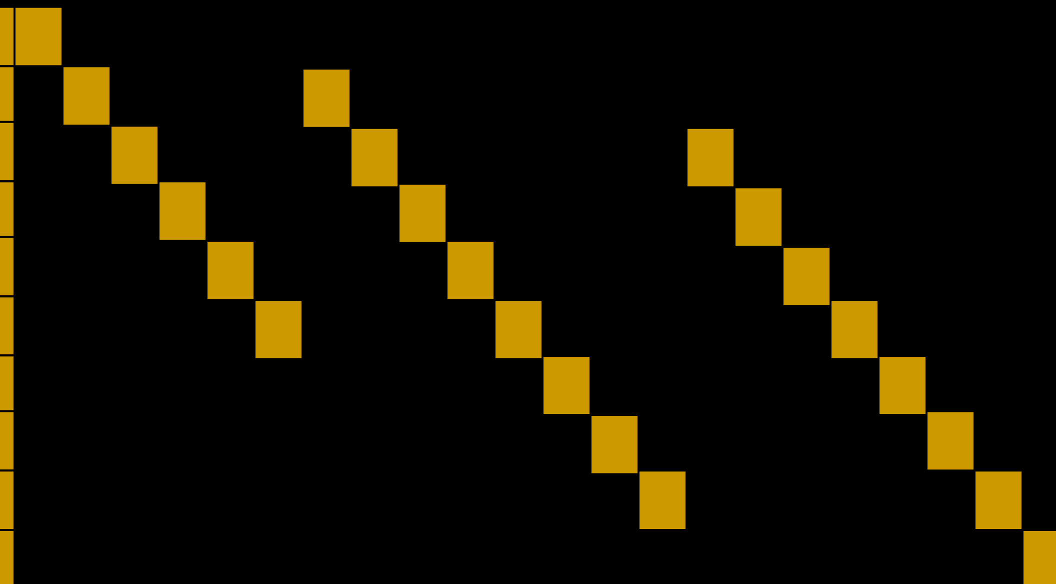


12:

11(1):

Execution coordinate *- fail?*

```
10: a=9;b=-10
11: if b=11 goto 18;
12: b=b+1;
13: if b>12 goto 10;
14: c=a+b;
15: if c<0 goto 11;
16: print c;
17: d=sqrt (c);
18: goto 12;
19: end;
```



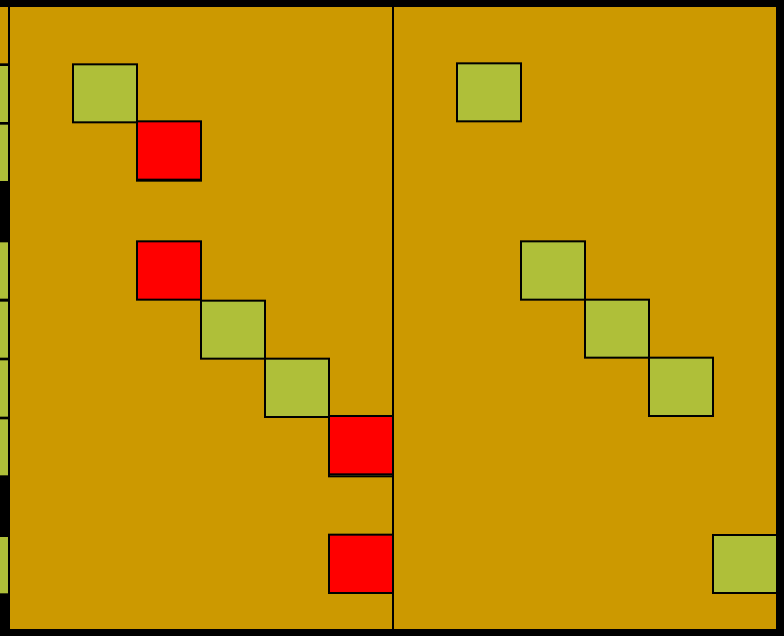
1[illegible]:

Execution coordinate *- take-away*

- Coordinate is tuple of (location *or* repetition)
  - e.g. [`11(2), 13, Rt:4126`]
- Shows us *(for structured code)*:
  - Location in code
  - Moment in execution time
  - Entire history leading up to that moment.
- Not all `gotos` violate ‘structure’:
  - `return, continue, break`
  - loops
  - assembler

