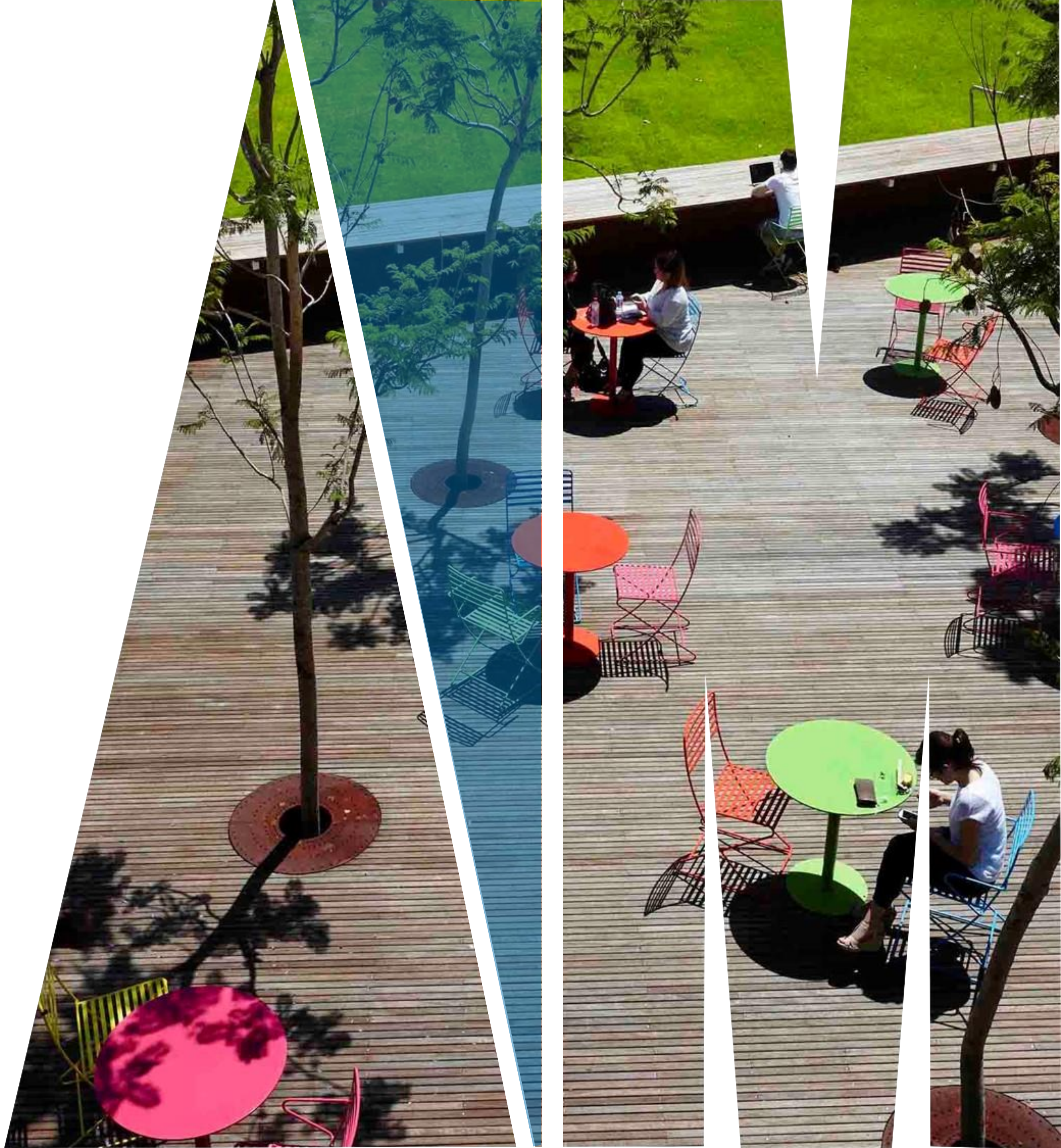
MONASH University

FIT2099 Object-Oriented Design and Implementation

The design process
(Part 1)



Outline

Software design as a **creative** act

Approaches to software design

- brainstorming and model-storming
- top-down
- bottom-up
- scenario-based
- CRC cards

The one big difference between art & design.

Artists have an audience of one. They work the art until they are satisfied.

Designers have an audience of many. They work the design until it effectively communicates to the target customer.

"**Clients** are the difference between design and art."

— Michael Bierut

A good design is **measurable**

TOP 10

SKILLS OF 2025

- 🔵 Analytical thinking and innovation
- 🟢 Active learning and learning strategies
- 🔵 Complex problem-solving
- 🔵 Critical thinking and analysis
- 🔵 Creativity, originality and initiative
- 🟠 Leadership and social influence
- 🟣 Technology use, monitoring and control
- 🟣 Technology design and programming
- 🟢 Resilience, stress tolerance and flexibility
- 🔵 Reasoning, problem-solving and ideation

🔵	Problem-solving
🟢	Self-management
🟠	Working with people
🟣	Technology use and development

Gamas
Customs & Logistics Group

CREATIVE
PROGRAMMING SKILLS

Software design is **a creative process**

- more than one way to do it well

We can identify some **techniques good designers use**

- if we practice them, we will get better at them

Applying techniques well **doesn’t guarantee** that our designs will always be good

- any more than taking art lessons will mean we produce excellent paintings
- but getting good at design is *always* going to involve **lots of practice**!

WHERE DO YOU START?

Start by ***understanding the problem domain***

Draw models of the problem domain, e.g.

- **Conceptual or Domain Class Diagrams** (to understand the concepts within the domain, and how they are related)
- **Activity diagrams** (to model business **processes**)

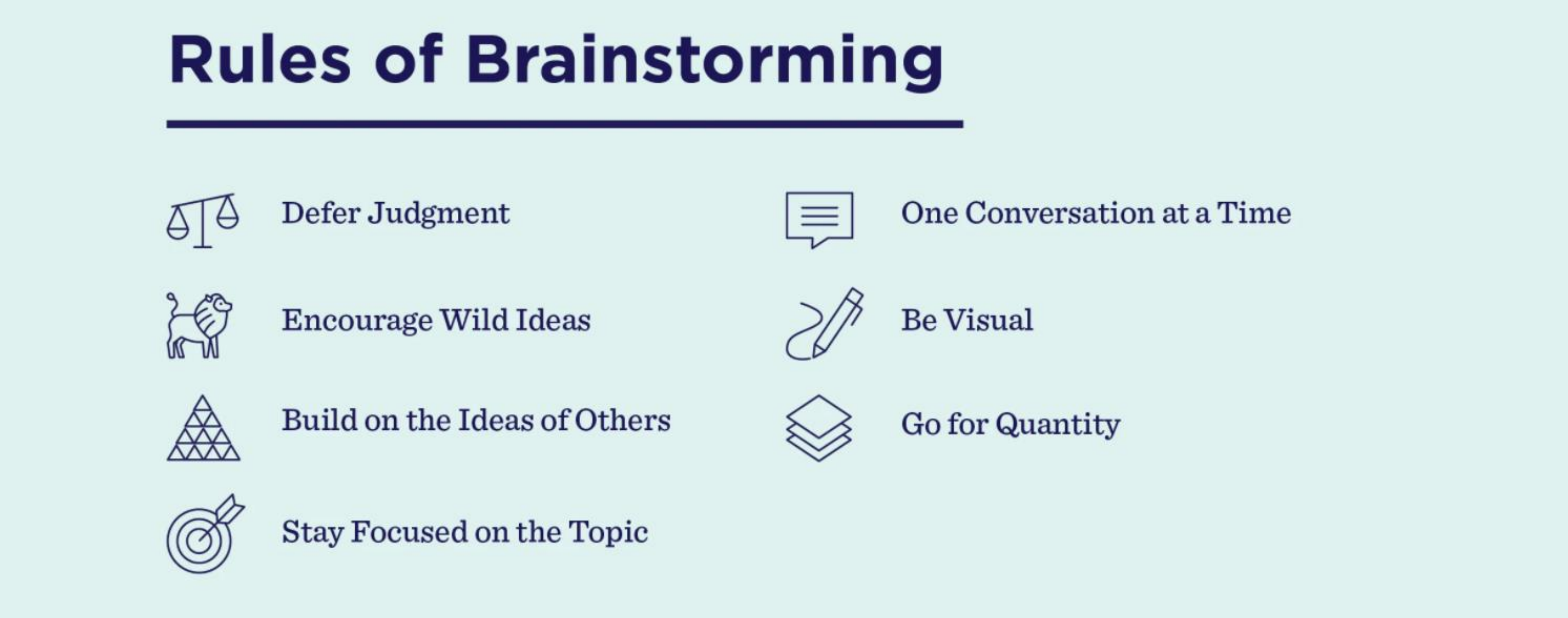
These can be evolved towards a design

Understanding the problem better will often make a solution obvious

THE
COLLABORATIVE DESIGN

Working with **a partner** or **small team** often works better than working alone, in software design.

