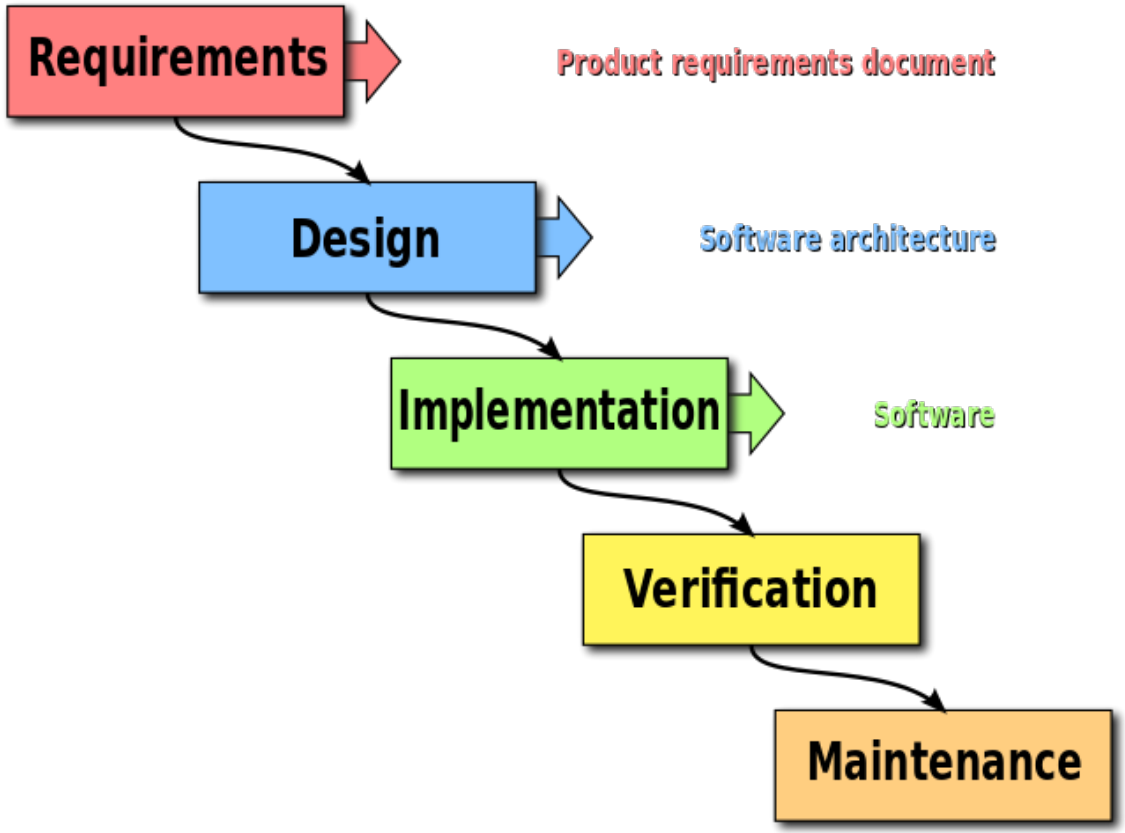
When do we design, and what do we make?

FIT2099: SEMESTER 1 2018

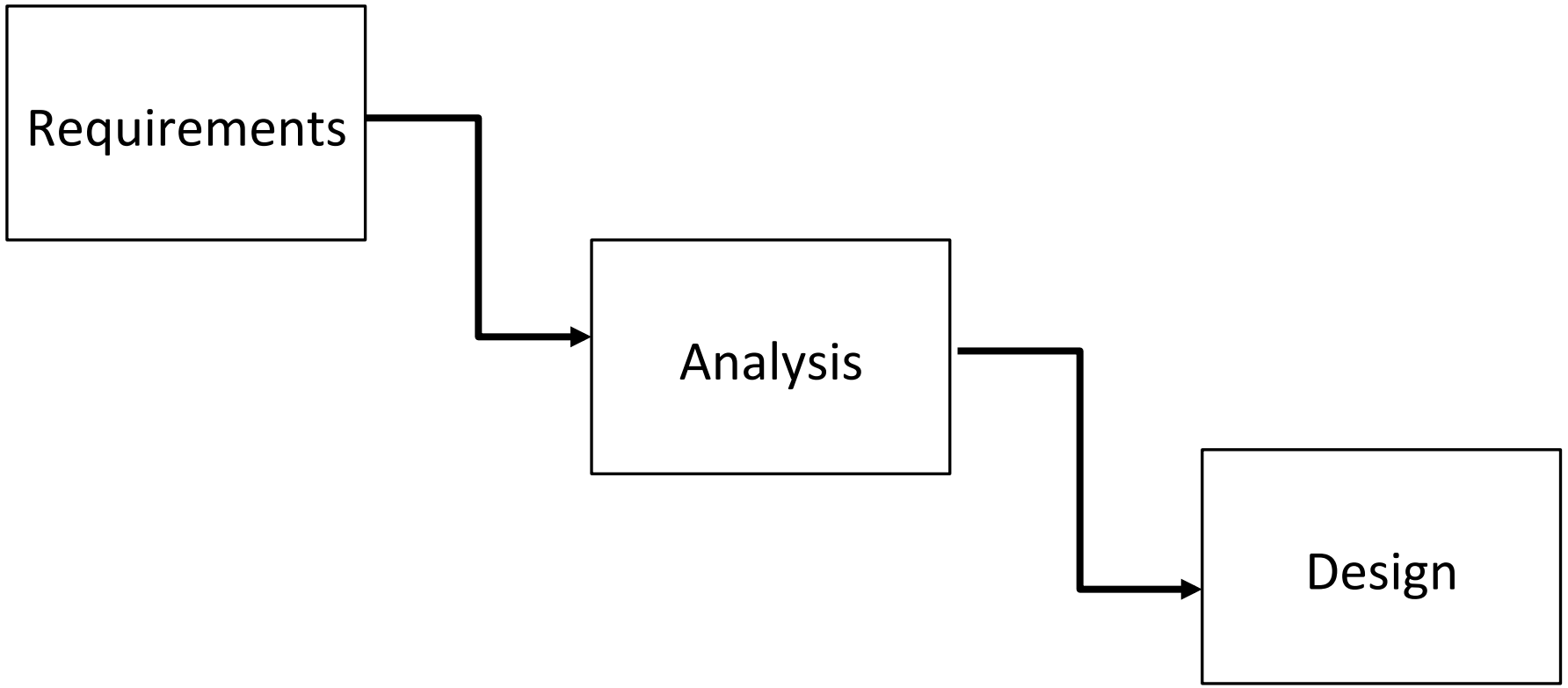
Where were we?

- Talked about *notations for capturing design decisions*
- But didn't explain *how/where/why*