Execution coordinate *- the problem*

```
10: for b=(-1..1) {
11:     if (b<0)
12:         print (GORED);
13:     else
14:         print (GOBLUE);
15:     print (b);
16:     if (b<0)
17:         print (‘ DR\n’);
18:     else
19:         print (‘ CR\n’);
20: }
```



Execution coordinate *- the problem*

```
10: for b=(-1..1) {
11:     if (b<0)
12:         print (GORED);
13:     else
14:         print (GOBLUE);
15:     print (b);
16:     if (b>0)
17:         print (‘ DR\n’);
18:     else
19:         print (‘ CR\n’);
20: }
```

```
10: for b=(-1..1) {
11:     if (b<0) {
12:         print (GORED);
13:         print (b);
14:         print (‘ DR\n’);
13:     } else {
14:         print (GOBLUE);
15:         print (b);
16:         print (‘ CR\n’);
19:     }
20: }
```

Execution coordinate - *summary*

~~`goto`~~ `if..goto`	`if..if`
▪ Destroys temporal identity ▪ Destroys history ▪ Creates flow ‘spaghetti’	▪ Conflates temporal identity ▪ Destroys history ▪ Creates flow ‘wormholes’
Imply `goto` from code structure ▪ Not all `goto`s are evil	Imply decision from code structure ▪ Not all conditions are evil

`if`s considered harmful

- Need a new notion of ‘if-structure’:
  - Conditions are ‘concentrated’
  - Decisions are implied
- Need new mechanisms for `if`
  - Don’t overstay your welcome
  - Specialist classes & downcasting
- Need to express those in conventional languages

`if`s considered harmful

`Assert`

ifs considered harmful - **assert**

```
float sqrt (a) {
   assert (isNumeric (a)) or error ('Numeric type required');
   assert (a>=0) or throw DomainError (…);
   r = // do some computation on a
   return r;
}
```

- Principle: not to *control* the flow, but to *abort* it.
- Species:

```
assert (…) or return (); -- return default value or null object
assert (…) or throw …;   -- refuse to process; contract violation
assert (…) or die ();    -- do shutdown or recovery process
assert (…) or error ();  -- Generate compile-time error (if possible)
```

`if`s considered harmful - `assert`

In practice:

- It's easy:
  - `if (!condition) throw …;`
- `assert () or error` needs compiler support.

`if`s considered harmful

Don’t overstay your welcome

`if`s considered harmful - *Values*

```
number fact (number n) {
   if (n==1)
      return 1;
   else
      return n*fact(n-1);
}

string fact (string s) {
   return "Can't fact " + s;
}
```

```
fact (1)=1;
fact (number n)=n*fib(n-1);
fact (string s)="Can't fact "+s;
```

ifs considered harmful - *Values*

```
for b=(-1..1) {
   if (b<0)
       print (GORED);
   else
       print (GOBLUE);
   print (b);
   if (b<0)
      print (‘ DR\n’);
   else
      print (‘ CR\n’);
}
```

```
for b=(-1..1) {
   print fmt (b);
}

string fmt (number b<0){
   return GORED + b + ‘ DR\n’;
}
string fmt (number b){
   return GOBLUE + b + ‘ CR\n’;
}
string fmt (number b=0){
   return GOBLACK + ‘nil\n’;
}
```

`if`s considered harmful - *Values*

In practice:

- Do conditions before anything else,
  - e.g. fact example
- Use ordinary polymorphism over specialist classes
  - e.g. `fmt (plusMoney (b)) {…};`
- Use Haskell!

