THE UNIVERSITY *of* EDINBURGH
informatics

Usability and Confusion in Configuration Languages

Paul Anderson <dcspaul@ed.ac.uk>

Work by Adele Mikoliunaite

with Kami Vaniea

http://homepages.inf.ed.ac.uk/dcspaul

Motivation

Configuration errors are responsible for a lot of system failures

- 2009 - the Swedish Internet down for 1 hour
- 2010 - Facebook down for 2.5 hours

Administrators have to deal with many different “languages”

- Every application has its own configuration file format
- Special-purpose configuration languages can be very complex
- We suspect that this is a common source of errors

We are interested in designing new languages

- Which are both more expressive, but also more intuitive
- This involves understanding what is “intuitive” to real, working system administrators

The Survey

Adele surveyed about 350 users

- Mostly real, working system administrators
- Representative of the target audience

Asked questions about a small number of examples

- The examples use a “made up” but typical language
- Very little direction given - looking for intuitive answers
- Multiple-choice questions & opportunity for free-text comments

Questions focussed on a number of concepts

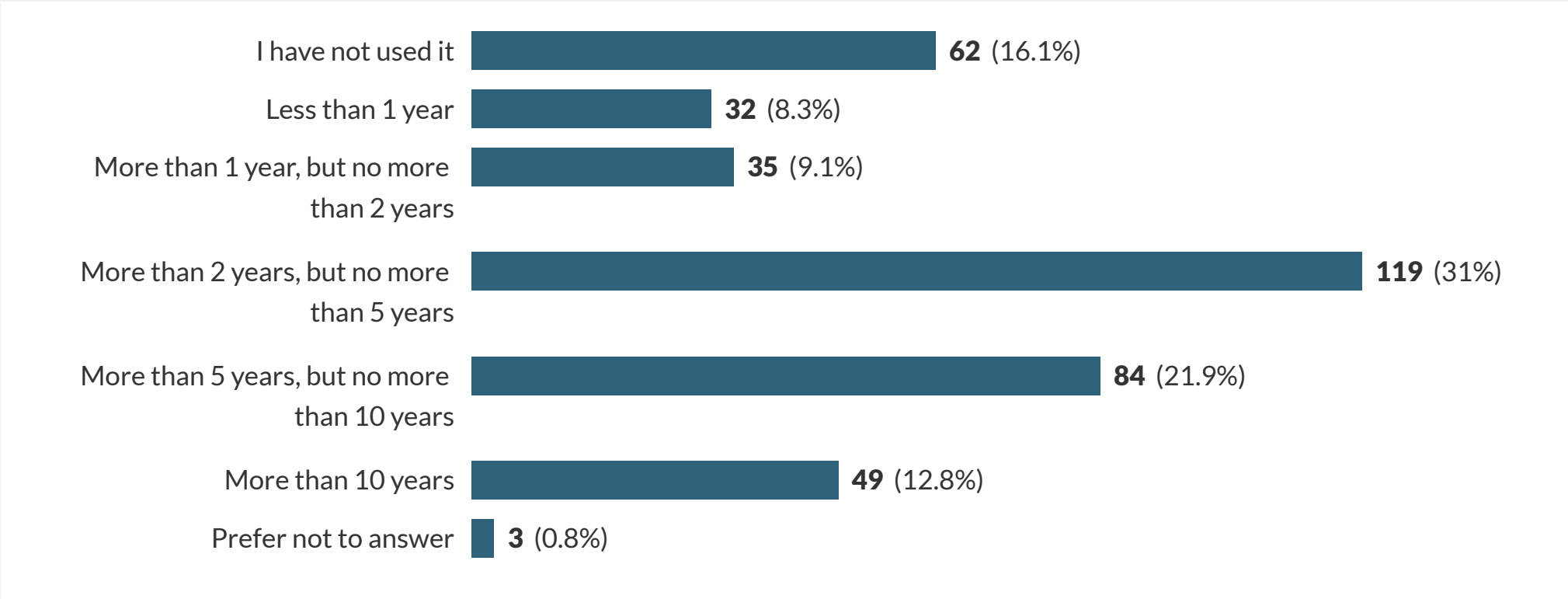
- “Order”
- “Inheritance”
- “References” (static vs dynamic) & “Scope”

There are no “right” or “wrong” answers

- But the results were very interesting!