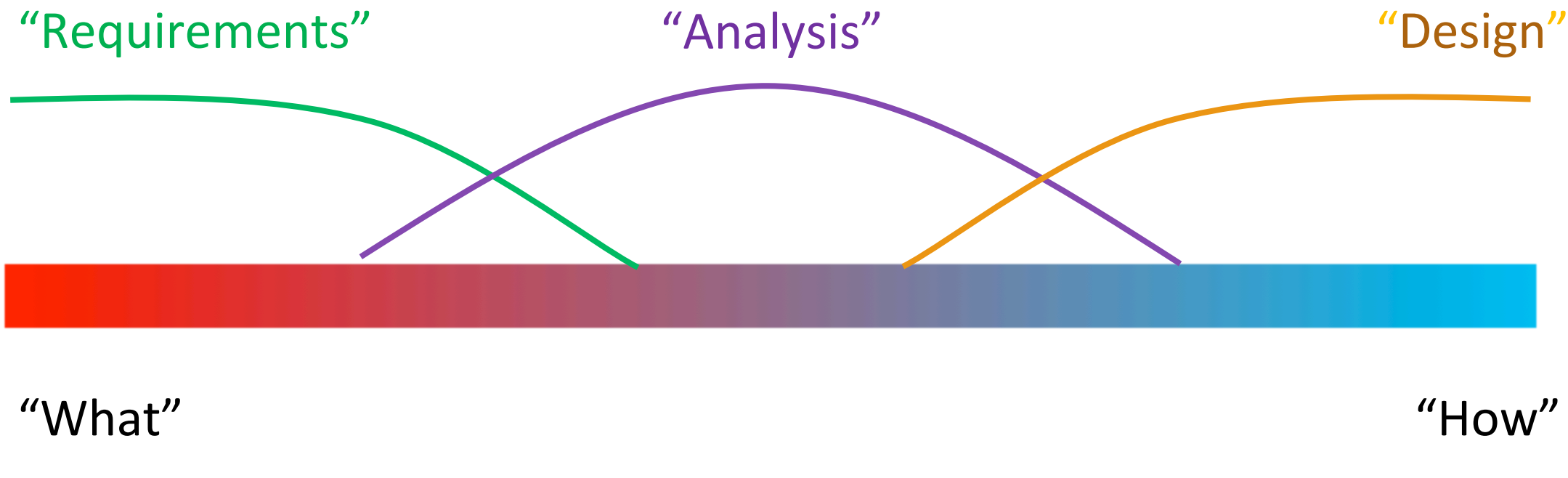
Recap: Waterfall



A slight refinement



In practice



In practice…Just In Time, Just Enough design*

Toyota production line.
Source: https://www.flickr.com/photos/toyotauk/4711057629

* Phrase from Bob Hartman, http://agileforall.com/new-to-agile-remember-one-thing-just-enough-just-in

Typical times for conscious design

- Project inception
  - Architectural design – big decisions!
- Before implementing something complex or risky
- Refactoring existing code
  - Design was bad from the start
  - Design started out good, is no longer good.

```
ivate static double FindInterestRate(int operationYear,
                                     Dictionary<int, double>
arToInterestRates) //where 0 is the first year

  if (operationYear < 0)
      return 0;
  else
  {
      for(int i = 1; i < yearToInterestRates.Count; i++)
      {
          if (operationYear < yearToInterestRates.ElementAt(i).Key - 1)
              return yearToInterestRates.ElementAt(i - 1).Value;
      }
      return yearToInterestRates.Last().Value;
  }
```

Source: The Daily WTF http://thedailywtf.com/articles/dictionary-definition-of-a-loop

Technical debt

You have a piece of functionality that you need to add to your system. You see two ways to do it, one is quick to do but is messy - you are sure that it will make further changes harder in the future. The other results in a cleaner design, but will take longer to put in place.

Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem. In this metaphor, doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt. Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice.

-- Martin Fowler

Technical debt…

- is inevitable in real systems
- happens because sometimes we need to get features out the door
- can be “repaid” by refactoring
- but you need to refactor to a better design

What are we aiming to produce when designing?

- A “good” design
- An understanding of that design *in the heads of stakeholders*
  - Stakeholder – anyone who is affected by a (design) decision
  - Designers (obviously!)
  - Implementers
  - Maintainers
  - Reviewers
  - Users
- Any documents/diagrams are in aid of the above

So…where do you start?

Let’s give an example…

In MADAM marking schemes, Tasks can have a comment field associated with them (you may have seen this if you have taken a unit that uses MADAM). We want to add “clippings” to comment fields – snippets of text that markers can paste into the comment field for that task by clicking a button Clippings can be created, edited, and deleted during marking. Lecturers should be able to share clippings with all the markers, but non-lecturer markers can also create clippings for their own use.

A preliminary domain model

A sequence diagram

A revised domain model

A revised sequence diagram

A more complete class diagram…

Observations..

- Both static and dynamic modelling required.
- Feedback between both
- Refine and make more concrete as we go − going from analysis to design.
- Feeds back into requirements (we can't implement Lecturer because system doesn't really have lecturer concept and that's a major new feature in itself).
- Sometimes we would actually write some prototype code along the way.
- Output:
  - Diagrams.
  - Understanding in designer's mind.
- Can present these to colleagues before implementation.

We've said this design is "good enough"

- How will we verify this?
- Coming up:
  - Things to look for in a good design (heuristics)
  - How to look at a design (reviews)