The SemVer Talk 1.1.0

Web Directions Code 2016

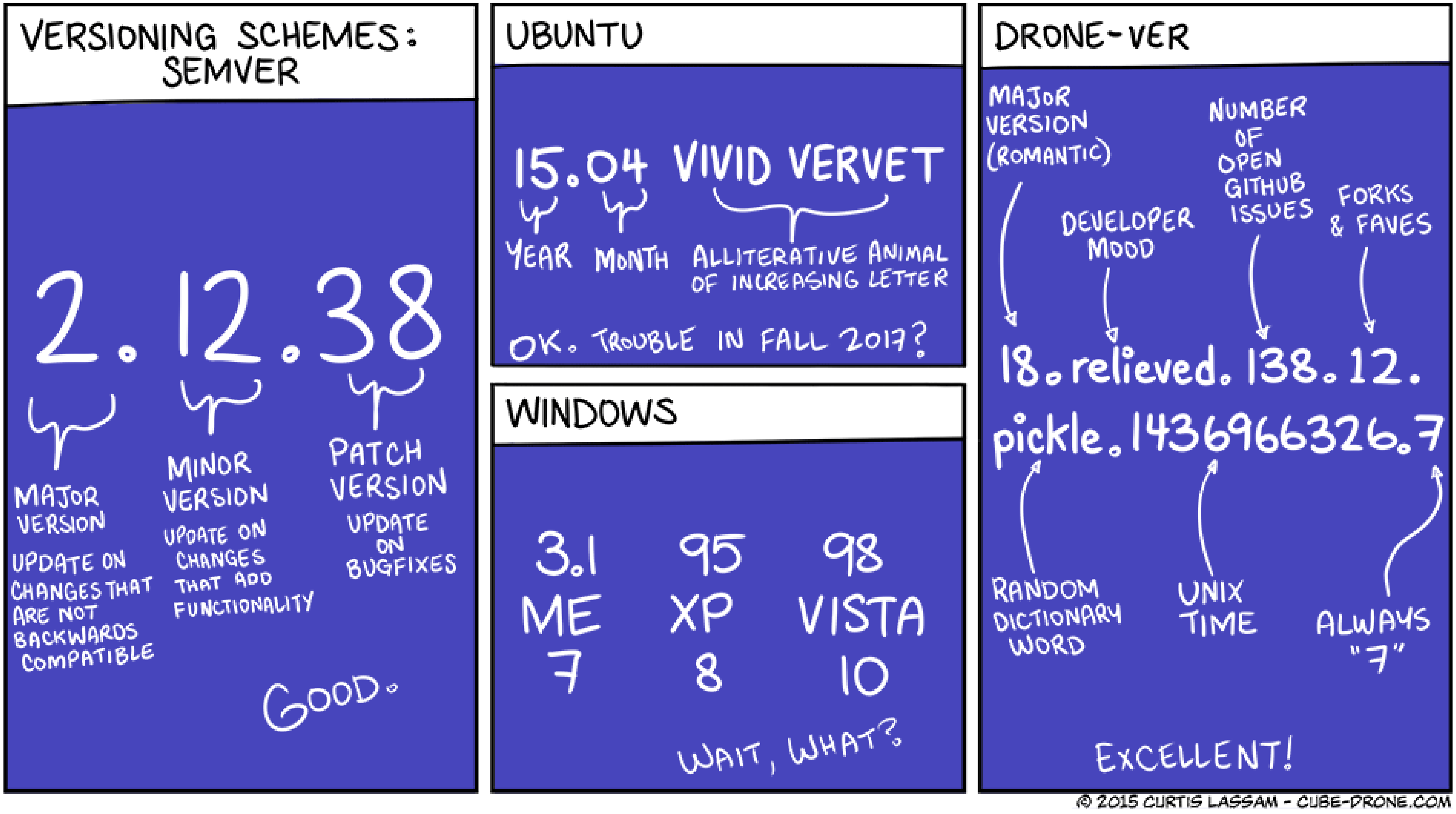
Ben Buchanan, @200okpublic

command-line.net

Versions everywhere

```
{
  "dependencies": {
    "express": "~4.13.3",
    "grunt-cli": "~0.1.13",
    "mustache": "~2.2.1",
    "socket.io": "~1.3.7"
  }
}
```

1.nervous.0.0.kumquat.1469273302.7



Version schemes

- Sequential
- Date of release
- Degree of change
- Degree of compatibility
- Random ^%#@

ReadMyBlogVer

Where the versions mean nothing,
you just have to read their blog.