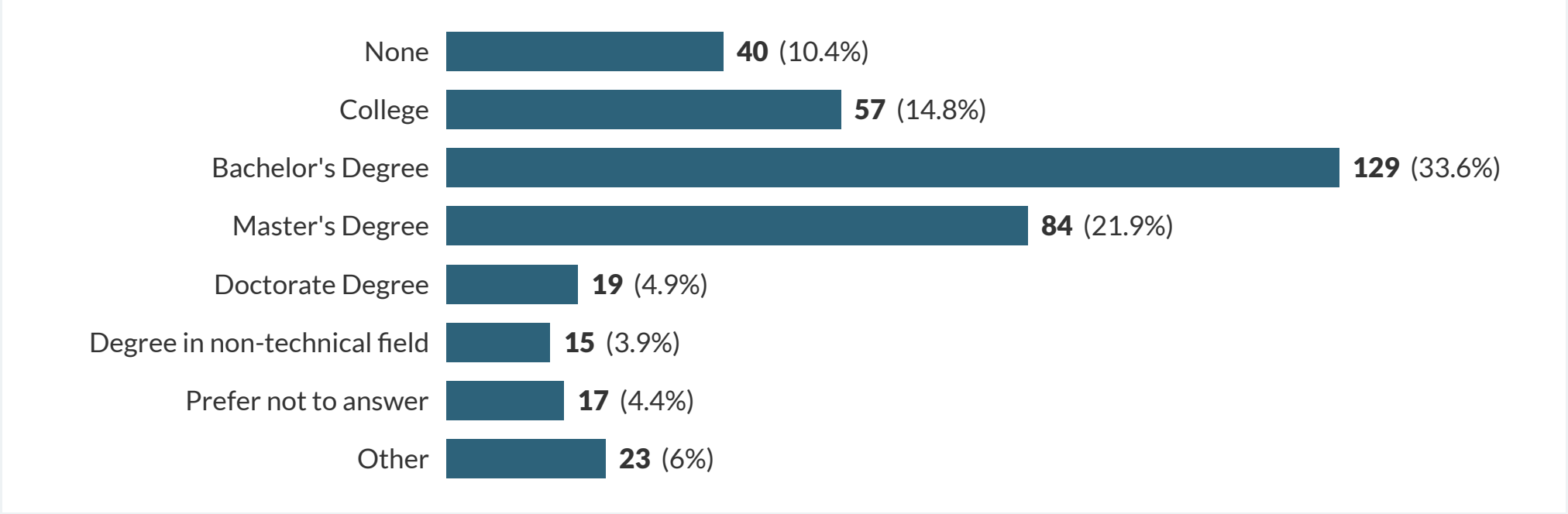
Experience

How many years have you used configuration management?



Education

What is the highest level of education you have achieved in Computer Science/IT field?



Some Background ...

Configuration Files

Configuration Languages

Motivation ...

- A common language for specifying configuration file contents
  - or at least, a macro-language for composing pre-defined fragments and providing parameterised templates
- Enforcing relationships between parameters in different files
  - the firewall permits access to the port being used by the web server
- Sharing of common configuration between different machines
  - machines A and B are both web servers
- Enforcing relationships between the configuration of different machines
  - the client should use the same port number as the server
- Managing policy at a higher level
  - students should not be allowed to log in to any machine with access to the exam papers

Imperative vs Declarative Languages

Imperative languages

- These "feel like" imperative scripting languages
  - with some domain-specific features
- They provide operations to explicitly manipulate configuration files
  - and control structures to sequence the operations
  - eg. "add this line to file X if it is not already present"
- They are popular because this is familiar to most sysadmins

Declarative languages

- These "feel like" the configuration files themselves
  - most of the data is attribute/value pairs of configuration parameters
- They provide features to specialise and compose configurations
  - parameterisation, inclusion, inheritance

Most practical languages have elements of both of these

- But they will tend towards one approach or the other

Imperative vs Declarative Languages

Imperative (Bash script)

```
if [ [ 0 ne $(getent passed elmo > /dev/null)$? ]
then
  user add elms –gid sysadmin -n
fi
GID=`getent passwd elmo |awk -F: {print $4}'`
GROUP=`getent group $GID |awk -F: {print $1}'`
if [ $GROUP != $GID ] && [ $GROUP != sysadmin ]
then
  user mod –gid $GROUP $USER
fi
```

Declarative (Puppet)

```
user { 'elmo':
  ensure => present,
  gid => 'sysadmin',
}
```

Order ...