THE BRAINSTORMING TECHNIQUE



Rules of Brainstorming

- Defer Judgment
- Encourage Wild Ideas
- Build on the Ideas of Others
- Stay Focused on the Topic
- One Conversation at a Time
- Be Visual
- Go for Quantity

THE BRAINSTORMING TECHNIQUE

General approach to **solving problems** requiring creativity in groups

- Popularized by Alex Faickney Osborn - advertising executive and author

Can be a good way to start

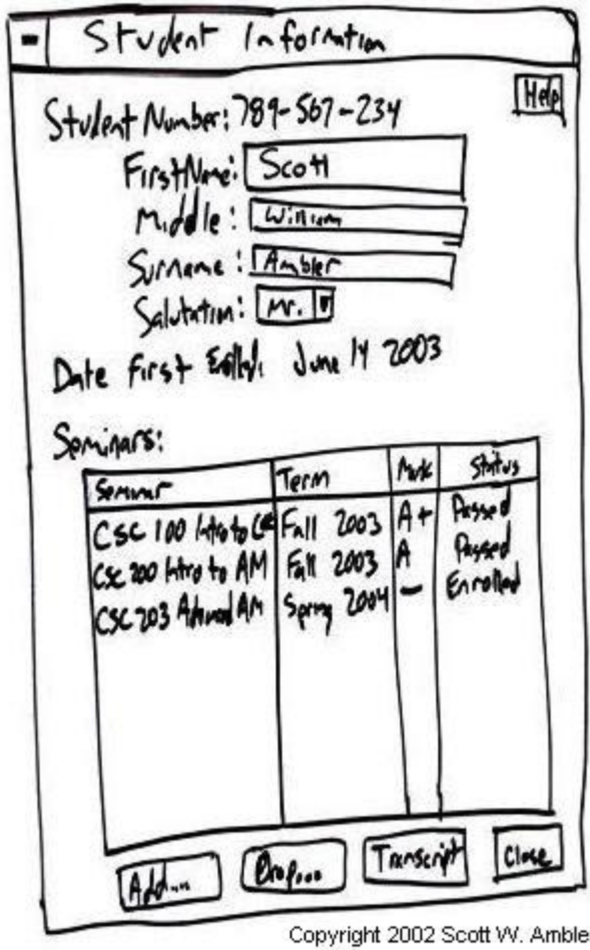
- you can throw out the chaff later

"**Model storming**" is a software-specific variation – see reading on Moodle

- perhaps not so much withholding of criticism

WHAT IS

MODEL STORMING?