ReadMyBlogVer doesn't scale

reveal.js uses **484 NPM packages**.

484 * 2 minutes = ~16 hours.

People don't read your blog

Your API is a tiny piece
of a much larger experience.

Enter Semantic Versioning...

SemVer communicates changes to a public API.

semver.org

SemVer is the most popular version scheme in web development.

Most popular?

Preferred in most current dev stacks.

91% of Hashnoders surveyed* use it.

* survey was not in any way scientific.

Semantic

adjective

Relating to meaning in language or logic.

SemVer is...

A **degree of compatibility** scheme.

SemVer describes changes to the API.

Anchored to `1.0.0`

`0.x.y` = early development
`1.0.0` = first public API
`2.0.0` = first breaking change

SemVer is not...

Based on the size, number, or general vibe of the changes.

SemVer is not...

Guessing if your code works,
or estimating your upgrade work.

1.2.3

1 = major

2 = minor

3 = patch

`1.2.3`

`1 = major`

2 = minor

3 = patch

1.$\boxed{2}$.3

1 = major

$\boxed{\texttt{2 = minor}}$

3 = patch

1.2.$\boxed{3}$

1 = major

2 = minor

$\boxed{\texttt{3 = patch}}$

Pre-release & Build

`1.2.3-beta.1`

`1.2.3-beta.1+001`

Precedence

`1.2.3`

↑

`1.2.3-beta.1`

~~`1.2.3-beta.1+001`~~ (builds ignored)

X.Y.Z

Three numbers, not one.

Each increments sequentially.

Each increments indefinitely.

`1.9.0`

Can but **does not have to** increment to `2.0.0`

`1.9.0` can increment to

`2.0.0`

`1.10.0`

`1.9.1`

What do the terms mean?

Major = breaking changes

Minor = new features

Patch = bug fixes

Breaking changes

Code changes which are
not backwards-compatible
are called breaking changes.

Imagine this API:

```
// returns a small black coffee
COFFEE.gimme('large')
```

Wait, `large` returns `small`?

Next release

```
// returns a large black coffee
COFFEE.gimme('large')
```

Fixed! That's a patch: `1.0.1`

Next release

```
// adds types of coffee
COFFEE.gimme('large', 'latte')
```

New feature! That's a minor: `1.1.0`

Next release

```
// has a more extensible format
COFFEE.gimme({
  'size':'large',
  'type':'latte'
})
```

```
// but old calls no longer work
COFFEE.gimme('large', 'latte')
```

That's a breaking change: `2.0.0`

Shorthand

SemVer compresses information.

We judge risk every day

```
Update available 1.7.7 → 1.7.9
Run npm i -g bower to update
```

How risky is this upgrade?

`1.0.0` to `2.0.0` = dangerous
`1.0.0` to `1.1.0` = safe
`1.0.0` to `1.0.1` = safe

What will happen when I upgrade?

`1.0.0` to `2.0.0` = **things will break**

`1.0.0` to `1.1.0` = **you can use something new**

`1.0.0` to `1.0.1` = **something was fixed**

What do I need to read?

`1.0.0` to `2.0.0` = **upgrade guide, definitely**

`1.0.0` to `1.1.0` = **release notes, maybe**

`1.0.0` to `1.0.1` = **nothing**

Non-code SemVer

Versions can help with design, copy...

Does this look familiar?

```
website-new.psd
website-new_new.psd
website-new_new-blue.psd
website-new_new-with-bigger-logo.psd
website-new-final.psd
website-new-final-fixed.psd
```

Better!

```
website-0.1.psd
website-0.2.psd
website-1.0.psd
website-1.0.1.psd
website-1.1.0.psd
website-2.0.psd
```

So we’ve solved everything?

Excellent! **Job done**.

Many people don't follow SemVer.

:(

Common breaches

- Breaking changes in minor or patch
- New features in a patch
- Skipping versions
- Modifying a deployed package
- Permanent Zero

Protect yourself

Lock noncompliant dependencies.

Avoid confusing auto-upgrade syntax.

Use shrink wrap.

Auto upgrade in NPM

`*` auto upgrades major
`^1.0.1` auto upgrades minor
`~1.0.1` auto upgrades patches

x is easier to read

`x` auto upgrades major
`1.x` auto upgrades minor
`1.0.x` auto upgrades patches

Shrink wrap

Include resolved dependency tree details when you tag your project.

Why don't people use SemVer?

“Too hard to make changes”

“Not followed, why bother”

Too hard to make changes?

Put the user first.

Plan your API.

Use deprecation.

Backwards compatibility

+

Deprecation

=

Less pain for users

Revisiting our coffee API

```
// returns coffee
COFFEE.gimme({ 'size':'large' })
```

```
// breaks
COFFEE.gimme('large')
```

Required a `2.0.0` release.

Option: accept both

```
// returns coffee
COFFEE.gimme({ 'size':'large' })
```

```
// returns coffee + warning
COFFEE.gimme('large')
```

Minor release: `1.2.0`

Option: different name

```
// returns coffee
COFFEE.giveMe({ 'size':'large' })
```

```
// returns coffee + warning
COFFEE.gimme('large')
```

Minor release: `1.2.0`

Deprecation

Gives you freedom

Gives users time to update

Common pattern

`1.2.0` feature deprecated
`2.0.0` removed from docs
`3.0.0` removed from code

Pre-releases

+

API planning

=

Less pain for everyone

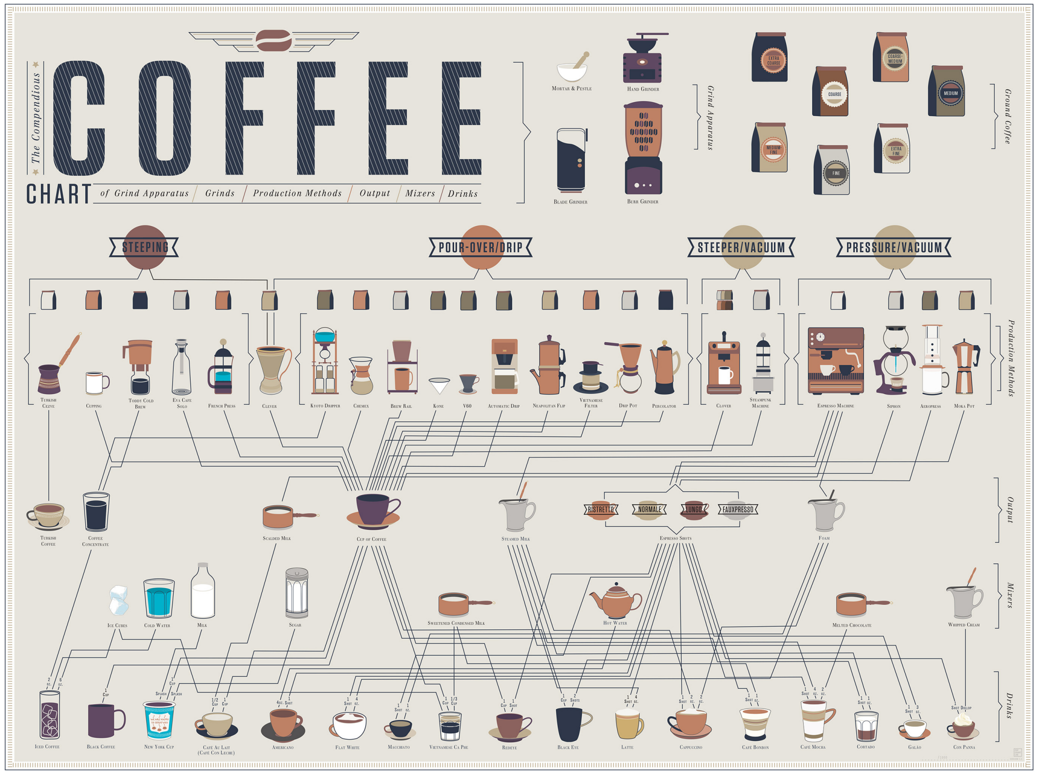
Pre-release feedback

```
// v1.0.0-beta.1
COFFEE.gimme('large');
```

```
// v1.0.0-beta.2
COFFEE.gimme('large','latte','skim');
```

```
// v1.0.0
COFFEE.gimme({
  'type': 'latte',
  'size': 'large',
  'milk': 'fullcream'
})
```

API planning (know the domain)



graphic: popchartlab.com

But...