Order

"Order Doesn't Matter"

- Is a mantra of declarative configuration advocates
- But in practice, it very often does!

There are different types of "order"

- "Lexical", "Evaluation", and "Deployment"

Hidden dependencies can cause deployment order problems

- We are not concerned with this aspect here
- But it does add to the confusion in practice …

```
package { 'golang-go': ensure => present }
package { 'perl': ensure => absent }
```

Password File

This looks “declarative” ...

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

What happens if we have a duplicate name?

- Last one wins?
- First one wins?
- Random one wins?
- Error?
- Attributes composed in some way?

Apache

This looks “declarative” …

```
<IfModule mime_module>
    TypesConfig conf/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
</IfModule>
```

What happens if we have duplicate values?

- Evaluation order and lexical order are not the same! …

“How the sections are merged"

The configuration sections are applied in a very particular order. Since this can have important effects on how configuration directives are interpreted, it is important to understand how this works. The order of merging is:

1. <Directory> (except regular expressions) and .htaccess done simultaneously (with .htaccess, if allowed, overriding <Directory>)
2. <DirectoryMatch> (and <Directory "~">)
3. <Files> and <FilesMatch> done simultaneously
4. <Location> and <LocationMatch> done simultaneously
5. <If>

Apart from <Directory>, each group is processed in the order that they appear in the configuration files. <Directory> (group 1 above) is processed in the order shortest directory component to longest. So for example, <Directory "/var/web/dir"> will be processed before <Directory "/var/web/dir/subdir">. If multiple <Directory> sections apply to the same directory they are processed in the configuration file order. Configurations included via the Include directive will be treated as if they were inside the including file at the location of the Include directive.

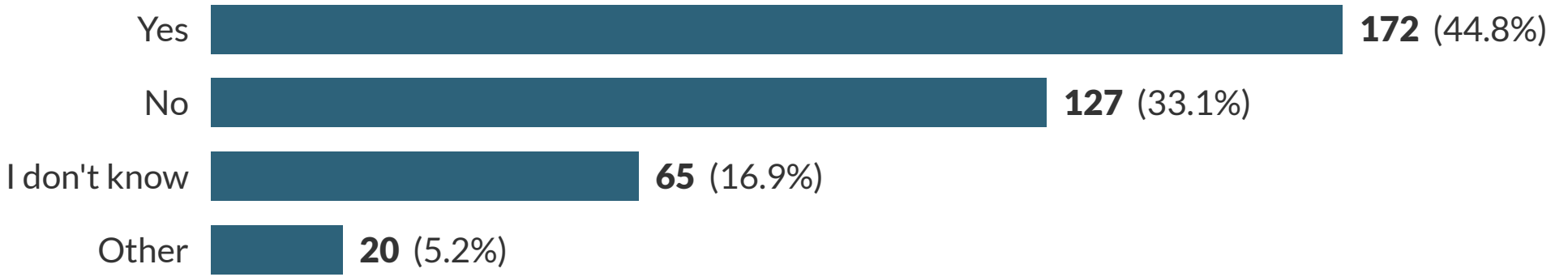
Sections inside <VirtualHost> sections are applied after the corresponding sections outside the virtual host definition. This allows virtual hosts to override the main server configuration.

When the request is served by mod_proxy, the <Proxy> container takes the place of the <Directory> container in the processing order.

https://httpd.apache.org/docs/2.4/sections.html

Do you think order matters?

