Electronics and
Computer Science

University of Southampton

# Introduction to Formal Methods
# &
# Event-B

COMP6226: Software Modelling Tools and Techniques for Critical Systems

Dr A. Rezazadeh (Reza)
Email: ra3@ecs.soton.ac.uk or ar4k06@soton.ac.uk

November 24

University of Southampton

# Overview

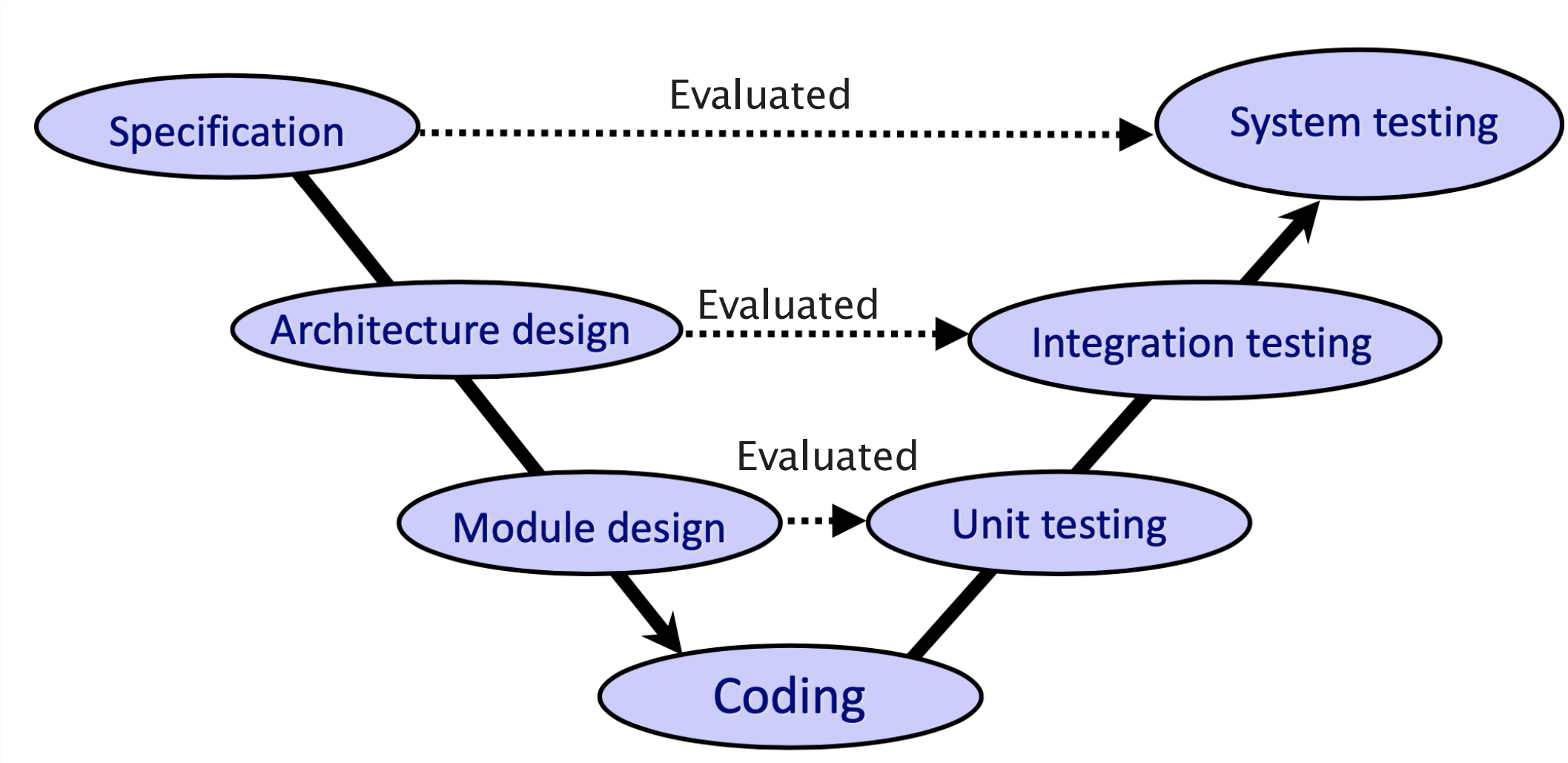
- Objectives
- V Model what is wrong with this model?
- Software Defects
- Need for Precision and Abstraction
- Testing vs Proving
- Formal Methods
- Event-B in Software Development
- Elements of Event-B Formalism
- A Simple Event-B Example
- Notations used in Event-B
- Other Formalisms
- Event-B an evolution of previous formalisms