If we bump version numbers,
people will think the API
is unstable!

Hauptversionsnummernerhöhungsangst

noun

Fear of increasing the major version number

Be judicious

`50.1.1` is fine

`1.50.1` is great

`1.1.50` is not so great

“Not followed, why bother”

Lack of compliance
does not invalidate standards.

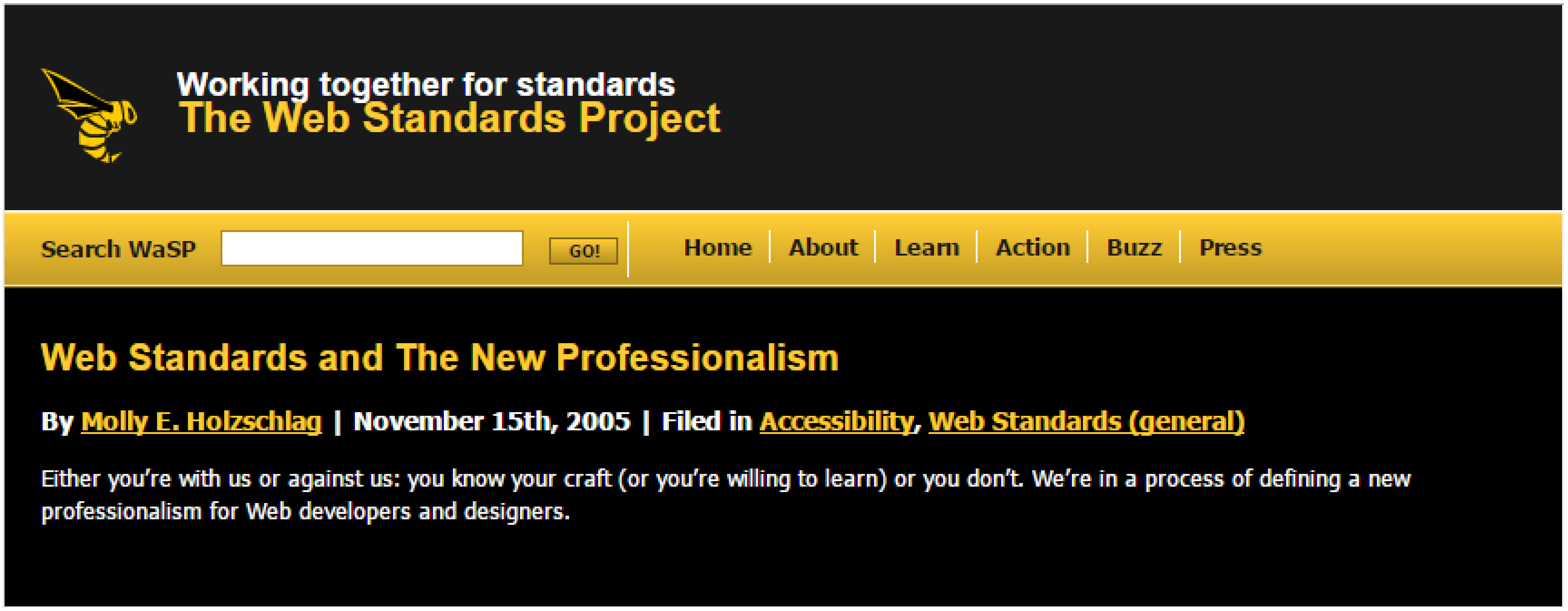
Advocate.

This is not just a job, it's a **craft**.

Professionalism

We must earn titles like 'Engineer'
by displaying *engineering rigour*

This is not a new call



Web Standards and The New Professionalism

By Molly E. Holzschlag | November 15th, 2005 | Filed in Accessibility, Web Standards (general)

Either you're with us or against us: you know your craft (or you're willing to learn) or you don't. We're in a process of defining a new professionalism for Web developers and designers.

Professionalism

API stability.

Predictability.

Quality.

Professionalism

SemVer is a small piece.

Use it.

Demand it.

1.2.3

1 = broken

2 = improved

3 = fixed

semver.org

1.2.3

1 = broken

2 = improved

3 = fixed

semver.org

Thank you. @200okpublic, command-line.net