Talking about software development

We are *software developers*, but talk surprisingly little about software development.

I want us to be more *concerned with and actively talk* about software development.

Talking about software development

We are *software developers*, but talk surprisingly little about software development.

I want us to be more *concerned with and actively talk* about software development.

I notice that we don't do it like they do it.

So why care about *our history*?

The Art of Unix Programming

Those who do not understand Unix are condemned to reinvent it, poorly. — Henry Spencer

The Art of Unix Programming

Those who do not understand Unix are condemned to reinvent it, poorly. — Henry Spencer

This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface. — Eric S Raymond

The Art of Unix Programming

Those who do not understand Unix are condemned to reinvent it, poorly. — Henry Spencer

This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface. — Eric S Raymond

1. Rule of Modularity: Write simple parts connected by clean interfaces.
2. Rule of Clarity: Clarity is better than cleverness.
3. Rule of Composition: Design programs to be connected to other programs.
4. ...

Question: How can we live up to these rules?

Version numbers

Version numbering:

Version numbers

Version numbering:

- 25.3
- 8.0.0960
- 4.13.7
- 4.13.b1
- 4.0.2
- 2.60.3
- 3.0
- 1.13.3
- 1.13.0rc2
- 0.20.3
- 1.8.3.1

Version numbers

Version numbering:

- 25.3
- 8.0.0960
- 4.13.7
- 4.13.b1
- 4.0.2
- 2.60.3
- 3.0
- 1.13.3
- 1.13.0rc2
- 0.20.3
- 1.8.3.1

Version numbering:

- 2017.10
- 2017.10

Software release cycle

Why the nagging about version numbering?

Well it's not about giving software a name consisting of numbers and dots!

These are the facts of the case and they are undisputed.

(Skip if no intention of making reusable code)

The anatomy of the version number

Version numbering: Major.Minor.Micro (Micro = Patch)

- The major number should be increased whenever the API changes in an incompatible way.

The anatomy of the version number

Version numbering: Major.Minor.Micro (Micro = Patch)

- The major number should be increased whenever the API changes in an incompatible way.
- The minor number should be increased whenever the API changes in a compatible way.

The anatomy of the version number

Version numbering: Major.Minor.Micro (Micro = Patch)

- The major number should be increased whenever the API changes in an incompatible way.
- The minor number should be increased whenever the API changes in a compatible way.
- The micro number should be increased whenever the implementation changes, while the API does not.

The anatomy of the version number, cont'd

Pre-alpha $\rightarrow$ alpha $\rightarrow$ beta $\rightarrow$ release candidate $\rightarrow$ gold

The anatomy of the version number, cont'd

Pre-alpha $\rightarrow$ alpha $\rightarrow$ beta $\rightarrow$ release candidate $\rightarrow$ gold

- If Micro contains a letter, a=alpha, b=beta, rc=release candidate
- beta is intended stable, but may change
- rc is feature frozen

The anatomy of the version number, cont'd

Pre-alpha → alpha → beta → release candidate → gold

- If Micro contains a letter, a=alpha, b=beta, rc=release candidate
- beta is intended stable, but may change
- rc is feature frozen

Examples:

- 2.3.pre-alpha1
- 2.3.pre-alpha2
- 2.3.a1
- 2.3.a2
- 2.3.b1
- 2.3.rc1
- 2.3.rc2

Why the obsession with version numbers?

Because better men than we paved the road. They wrote Unix, GNU coreutils, Linux, all the software that we use and adore. They found a way.

> *The first and most important quality of modular code is encapsulation.*
> *They communicate using APIs—narrow, well-defined sets of procedure calls and data structures. — Eric S. Raymond*

A version is defined by its API, its functionality

Once a function goes in, it must stay in until next major version!

Consequence of software development

Mantra: Bad code can be deleted, bad API is legacy

- API = functionality
- code = machinery

Code is something that coincidentally makes the API work.

Consequence of software development

Mantra: Bad code can be deleted, bad API is legacy

- API = functionality
- code = machinery

Code is something that coincidentally makes the API work.

> *Due to the required backwards compatibility there is certainly a code-complexity price related to this.*
> — *Joakim*

- PEP-8

- PEP-8
- YAGNI (You ain't gonna need it)

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write
- External facing APIs are where design up-front matters!
  - Changing API is painful
  - creating backwards incompatibility is horrible
  - design carefully! (But keep simple things simple ...)

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write
- External facing APIs are where design up-front matters!
  - Changing API is painful
  - creating backwards incompatibility is horrible
  - design carefully! (But keep simple things simple ...)
- If a function or method is more than 30LOC, break it up!

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write
- External facing APIs are where design up-front matters!
  - Changing API is painful
  - creating backwards incompatibility is horrible
  - design carefully! (But keep simple things simple ...)
- If a function or method is more than 30LOC, break it up!
- Refactor — keep in mind that programming is about abstractions (and we discover new abstractions as we go along)

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write
- External facing APIs are where design up-front matters!
  - Changing API is painful
  - creating backwards incompatibility is horrible
  - design carefully! (But keep simple things simple ...)
- If a function or method is more than 30LOC, break it up!
- Refactor — keep in mind that programming is about abstractions (and we discover new abstractions as we go along)
- Always see your test fail once! (Here's a question: Can we have a robot making random changes in code and see if tests fail?)

- PEP-8
- YAGNI (You ain't gonna need it)
- Test *your* code, not others'
- API: Simple things should be simple, complex things should be possible (*interface usage errors belongs to the interface designer — Scott Meyers*)
- Code is the enemy — write less, delete, don't write
- External facing APIs are where design up-front matters!
  - Changing API is painful
  - creating backwards incompatibility is horrible
  - design carefully! (But keep simple things simple ...)
- If a function or method is more than 30LOC, break it up!
- Refactor — keep in mind that programming is about abstractions (and we discover new abstractions as we go along)
- Always see your test fail once! (Here's a question: Can we have a robot making random changes in code and see if tests fail?)
- Continuously address technical debt.

Code is mass

Good PR: +14, -521 — Bad PR: +3123, -1

> *One of my most productive days was throwing away 1000 lines of code. — Ken Thompson*

Code is mass and has a weight. And somebody is going to carry it.

Code is mass

Good PR: +14, -521 — Bad PR: +3123, -1

> *One of my most productive days was throwing away 1000 lines of code. — Ken Thompson*

Code is mass and has a weight. And somebody is going to carry it.

If we have the choice between implementing a feature, and using an existing library, the pros and cons are:

- implement it yourself, you (or rather your team) carries the weight
- use somebody else's implementation, they carry the weight, you only carry the load of using that library (which may or may not be expensive)

Code is mass cont'd

If all else is equal, *less code* is better than *more code*. Fewer lines equals lower weight. (Not invented here)

Code is mass cont'd

If all else is equal, *less code* is better than *more code*. Fewer lines equals lower weight. (Not invented here)

We should think in terms of code as being something that's just there for the API to work.

Code is mass cont'd

If all else is equal, *less code* is better than *more code*. Fewer lines equals lower weight. (Not invented here)

We should think in terms of code as being something that's just there for the API to work.

> *Thou shalt study thy libraries and strive not to re-invent them without cause, that thy code may be short and readable and thy days pleasant and productive.*
> *— Henry Spencer's "The Ten Commandments for C Programmers"*