Student Information

Student Number: 789-567-234

FirstName: Scott

Middle: William

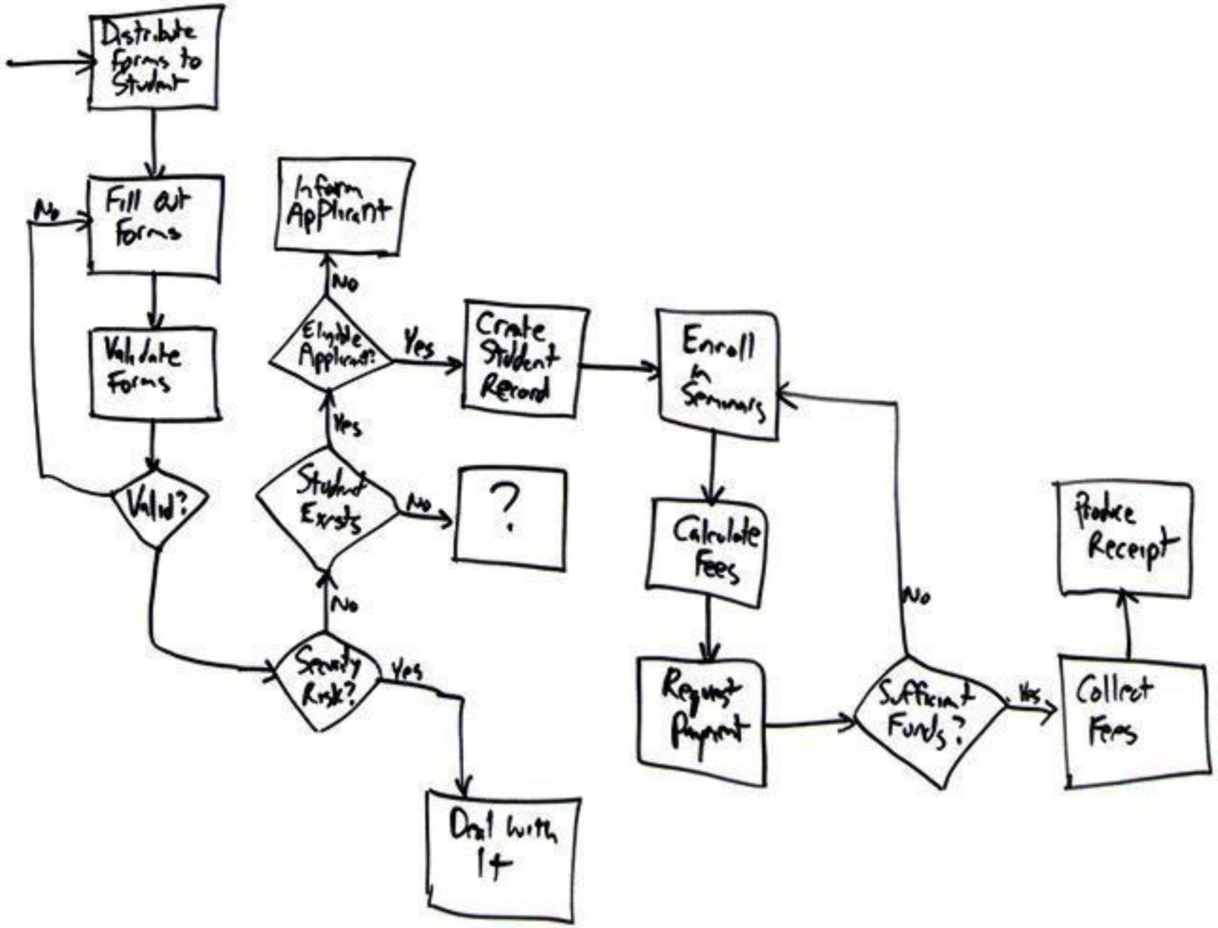
Surname: Ambler

Salutation: Mr. ▼

Date First Enrolled: June 14 2003

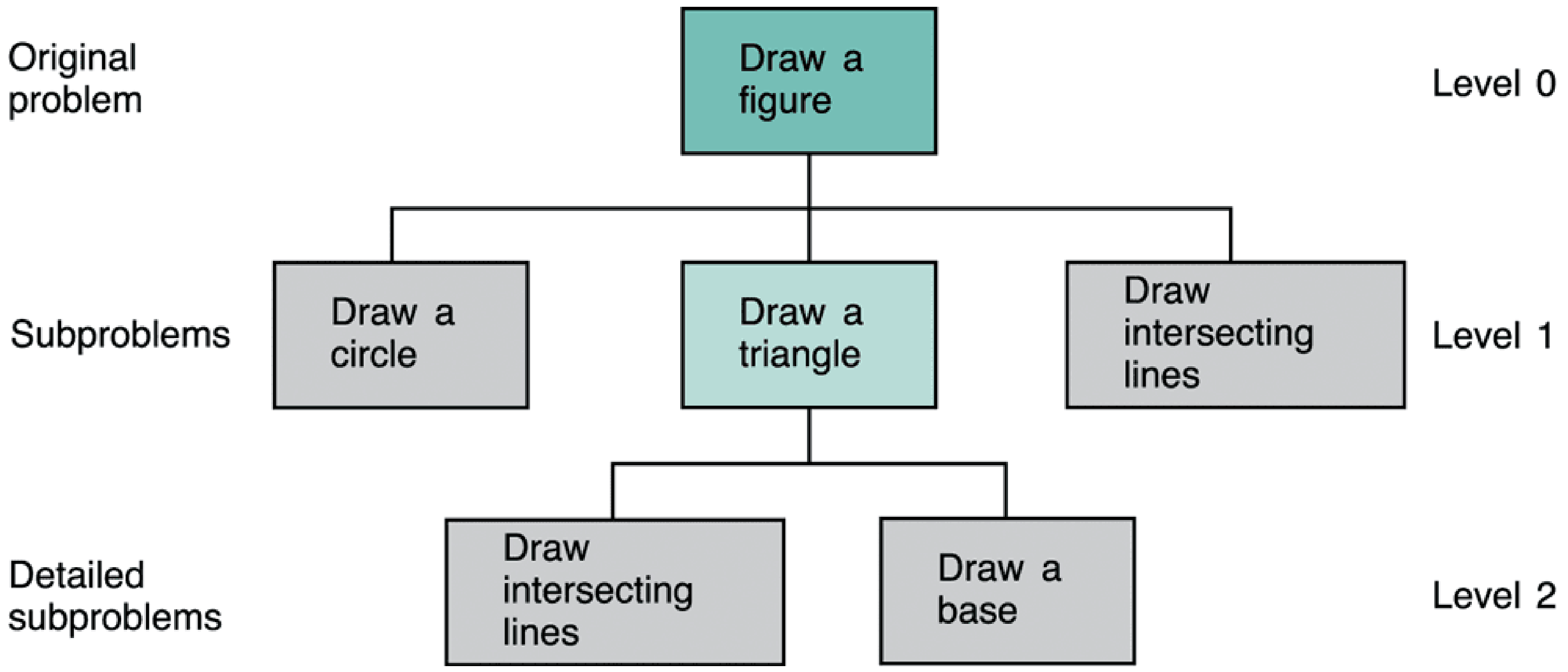
Seminars:

Seminar	Term	Mark	Status
CSC 100 Intro to CS	Fall 2003	A+	Passed
CSC 200 Intro to AM	Fall 2003	A	Passed
CSC 203 Advanced AM	Spring 2004	—	Enrolled



Check: http://agilemodeling.com/essays/modelStorming.htm

WHAT IS
TOP-DOWN PROGRAMMING?



WHAT IS
TOP-DOWN PROGRAMMING?

Top-down design

- start with **high-level problem**
- divide into **sub-problems**
  - perhaps recursively
- **design** to solve those
- put it **together**…

A very common approach in many branches of engineering

- but **can lead to repetition** due to repeated sub-problems if not careful
- may need **extensive refactoring** to reduce this

WHAT IS BOTTOM-UP PROGRAMMING?

Start with a small problem that you can solve

Design a solution to that

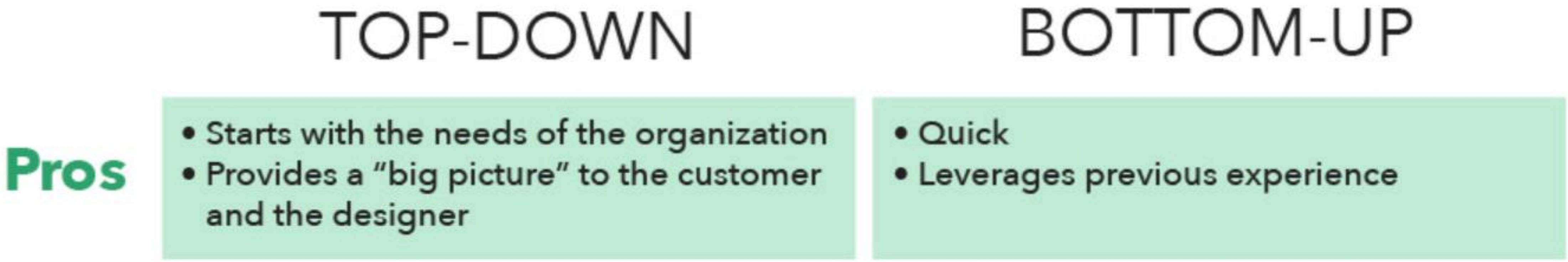
Do a few more…

Start putting them **together**

Voila…a solution!

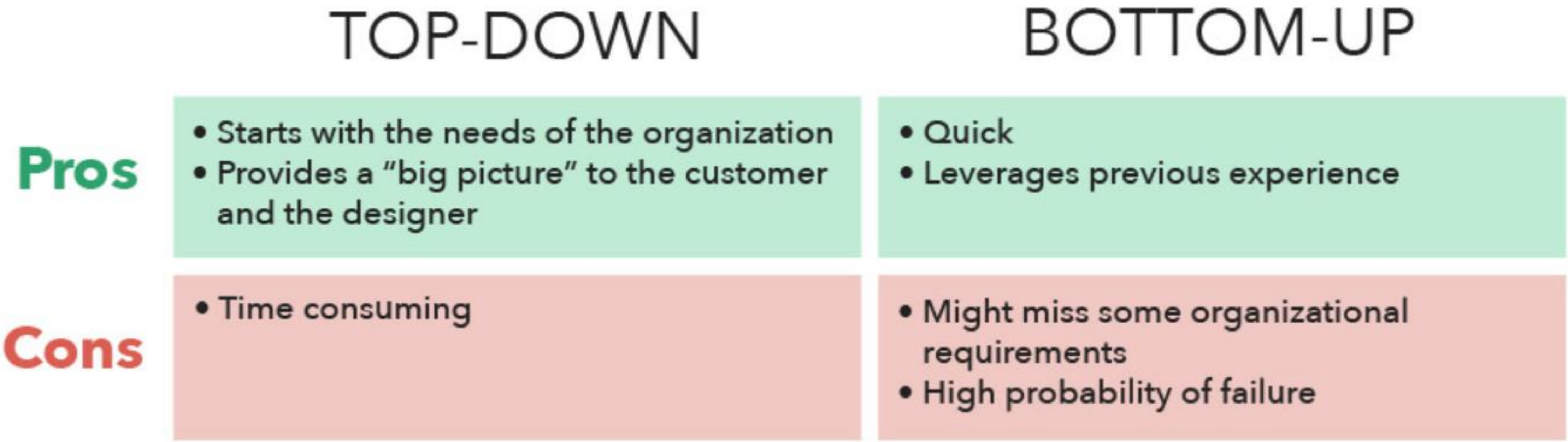
Can be useful to do a few “spikes” at the bottom level to gain understanding, and then to switch back to something more like top-down design – perhaps multiple times

TOP-DOWN VERSUS BOTTOM-UP



	TOP-DOWN	BOTTOM-UP
Pros	• Starts with the needs of the organization • Provides a "big picture" to the customer and the designer	• Quick • Leverages previous experience

TOP-DOWN VERSUS BOTTOM-UP



	TOP-DOWN	BOTTOM-UP
Pros	• Starts with the needs of the organization • Provides a "big picture" to the customer and the designer	• Quick • Leverages previous experience
Cons	• Time consuming	• Might miss some organizational requirements • High probability of failure

APPROACH

Have some **scenario**(s) that the thing being designed needs to support

- storyboard, use case, activity diagram, plain text, etc.
- this may come out of requirements or analysis (depending on whether thing is "the system" or some small part of it)

Work through your scenario(s)

