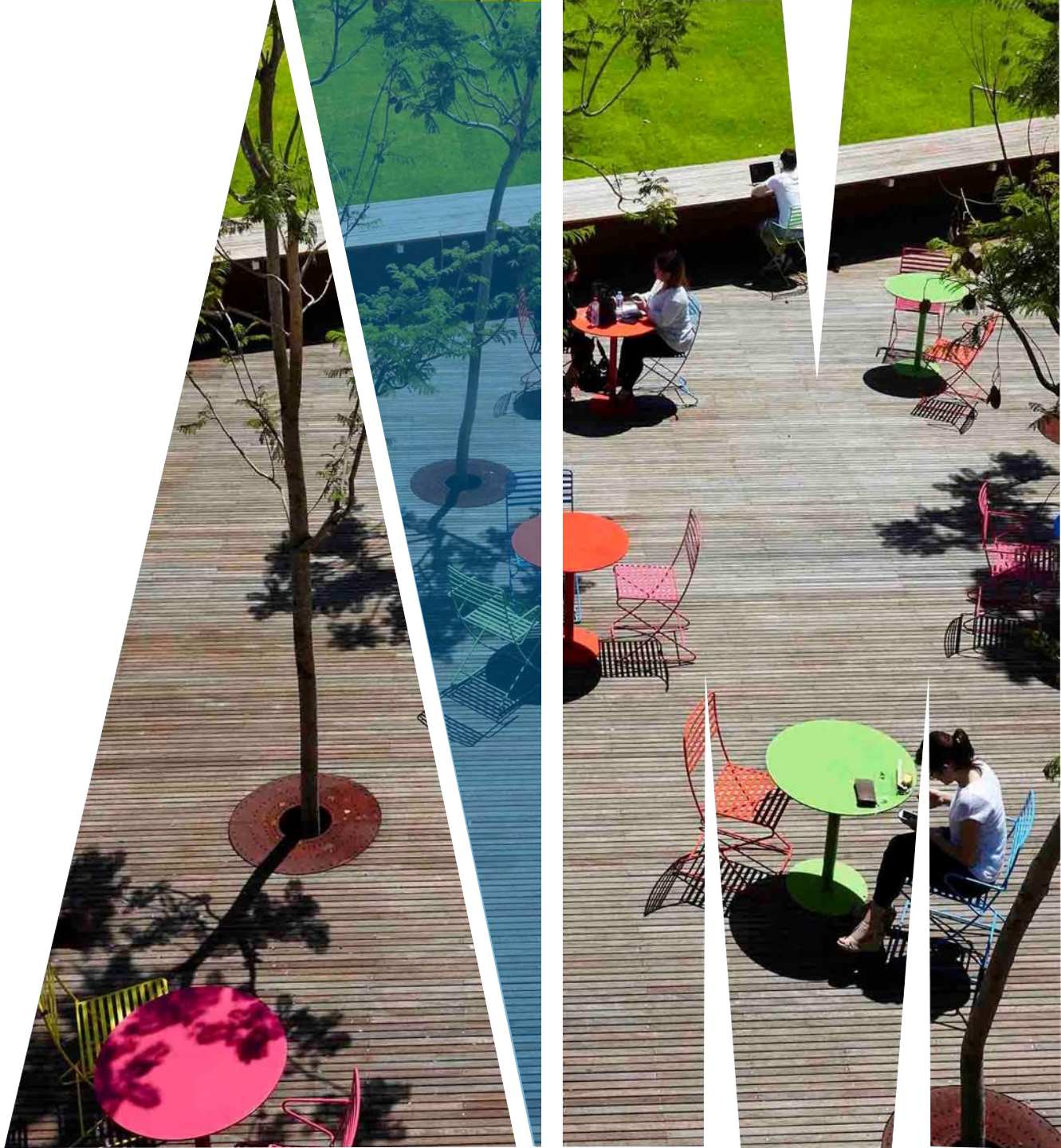
MONASH University

FIT2099 Object-Oriented Design and Implementation

Design:
how, when and why?