University of Southampton

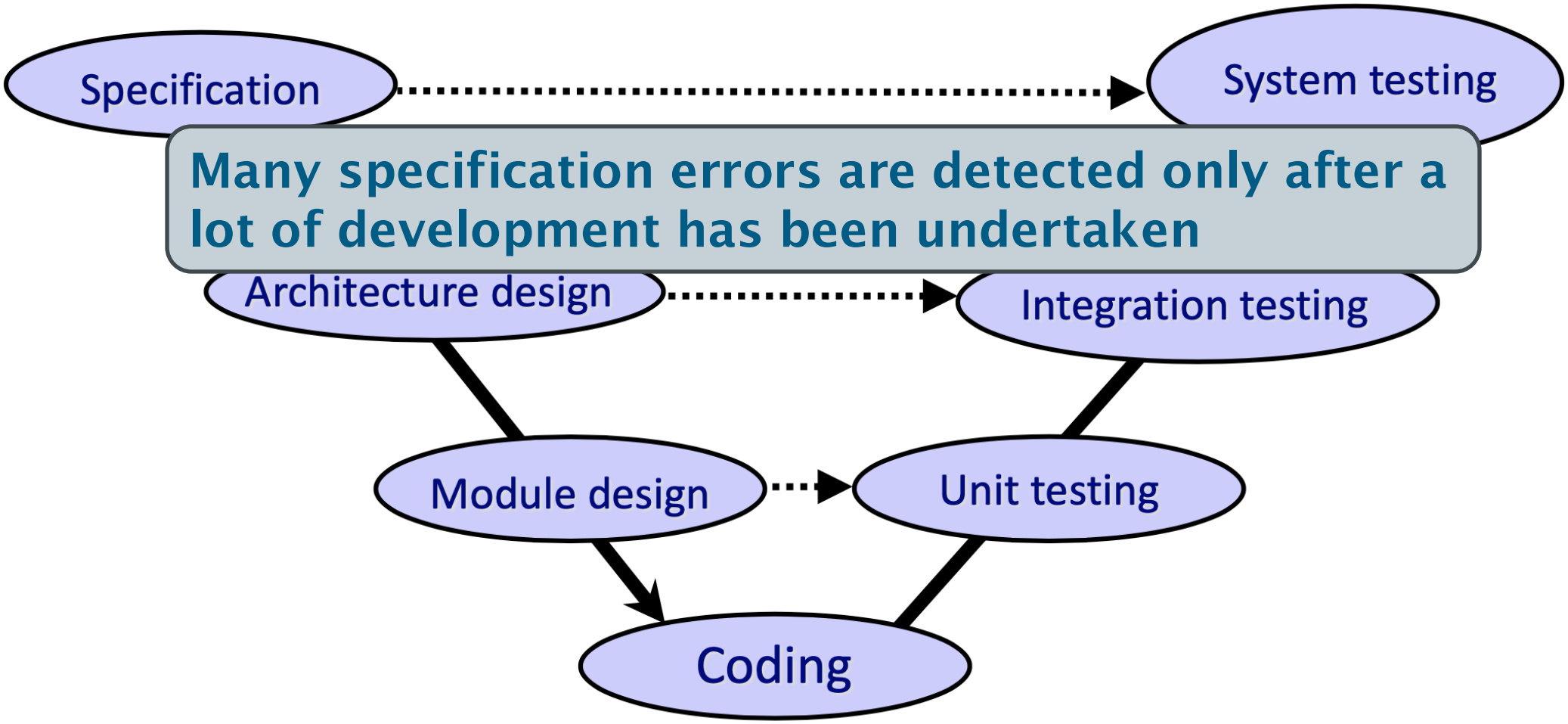
# Objectives

- Motivation for using formal modelling
  - Modelling vs Programming
  - Proving vs Testing
- Introducing formal modelling using Event-B
  - Behavioural modelling in Event-B: Machine, Events
  - Coffee Club Example
- Applications of Event-B

# V Model in Software Development



# What is wrong with the V model?



University of Southampton

# Defects Discovered too Late…

- "Requirements and architecture defects make up approximately 70% of all system defects"

- "80% of these defects are discovered late in the development life cycle"

- Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems Carnegie Mellon SEI, 2013

  https://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_47803.pdf

# Software Defects

- