Refactoring

“Keep it clean!”

Software Construction?

Draw blueprints	Plant things according to initial plan and conditions
Dig, pour, build	Some thrive, a few wither
Live happily ever after, homeowners maintain when problems arise	Move things as they crowd each other, and according to environmental conditions. Prune/split Monitor health, make adjustments.

When to refactor?

- Reduce violations of our principles
  - Make more “able”
  - Improve design
- Knowledge improves
  - Discover new axes of change
  - Better understanding of the problem
  - Improve design
- Performance

When to refactor?

When it's "wrong."

When current structures are terrible, or have become terrible.

Software won't be perfect, but we can try to keep things manageable.

What's our only constant?

But, but…

- That sounds like more work
- Don’t have time

- Do we let growths fester?
- Conjoined twins can just deal?

Do it now!

- Keep a handle on technological debt: You must pay it sometime.
- There won't be more time in the future.

How?

Very Carefully!

(Or you'll make things worse)

Guidelines

- Don’t add functionality while refactoring
- Have good tests first. Run them often to make sure you still pass.
- Baby steps!
- Many small modifications that result in large-scale change

“Refactor early, refactor often.”

- From “Pragmatic Programmer.”
- More often = fewer entire weekends
- Nip problems as you see them arise.

Conclusion

- If it's going to bite you, fix it, and everything that depends on it, which should be minimal, if we're doing it right. If we're not, fix *that*.

- Don't live with terrible design, if you can avoid it. The answer to "how do we do that?" may involve "differently than we are now."

- Manage the pain! You will be charged interest on all tech debts, so get it over with now.