MONASH University

Outline

Reflection on the work you have done in this unit

The software development lifecycle

- Analysis and design

When to design

FIT2099

SO FAR

We have seen the components that make up object-oriented programs

- **packages, classes, methods, attributes**

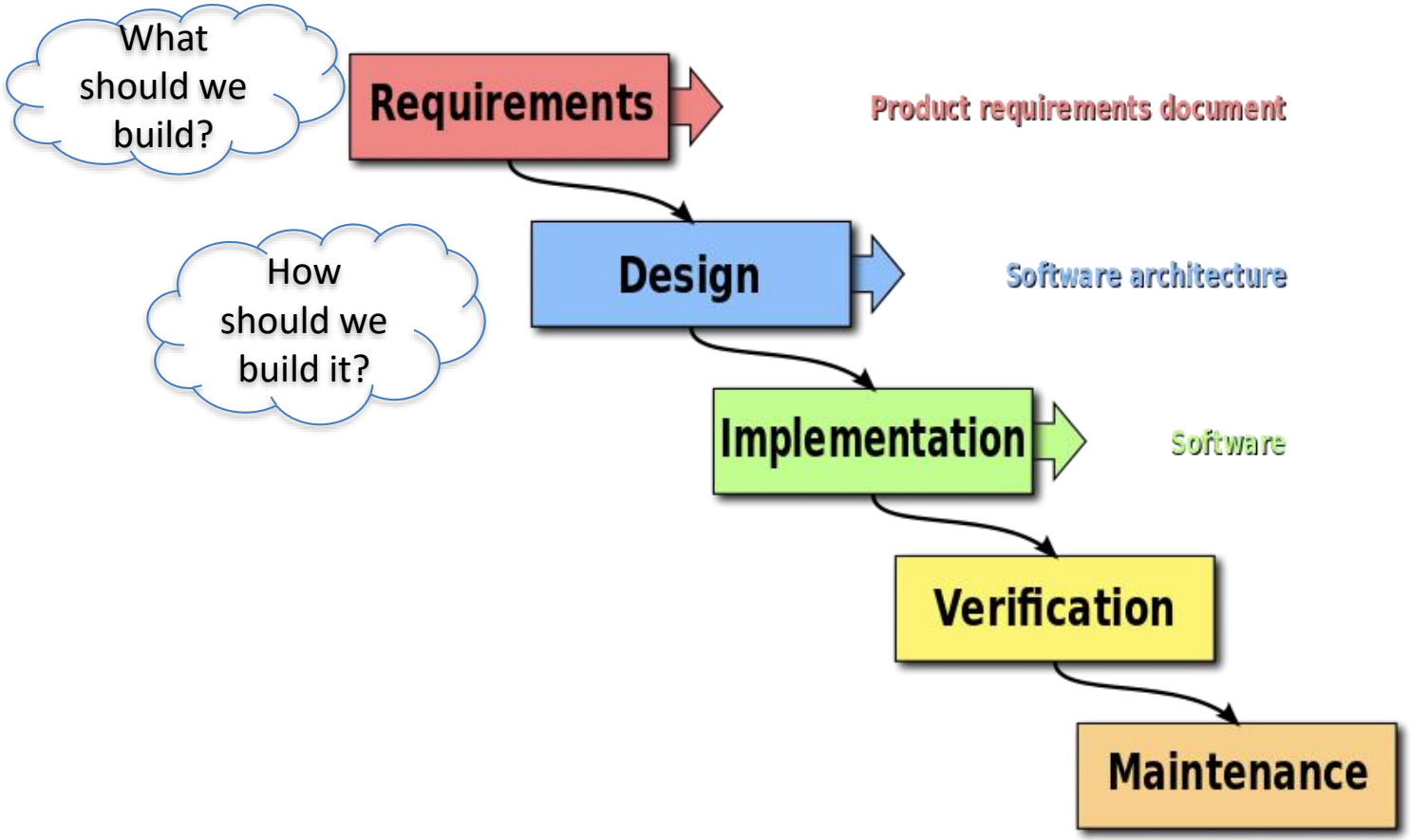
We have seen some notations for capturing object-oriented design decisions

- **class diagrams,** package diagrams, **sequence diagrams**, communication diagrams

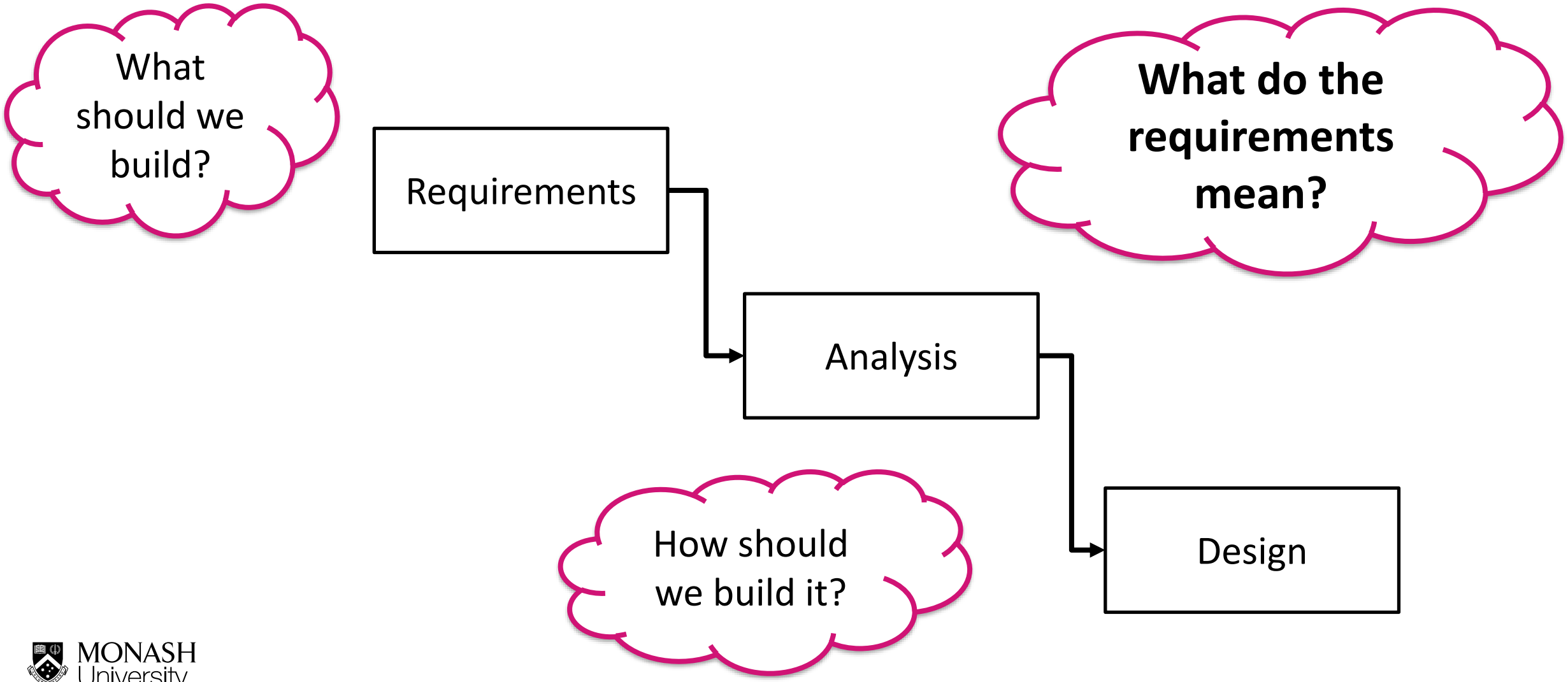
But we have not examined **where design fits in with the broader picture of software development**