`if`s considered harmful

Specialists

`if`s considered harmful - *specialists*

```
class littleClass {
}

class bigClass {
   littleClass c = null;

   bigClass () {} // ctor. might set c;

   bigMethod () {
      if (isNull (c)) {
         // do something
      }
      else {
         // do something similar involving c;
      }
   }
}
```

`if`s considered harmful - *specialists*

```
littleClass c = 4;     -- means: c = littleClass (4);
littleClass c = null;  -- means c = null;
                       -- should mean: c = littleClass (null);
```

Null pointer	Null object
▪ No interface at all	▪ Exposes the class interface
▪ Needs `if !isNull()` on each use	▪ Can be used naively
▪ All uses must be checked at runtime	▪ All uses can be checked at compile time

`if`s considered harmful - *specialists*

```
class littleClass {
}

class bigClass {
   littleClass c = littleClass (null); c = littleClass (null);

   bigClass () {} // ctor. might set c;

   bigMethod () {
      if (isNull (c)) {
         // do something
      }
      else {
         // do something involving c
      }
   }
}
```

`if`s considered harmful

Specialists and downcasts

`if`s considered harmful - *downcasts*

Every method has an implied condition:

```
if (this.isInstanceOf (thisClass)) {…}
```

```
class littleClass {
   private littleClass () {…}
   // declarations
   // methods
   littleClass ()
      -> littleNull ();
   littleClass (int k)
      -> littleInt (k);
   littleClass (object obj)
      -> littleThing (obj);
}
```


```
class littleNull:littleClass {
   littleNull () {…} //ctor
   // implementations
}

class littleInt:littleClass {
   littleInt (int k) {…} //ctor
   // implementations
}

class littleThing:littleClass {
   littleThing (object obj) {…}
   // implementations
}
```

`if`s considered harmful - *downcasts*

```
for b=(-1..1) {
   if (b<0)
       print (GORED);
   else
       print (GOBLUE);
   print (b);
   if (b<0)
       print (` DR\n’);
   else
       print (` CR\n’);
}
```

```
for b=(-1..1) {
   print (money (b));
}
```

```
class money:number { //string number:toString () inherited
   private money (number (n)) {}
   private class plusMoney:money {
      toString () { return GOBLUE + :toString() + “CR\n”);}
   }
   private class minusMoney:money {
      toString () { return GORED + :toString() + “DR\n”);}
   }
   money (number n) -> n>=0 ? plusMoney (n) : minusMoney (n);
}
```

`if`s considered harmful - *downcasts*

In practice:

- Use abstract base classes with static factory methods,
  - e.g. `x=myclass:new (…);`
- Expose the subclasses directly
  - e.g. `money b = plusMoney (4);`
- Precompile
  - e.g.

```
abstract downcastable class myclass {…};
new myclass(…) -> myclass:factory (…)
```

`if`s considered harmful - *Summary*

- `goto` is harmful.
- It causes spaghetti code, which is unreliable.
- Actually, it was always `if` that was harmful.
- It causes wormhole code, which is unreliable.

`if`s considered harmful - *Summary*

- The antidote to `goto` is ‘flow-structure’
- Structured languages replace `goto` with:
  - `for`
  - `while`
  - `throw`
- The antidote to `if` is ‘decision-structure’
- A decision-structured language would replace `if` with:
  - `assert () or …`
  - value-polymorphism
  - downcasts

`if`s considered harmful - *Summary*

- If you're careful, you can do structured programming with `goto`s.
- If you're careful, you can do decision-structured programming with `if`s
  - …while you're waiting for language technology to catch up.

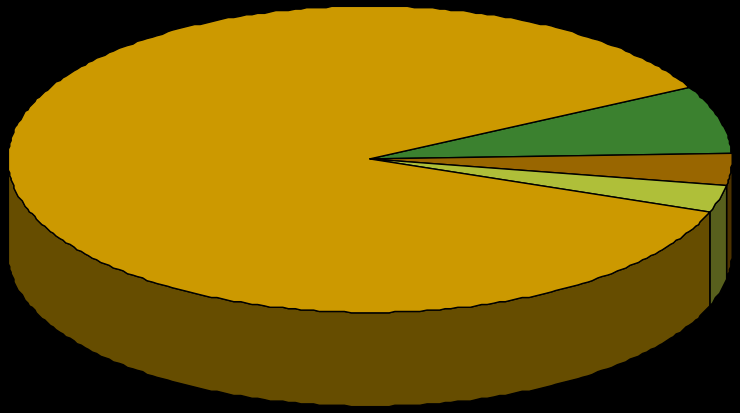
Take-aways

- (Misuse of) `if` causes more bugs than all other causes combined.
- This shouldn't be news – it's `goto` all over again;
- We need decision-structured language constructs;
- In the meantime, good programming hygiene helps.

`if`s considered harmful

Questions?

Bugs: primary



Bugs: secondary