- trace through your design as it stands

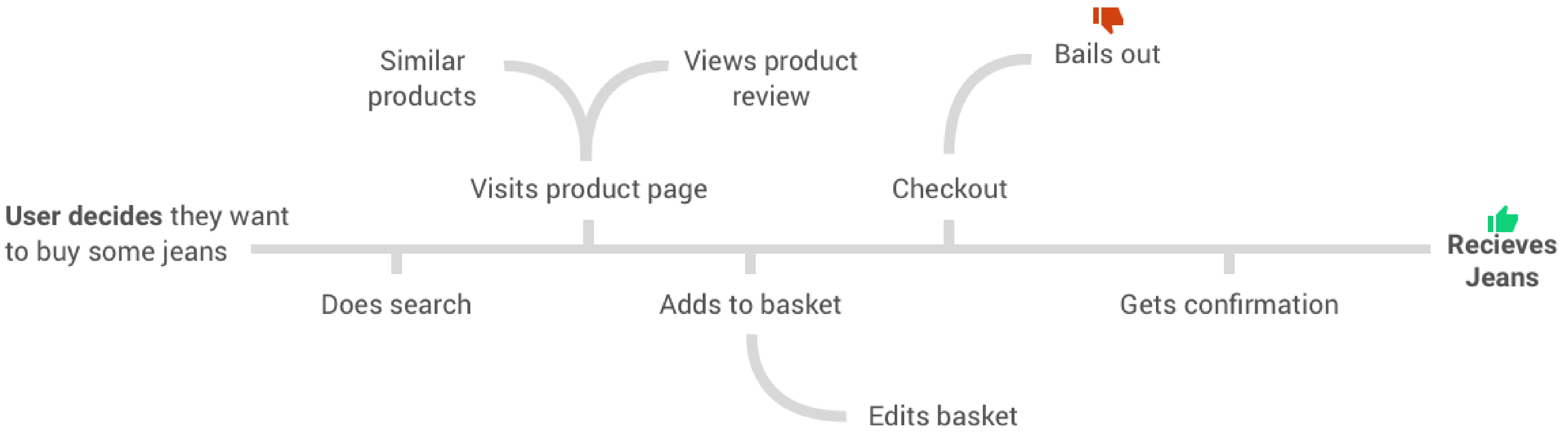
Modify/rework design to support scenario effectively

- keep quality properties in mind

Repeat with additional scenarios

SCENARIO-BASED DESIGN
APPROACH

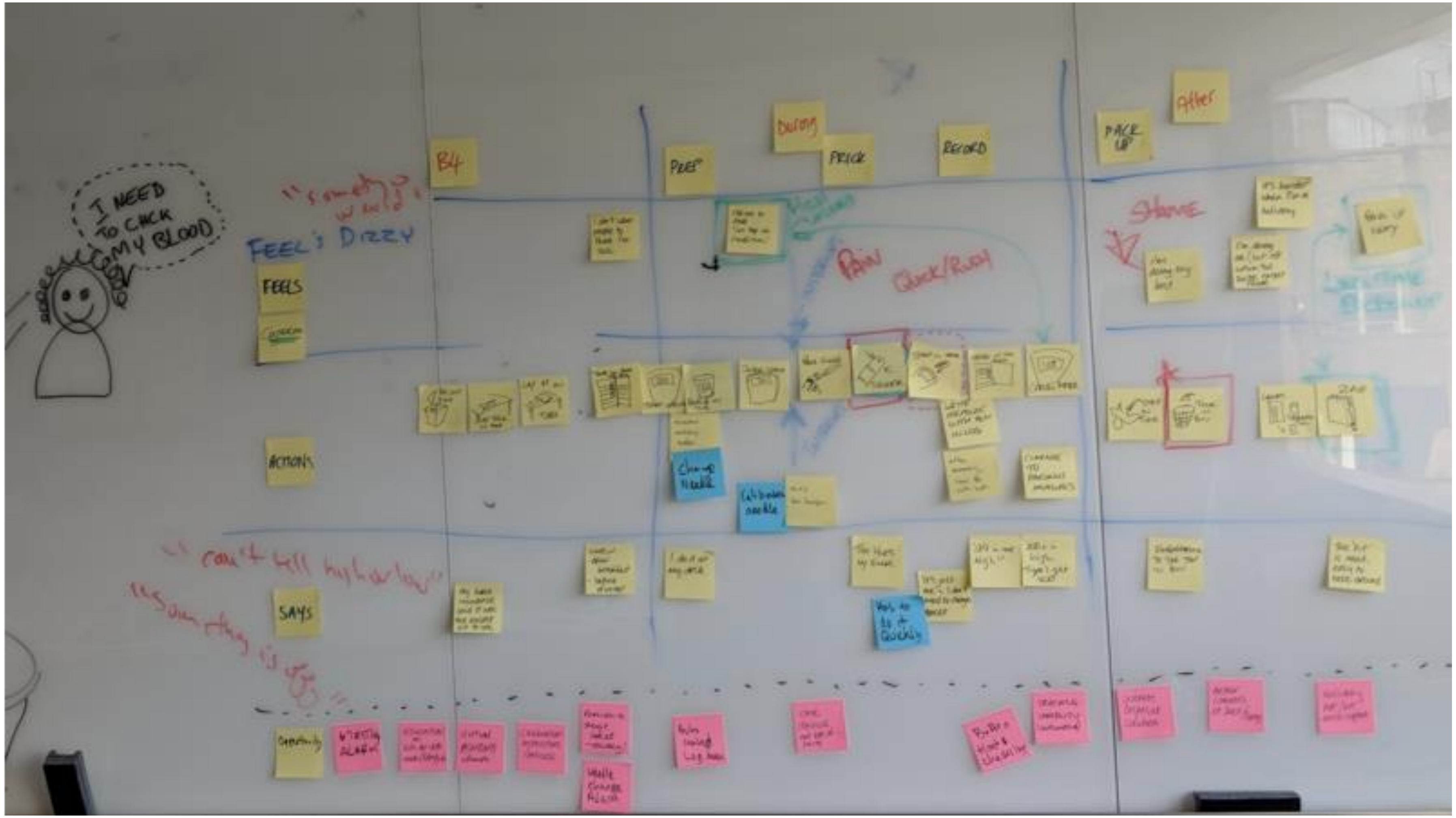
Scenario where a user would go through the flow of buying a pair of jeans (user journey mapping technique).



SCENARIO-BASED DESIGN

APPROACH

Scenario of the paths a user makes when using their diabetes medicine.