RECAP

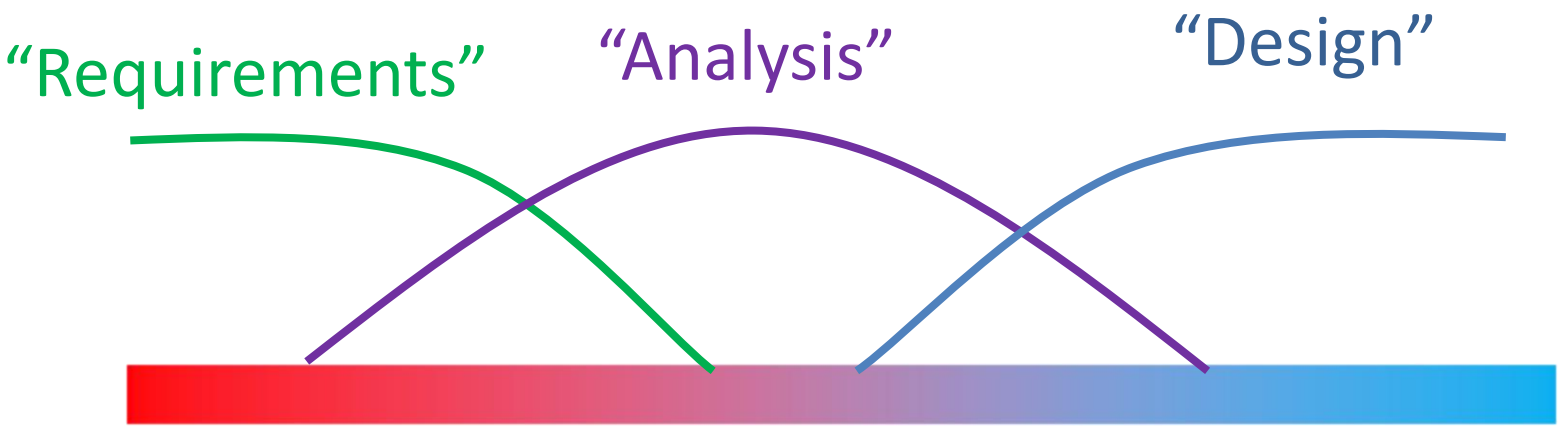
THE WATERFALL MODEL



A SLIGHTLY MORE REALISTIC REFINEMENT



DESIGN
IN PRACTICE



Showing these activities in neat, separate boxes is a bit **misleading**

In practice, there is usually a lot of overlap

- hard to distinguish between "requirement **elicitation**" and "requirement **analysis**"
- hard to distinguish between "**understanding** the domain" and "**modelling** the domain"

JUST IN TIME,
JUST ENOUGH DESIGN*

The vehicle manufacturer Toyota pioneered just-in-time delivery in the 1970s. Many of the concepts they introduced have found their way into software development, especially (but not only) in Lean and Agile development practices.

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

LEAN
PRINCIPLES

- ❑ Eliminate waste
- ❑ Amplify learning
- ❑ Decide as late as possible
- ❑ Deliver as fast as possible
- ❑ Empower the team
- ❑ Build integrity in
- ❑ Optimise the whole

LEAN
PRINCIPLES

- ❑ Eliminate waste
- ❑ Amplify learning
- ❑ Decide as late as possible
- ❑ Deliver as fast as possible
- ❑ Empower the team
- ❑ Build integrity in
- ❑ Optimise the whole

THE
LEAN PRINCIPLE #3

Decide as late as possible

The best decisions are based **on fact**, not speculation

The longer you wait to commit, **the more information** you have.

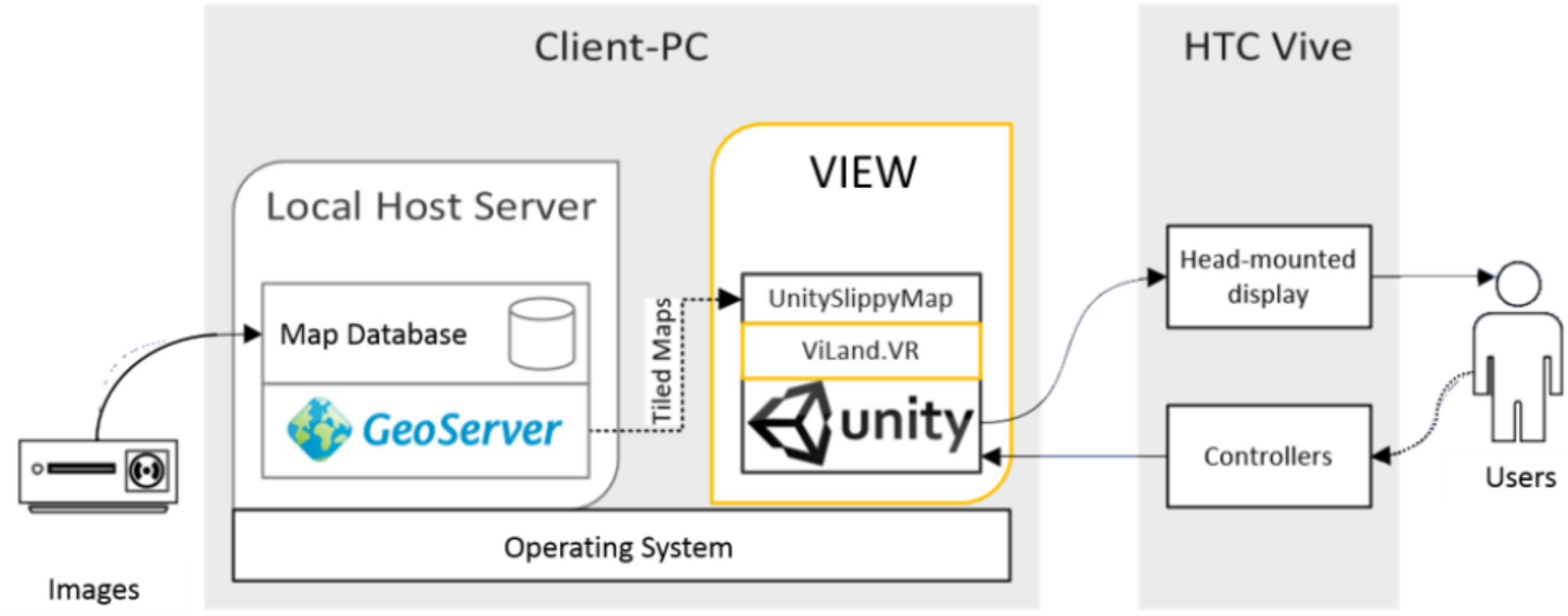
By contrast, **the earlier** you commit, the **less flexible** you can be.

TYPICAL TIMES FOR
FOR CONSCIOUS DESIGN

Project **inception**

- **architectural design** – big decisions!
- **hard/expensive to change**, so need to do it right

EXAMPLE ARCHITECTURAL DESIGN OF A VR SYSTEM



DOI: 10.33965/tpmc2019_201907L024
Conference: International Conference Theory and Practice in Modern Computing 2019

MONASH University

TYPICAL TIMES FOR FOR CONSCIOUS DESIGN

Project **inception**

- **architectural design** – big decisions!
- **hard/expensive to change**, so need to do it right

Before implementing something **complex or risky**

Refactoring existing code

- maybe the design was bad from the start
- maybe the design started out good, is no longer good

EXAMPLE OF unCONSCIOUS DESIGN

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

```
private static double FindInterestRate(int operationYear,
                                       Dictionary<int, double> yearToInterestRates) {
    //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;
    }
```

EXAMPLE OF unCONSCIOUS DESIGN

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

```
private static double FindInterestRate(int operationYear,
                                       Dictionary<int, double> yearToInterestRates) {
    //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;
    }
```

EXAMPLE OF unCONSCIOUS DESIGN

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

```
private static double FindInterestRate(int operationYear,
                                       Dictionary<int, double> yearToInterestRates) {
    //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;
    }
```

EXAMPLE OF unCONSCIOUS DESIGN

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

```
private static double FindInterestRate(int operationYear,
                                       Dictionary<int, double> yearToInterestRates) {
    //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;
    }
```

EXAMPLE OF unCONSCIOUS DESIGN

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

```
private static double FindInterestRate(int operationYear,
                                       Dictionary<int, double> yearToInterestRates) {
    //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;
    }
```

It's treating a Dictionary as a list or array rather than a Dictionary. This method is unnecessarily complex most likely due to some **earlier design commitments**.

Summary

We want to produce:

A "**good**" design

An understanding of that design **in the heads of stakeholders**

- a *stakeholder* is anyone who is affected by a (design) decision, including:
  - designers (obviously!)
  - implementers
  - maintainers
  - code/design reviewers
  - users (i.e. writers of client code, not end users)

Any documents/diagrams are in aid of the above

- they are not an objective in themselves

Summary

Reflection on the work you have done in this unit

The software development lifecycle

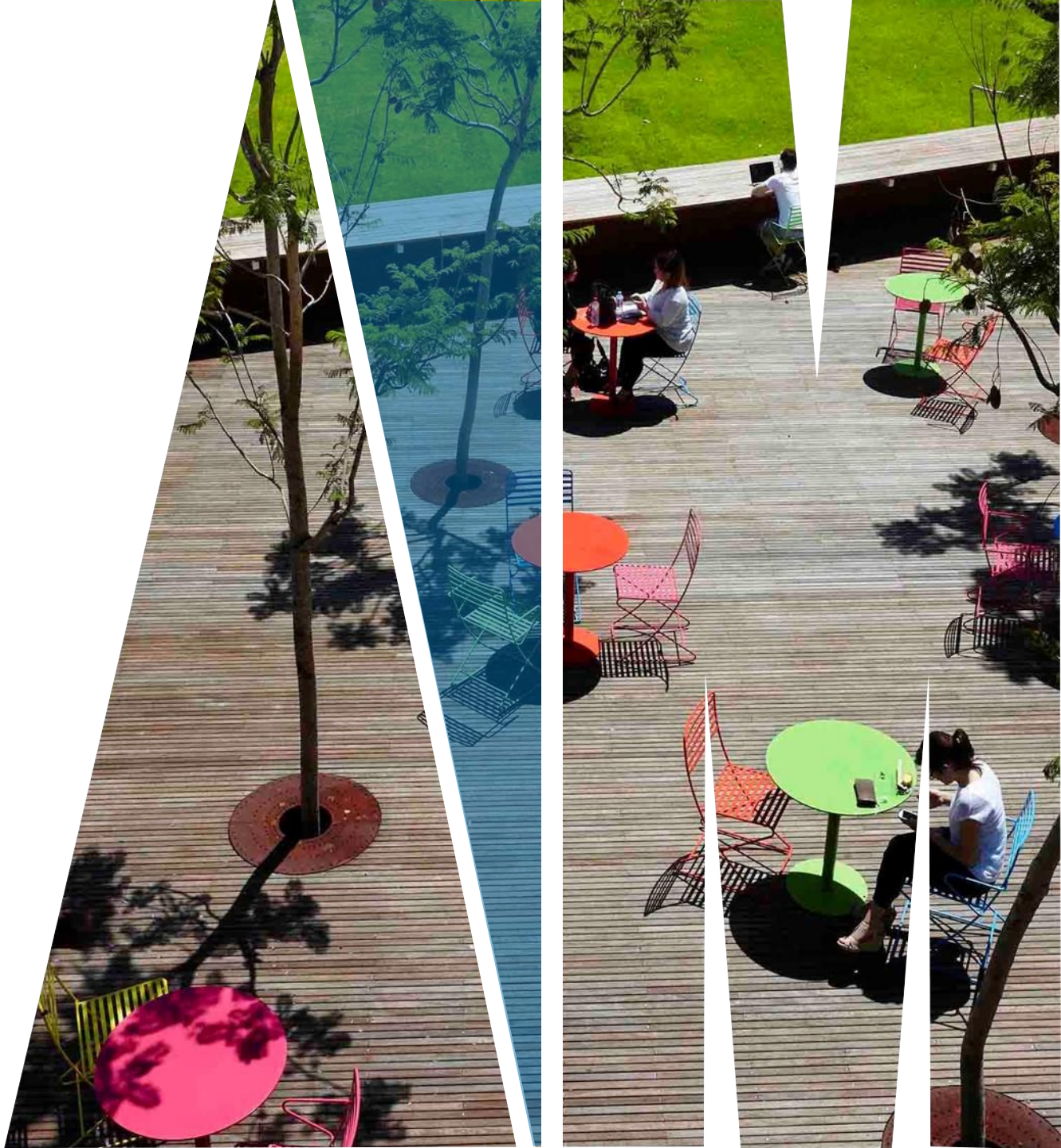
- analysis and design

When to design

Next chat on **Technical Debt**

MONASH University

Thanks



MONASH University