(in the pseudo code used in this survey)

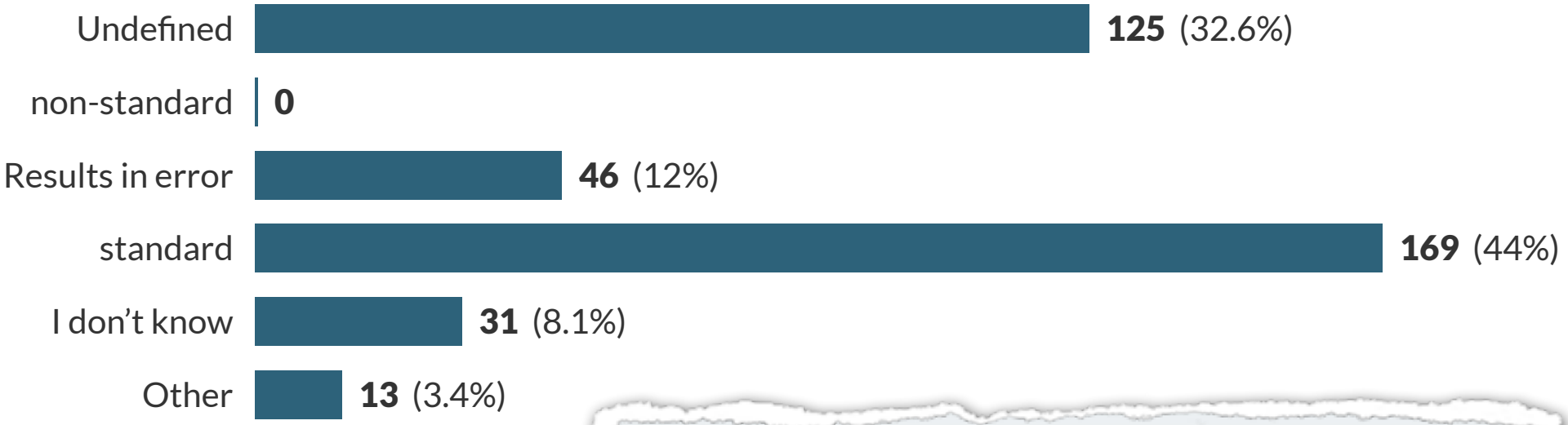


- *"I interpreted code order as irrelevant."*
- *"I didn't spot any cases where ordering would have been significant to interpretation. "*
- *"I interpreted code order in a way that a computer would read through it and execute it in order and with indentations denoting layers."*
- *"Order of properties should not matter. Order of variables should."*
- *"I worked on the assumption that order within a class was significant, but that order of class declaration was not."*

Profile.account

```
User {
   $card = active,
   Profile {
      account = $account,
      type = $type
   }
   $account = standard
}
```

Profile.account




```
User {
  $card = active,
  Profile {
    account = $account,
    type = $type
  }
  $account = standard
}
```

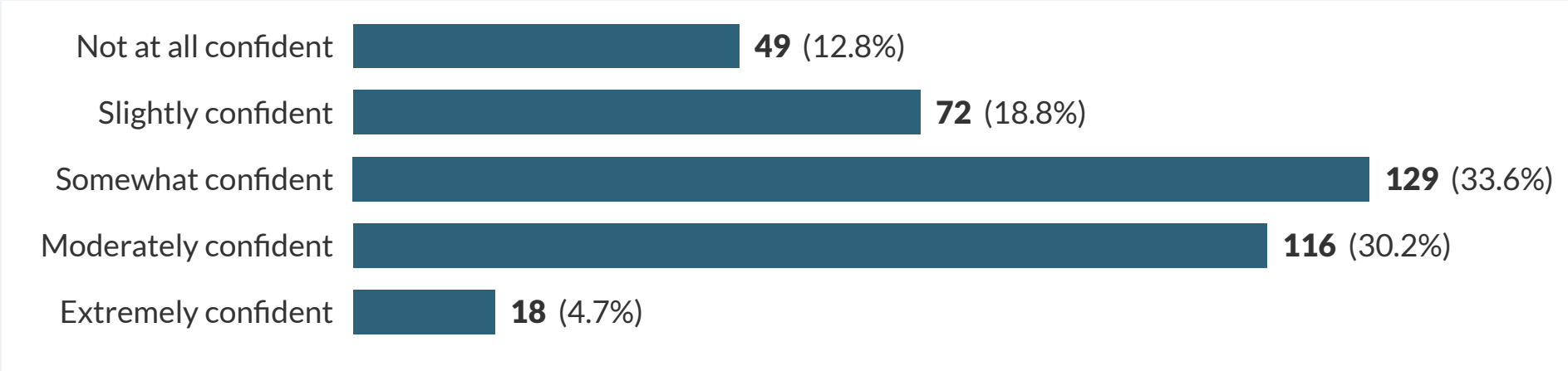
Confidence

It seems reasonable to have different answers

- Since we didn't give any guidance as to how the "made-up" language should be interpreted.

But ...

- How confident were you? ….



Inheritance ...

Inheritance

Inheritance is common in declarative configuration languages

- This is usually “instance” inheritance, not “type” inheritance
- This is very useful for specifying variants of some default configuration

The semantics can be extremely complicated

- Puppet recommends against the use of inheritance, except for a few specific purposes
  - Google “puppet inheritance”
- But the alternatives are not good
  - explicit parameterisation

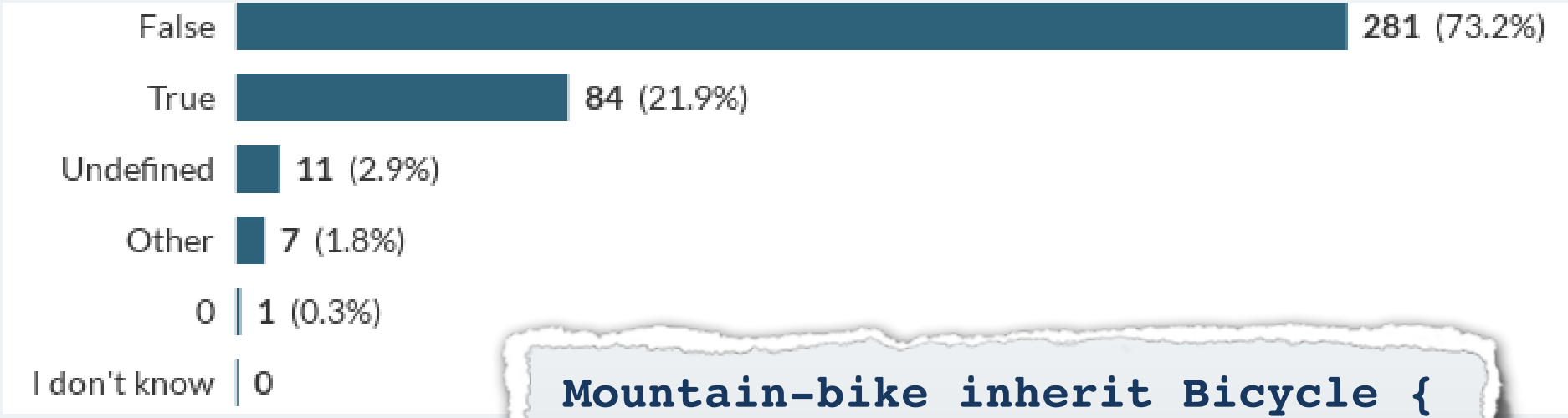
Multiple inheritance is not usually supported

- But is often desirable

Bicycle.kickstand

```
Mountain-bike inherit Bicycle {
   tyre = 26,
   kickstand = true
}
Bicycle {
   wheels = 700,
   tyre = 28,
   kickstand = false,
   brakes = true
}
```

Bicycle.kickstand




```
Mountain-bike inherit Bicycle {
   tyre = 26,
   kickstand = true
}
Bicycle {
   wheels = 700,
   tyre = 28,
   kickstand = false,
   brakes = true
}
```

Player.showbalance

```
Box {
   show-balance = true,
}
Player {
   tracks = 573,
   genres = 11,
   show-balance = false
}
MusicBox inherit Box inherit Player {
    genres = 9,
}
```

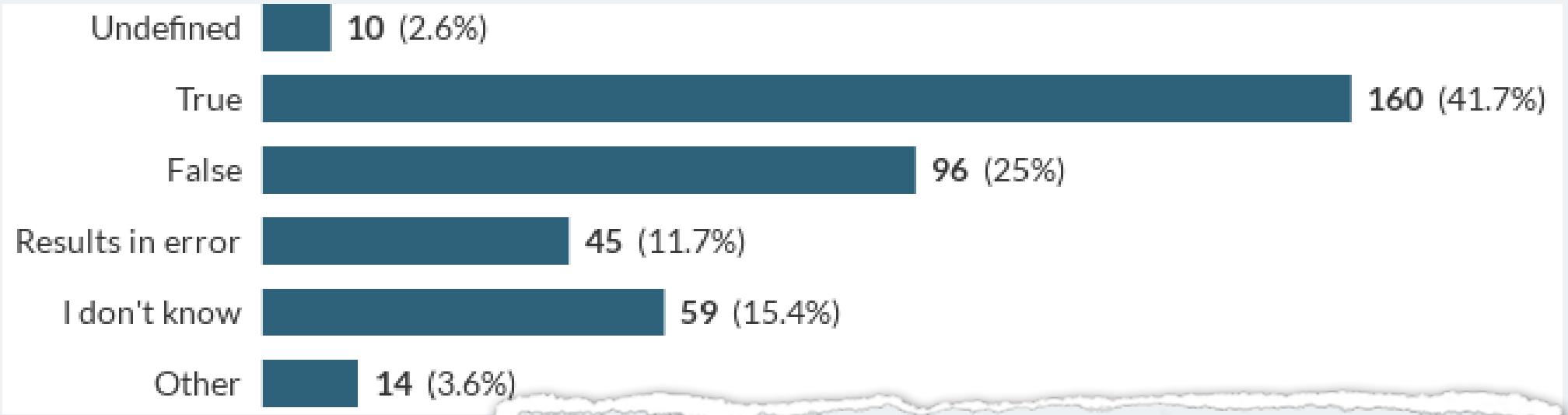
Player.showbalance


```
Box {
   show-balance = true,
}
Player {
   tracks = 573,
   genres = 11,
   show-balance = false
}
MusicBox inherit Box inherit Player {
    genres = 9,
}
```

MusicBox.showbalance

```
Box {
   show-balance = true,
}
Player {
   tracks = 573,
   genres = 11,
   show-balance = false
}
MusicBox inherit Box inherit Player {
    genres = 9,
}
```

MusicBox.showbalance

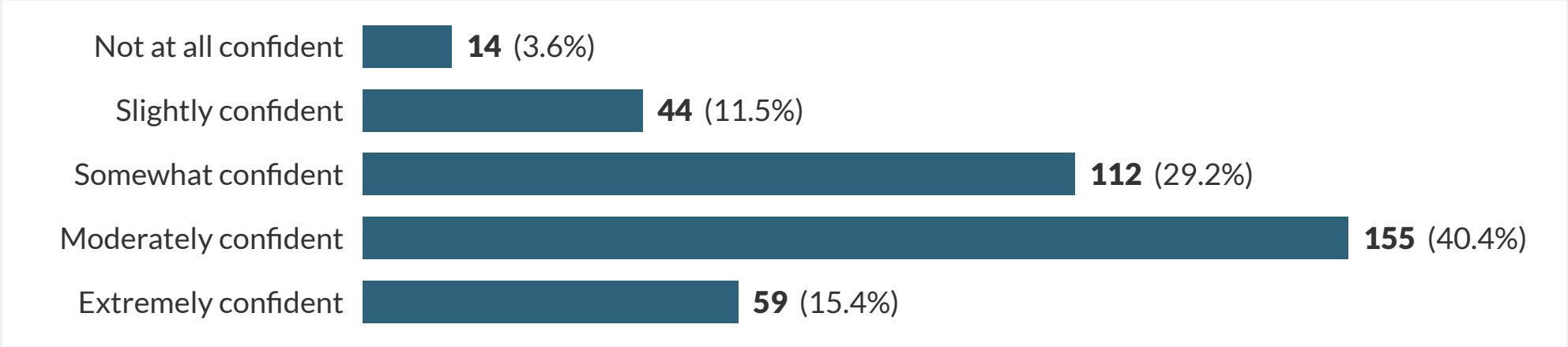



```
Box {
   show-balance = true,
}
Player {
   tracks = 573,
   genres = 11,
   show-balance = false
}
MusicBox inherit Box inherit Player {
    genres = 9,
}
```

Confidence

But …

- How confident were you? ….



Some Comments on Inheritance

- *“Inheritance is useful when used in high level programming languages. Inheritance in a CfgMgmt System often leads to complexity and should therefor be avoided. ”*
- *“Multiple inheritance was a mistake.“*
- *“it is strange that _instances_ can _inherit_. I would totally grok 'extends'. 'inherit' is more for classes.”*
- *“The inheritance question was not difficult at all, probably because I have been educated and trained in object-oriented design and programming.”*

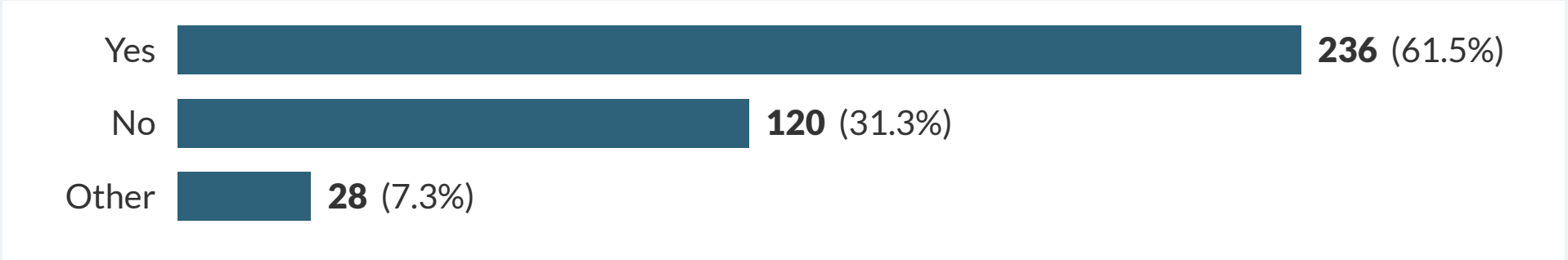
References ...

References & Scoping

References are a common source of confusion

- "Static" vs "Dynamic" references
  - Puppet has recently changed from largely dynamic to "mostly" static
- Interaction with other features (e.g. inheritance)
- Differences between "Resources", "Classes" and "Variables"
- Scope is also confusing
  - We didn't have the time to probe this explicitly

Do you know how static & dynamic scoping work?

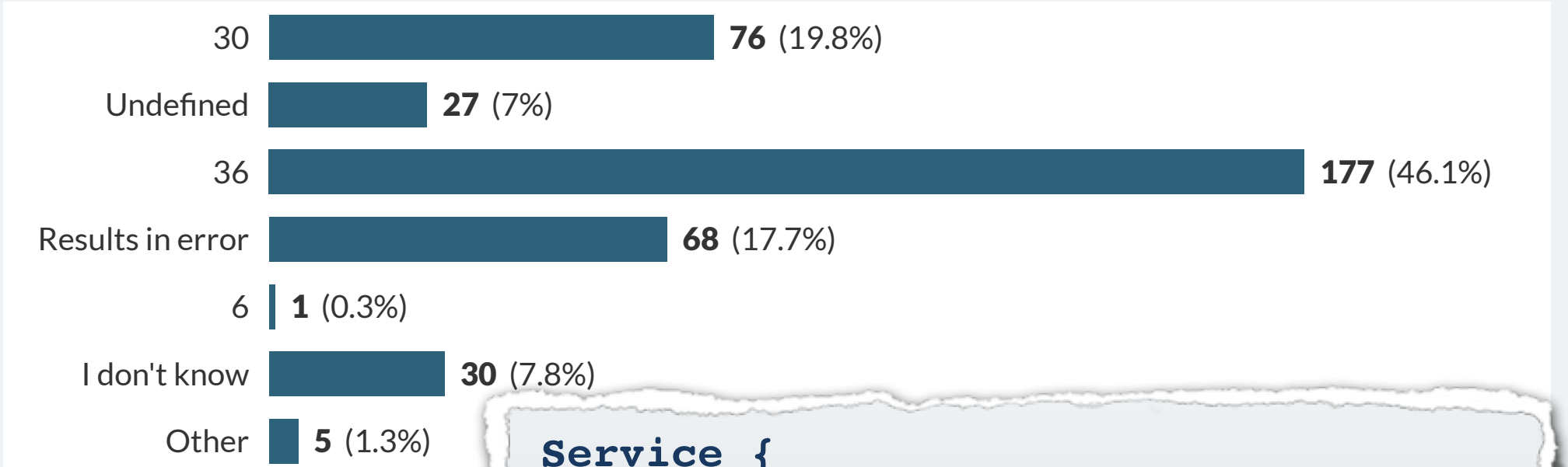


MyService.Client.port

```
Service {
   $port = 30,
   Client {
      port = $port,
   }
   Server {
      port = $port
   }
}
MyService inherit Service.Client {
   $port = 36
}
```

Unfortunate Typo!

MyService.Client.port



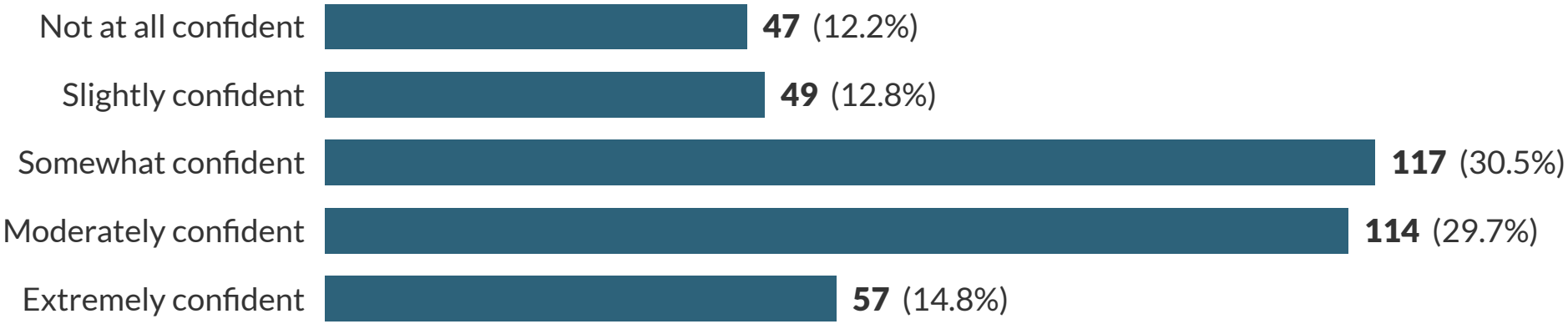

```
Service {
  $port = 30,
  Client {
    port = $port,
  }
  Server {
    port = $port
  }
}
MyService inherit Service.Client {
  $port = 36
}
```

Unfortunate Typo!

Confidence

But …

- How confident were you? ….



- Although this is not such a good indicator because of the typo

Some Observations (Conclusions?)

SysAdmins work with a huge variety of languages

- The intuition that they bring to new languages varies significantly
- They sometimes have a confidence in their interpretation which is not always justified
- It seems likely that this is responsible for at least some of the common configuration errors
- Existing languages often have very complex semantics
- Sysadmin programming experience can be very varied …

- *"Seems like an odd language since it appears to be defining types and variables. I suppose they could be class vars, but still seems odd."*
- *"Data structures should be immutable for least surprise."*
- *"It wasn't clear to me the purpose of the inherit syntax. Is that applying the Message formating to the Post?"*
- *"Order of properties should not matter. Order of variables should."*
- *"This sounds like another way of saying declarative vs imperative." (on static/dynamic scope)*

Some Implications ?

Some pointers for new language design ...

- Consider the usability for the target users
- Keep the semantics consistent and simple
- Explicitly use new/different syntax and terminology for features which behave differently from exiting languages (e.g. “inheritance”)
- Avoid dependence on properties such as “order” whose significance is frequently misinterpreted
- Have implementations which support simulated experimentation

Further Work

Some more analysis of the existing data would be interesting

- A search for significant correlations - e.g.
  - between previous language experience and various answers
  - between confidence factors and consistency in answers
- A more in-depth qualitative analysis of the free-text comments
  - many off these are very interesting

Experimental languages

- Incorporating some of the lessons from these investigations into an experimental language which can then be evaluated

This work is a good basis for a Phd thesis

- We have a proposal!

The End

References

- Adele Mikoliunaite
  - Usability of System Configuration Languages, MSc dissertation 2016
- Paul Anderson
  - The L3 Configuration Language
  - Assorted other talks & papers

http://homepages.inf.ed.ac.uk/dcspaul