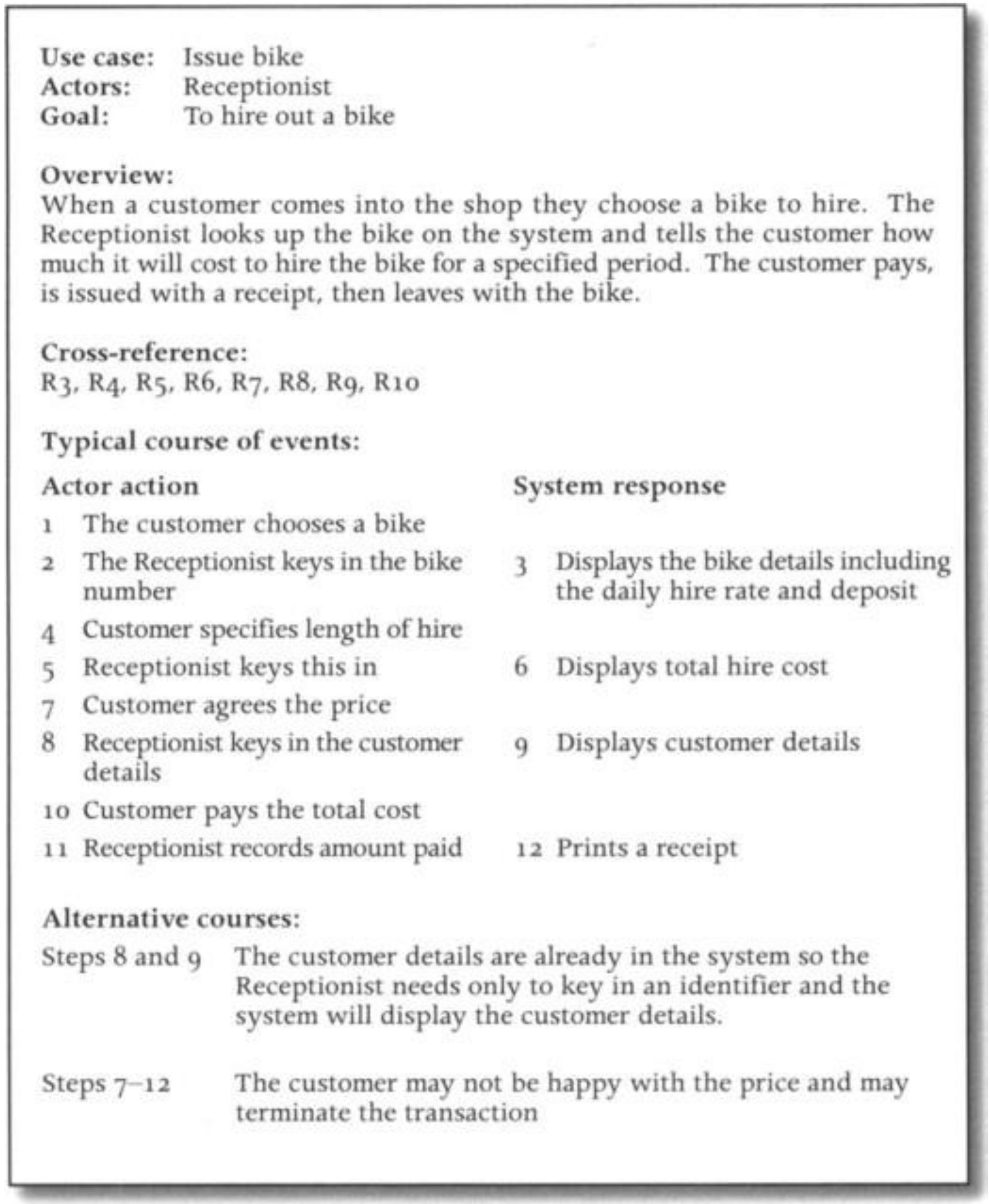


THE
USE CASES

Many teams using use cases eventually discover two disadvantages:

1) natural language text unfortunately allows a great deal of **ambiguity**, and

2) reading and reviewing any non-trivial use case has the potential to become **tedious**.



Use case: Issue bike
Actors: Receptionist
Goal: To hire out a bike

Overview:
When a customer comes into the shop they choose a bike to hire. The Receptionist looks up the bike on the system and tells the customer how much it will cost to hire the bike for a specified period. The customer pays, is issued with a receipt, then leaves with the bike.

Cross-reference:
R3, R4, R5, R6, R7, R8, R9, R10

Typical course of events:

Actor action	System response
1 The customer chooses a bike	
2 The Receptionist keys in the bike number	3 Displays the bike details including the daily hire rate and deposit
4 Customer specifies length of hire	
5 Receptionist keys this in	6 Displays total hire cost
7 Customer agrees the price	
8 Receptionist keys in the customer details	9 Displays customer details
10 Customer pays the total cost	
11 Receptionist records amount paid	12 Prints a receipt

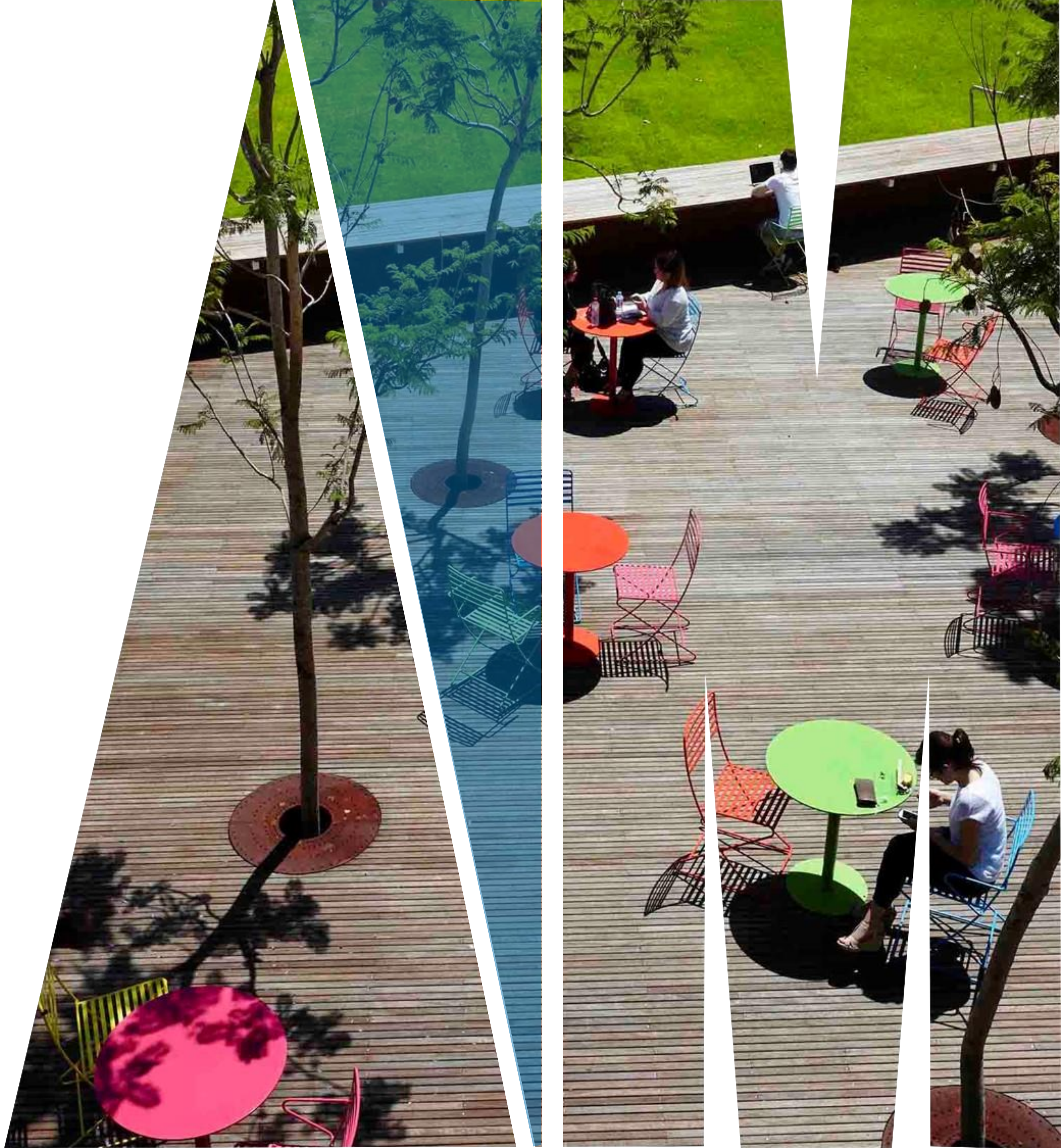
Alternative courses:

Steps 8 and 9 The customer details are already in the system so the Receptionist needs only to key in an identifier and the system will display the customer details.

Steps 7–12 The customer may not be happy with the price and may terminate the transaction

