How NOT to write software

Common patterns that ruins everything

Michal Hrušecký
Michal@Hrusecky.net

Who am I?

Why should you trust me?

- hobby programmer for about 20+ years
  - at least partly professional for 5+ years
- package maintainer for 10+ years

You shouldn’t

Years of experience doesn’t mean squat.

Listen, evaluate and decide what sounds reasonable.

Package maintainer?

Why is maintaining package anyhow relevant to software development…

And why you should do it if you are just starting your career…

- you don’t have time to learn everything
  - you have to fix problems with local knowledge
  - you’ll learn to debug alien code fast
- you’ll learn that everybody makes mistakes
  - even well known projects contain beginner type mistakes
- you’ll learn that you are actually not that bad developer
  - now that you have seen how everything is hacked together

Try to solve the real problems

And start with the one you set out to solve.

If writing mail client make sure it can send mails.

If editor, that it can edit files.

Scripting language and plugin system can wait.

Make it do something useful first.

No, your TODO list application doesn’t need 3D Engine first.

Is "Agile" still a thing?

Plenty of competing ways how to do it.

Good side:

- managers approved way to release early, release often

Bad side:

- mostly excuse for managers to have more meetings
- excuse for management not to have specification of the problem
- trainings available

Reinventing the wheel

There is plenty of libraries ready to solve your problems

Use them wisely. There is a fine line.

- Don’t reinvent everything
- Don’t drag in some zombies
- Try to avoid dragging in huge libraries to do simple stuff

When in Rome

…do as the Romans do

- respect the style of your language
- don’t try to bend it your way
- use the language best practices
- use the default build system
    - autotools, cmake for C/C++
    - setuptools for python
    - …

The right amount of knowledge

Do you know…

- every aspect of your language?
- every aspect of your OS?
- all possible side-effects of some operation?
- how to make compiler do your bidding?

Good for you, but **do not use it**.

Stick to the well known facts.

Using everything makes you a **jerk**, not a good programmer.

Use the right language

Some languages are better suited, some you should avoid.

The wrong language is:

awesome language you just started learning

if you work in a team where nobody knows it

~~Java most of the time~~

The right language is:

- one you are comfortable with
- your peers are comfortable with as well
- has support for what you want to do

Optimize the right stuff

You want to make sure that the part that most time is spend in can be done as fast as possible.

You are going to spend the most time maintaining the code. At least in the beginning.

And whe somebody tries to use it, you are gonna throw away most of it anyway.

Document the right stuff

Documenting the obvious is meaningless.

Documenting the complex part sometimes as well.

```
// If it was hard to write, it should be hard to read.

// Here be dragons. Thou art forewarned

// I'm really proud of the following lines.
```

Try to write code that is so clear that it doesn’t need documentation.

Few quotes to end with

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.