MONASH
University

Thanks

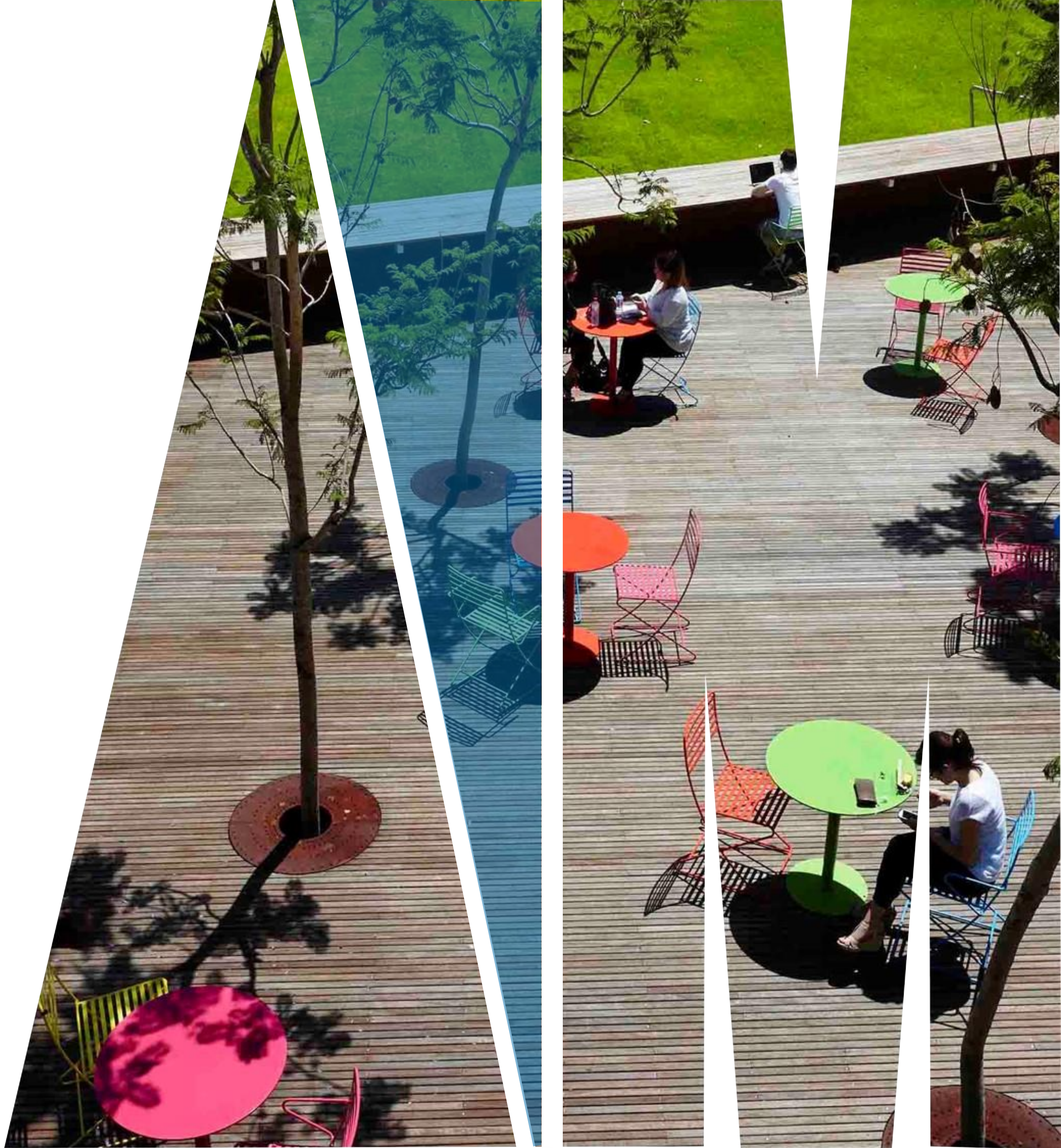


MONASH
University

MONASH
University

FIT2099 Object-Oriented Design and Implementation

The design process
(Part 2: CRC cards)

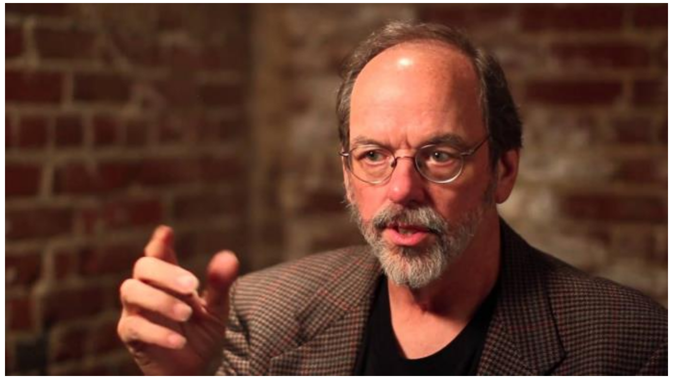


THE
CRC CARDS

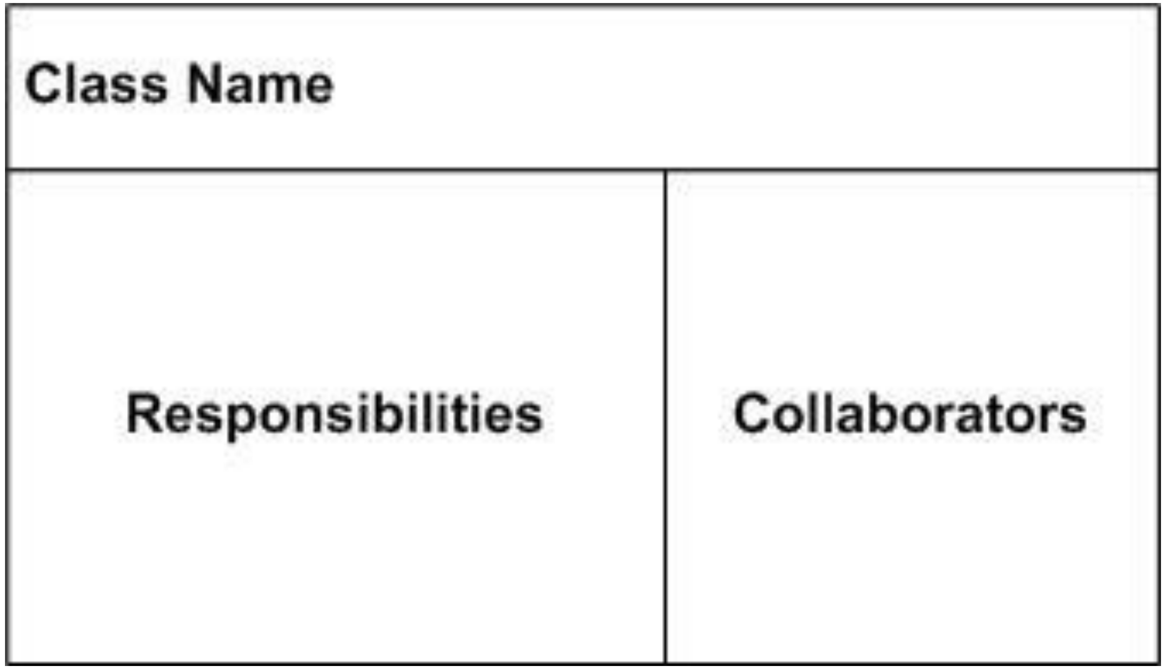
Class-Responsibility-Collaboration cards.

Invented by Ward Cunningham as an OO design **teaching** tool

You don't need a special notation for doing this. But some people find an alternative notation useful at some points



He also invented the 'wiki'



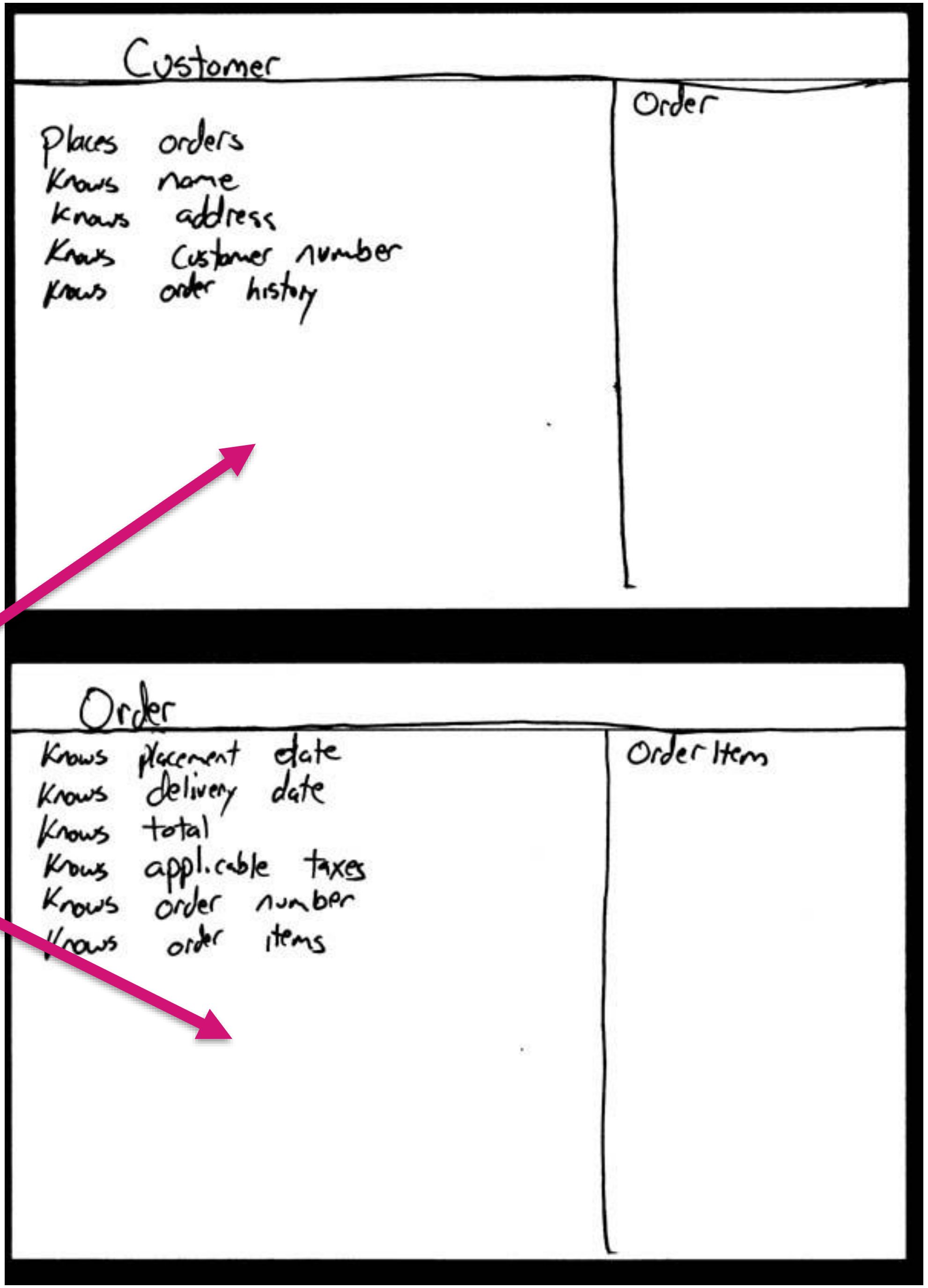
Class Name

Responsibilities

Collaborators

CRC CARDS
EXAMPLES

Responsibilities

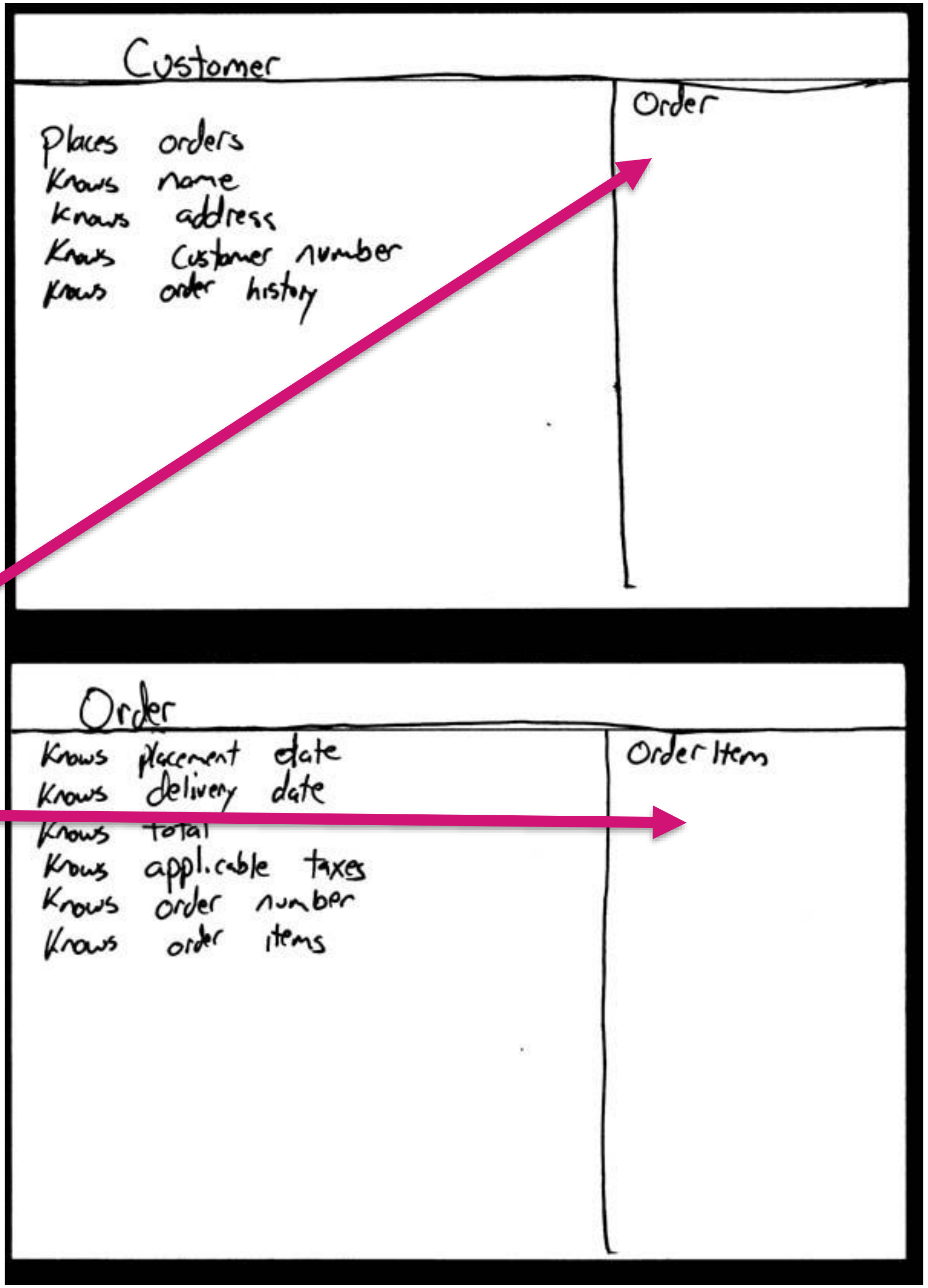


Customer	
Places orders	Order
Knows name	
knows address	
Knows customer number	
knows order history	

Order	
Knows placement date	Order Item
Knows delivery date	
Knows total	
Knows applicable taxes	
Knows order number	
Knows order items	

CRC CARDS

EXAMPLES



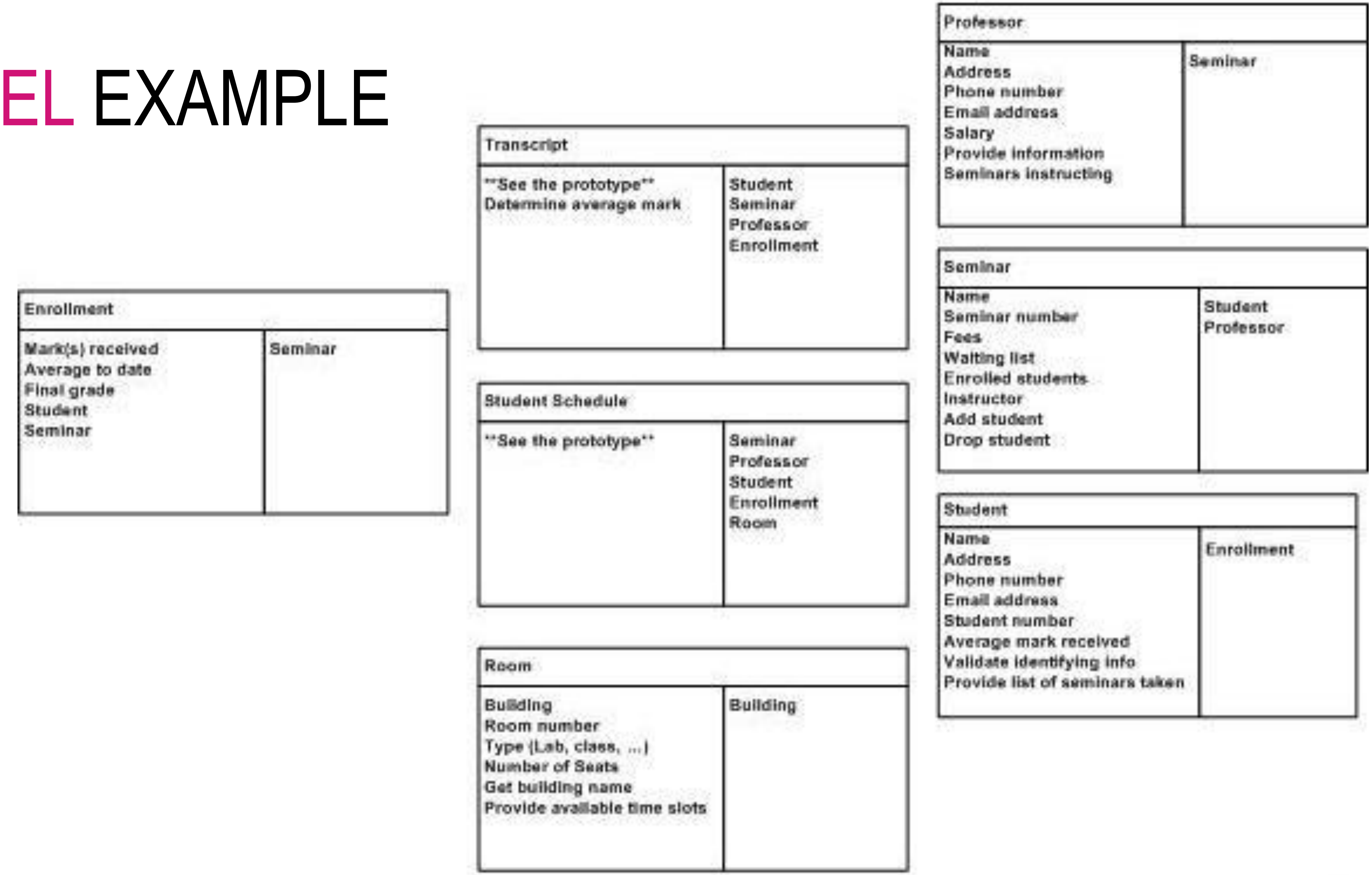
Customer	
Places orders Knows name knows address Knows customer number Knows order history	Order

Order	
Knows placement date Knows delivery date Knows total Knows applicable taxes Knows order number Knows order items	Order Item

Collaborators

MONASH University

CRC
MODEL EXAMPLE



Enrollment	
Mark(s) received Average to date Final grade Student Seminar	Seminar

Transcript	
See the prototype Determine average mark	Student Seminar Professor Enrollment

Student Schedule	
See the prototype	Seminar Professor Student Enrollment Room

Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

USING
CRC CARDS

We start with only **one or two obvious cards** and start playing "what-if"' (with scenarios)

If the situation calls for **a new responsibility**, either

- **add the responsibility** to one of the objects, or
- **create a new object**

Add collaborations as we go (associations)

If design can be improved, **rewrite the card**(s)

Use a magnet to stick them on a whiteboard, if available

USING
CRC CARDS

Have different people “play the object” during a scenario

Messages between objects -> *“Hey Unit, gimme a list of students enrolled in you…”*

Pick up the card whose role they are assuming while “executing” a scenario

When a new responsibility emerges, add it!

Use Case Roleplays
How can you check use cases?
Two people read out the card
– One person plays the user
– The other plays the system
…her team members listen
…es the use case make sense?
…each participant have all the information
…ey need it?

USING
CRC CARDS

If card becomes too full:

- copy the information on its card to a new card
- express responsibilities more succinctly/abstractly

If a succinct rewrite is not possible:

- **maybe your object is trying to do too much**
- remember the **SRP**! (single responsibility)
- **split object up** according to its responsibilities

Check this relatively recent article that talks about the use of CRC for identifying refactoring opportunities: https://www.simplybusiness.co.uk/about-us/tech/2020/03/refactoring-using-crc/

CRC CARDS

TOP-DOWN or BOTTOM UP?

Whatever works for the group!

> Design with the cards tends to progress **from knowns to unknowns**, as opposed to top-down or bottom up. We have observed two teams arriving at essentially the same design through nearly opposite sequences, one starting with device drivers, the other with high-level models.
>
> – Kent Beck and Ward Cunningham,
> *A Laboratory For Teaching Object-Oriented Thinking*

ARE CRC CARDS ENOUGH?

If you're Kent Beck, probably…

> Certainly XP de-emphasizes diagrams to a great extent. Although the official position is along the lines of "use them if they are useful", there is a strong subtext of "real XPers don't do diagrams".
>
> –Robert C. Martin, Is Design Dead

But…**not a good way communicate to "outsiders"**:

> We know of one case where finished cards were delivered to a client as (partial) design documentation. Although the team that produced the cards was quite happy with the design, the recipient **was unable to make sense of the cards** out of context.
>
> – Kent Beck and Ward Cunningham
> A Laboratory For Teaching Object-Oriented Thinking

CRC CARDS,
CONNASCENCE AND ENCAPSULATION

CRC card process helps with **encapsulation**

CRC cards encourage **small objects** with clear responsibilities

Doesn’t *guarantee* a good design

Always keep design principles in mind

Summary

Software design is a **complex creative activity**

“How” to do it is often **poorly articulated**

Designing in **small teams helps**

brainstorming, model storming, CRC cards, UML on whiteboards

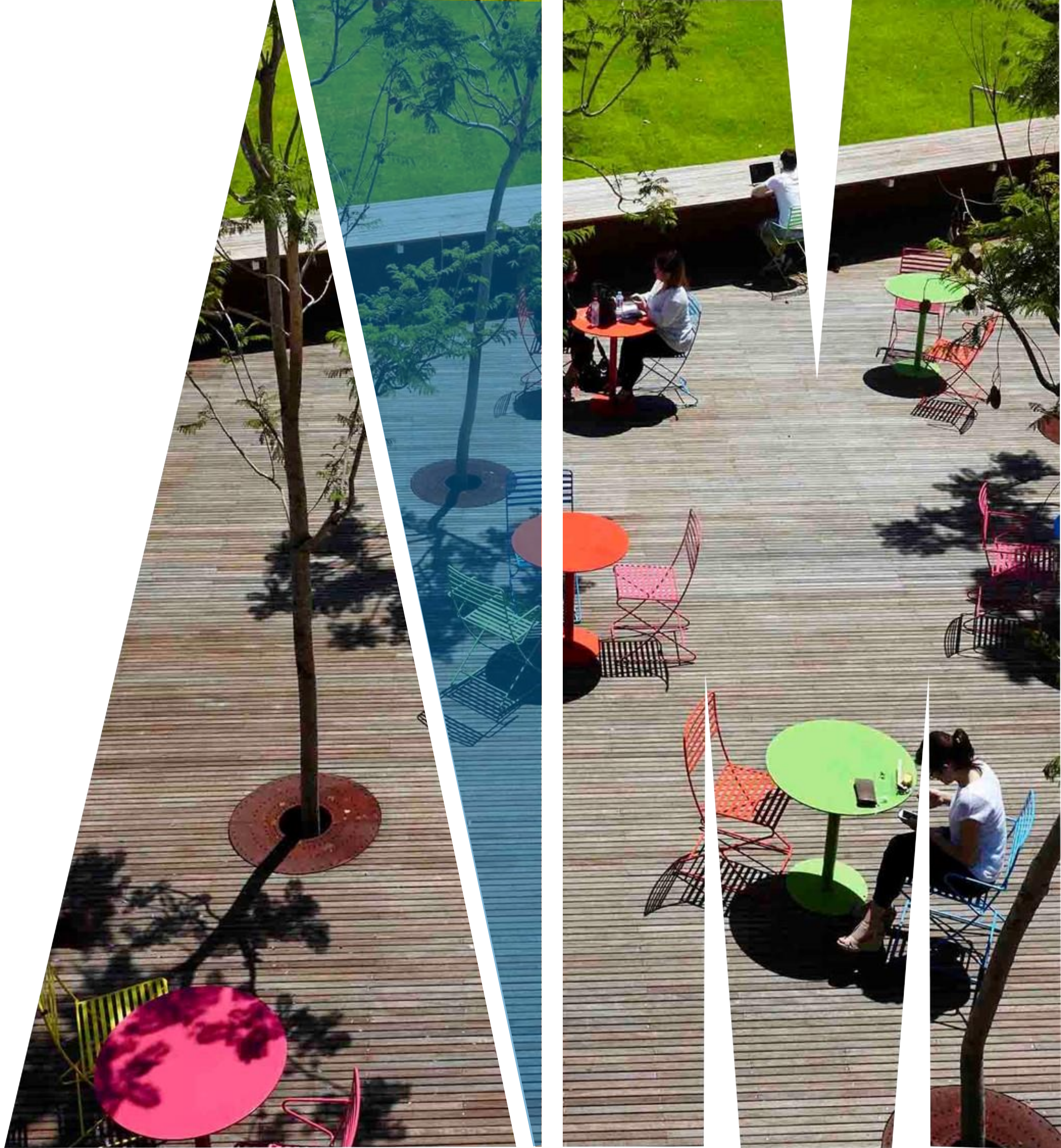
Top-down, bottom-up and working through scenarios

CRC cards – tool to help work through scenarios to evolve a design

Need lots of practice!

MONASH
University

Thanks



MONASH
University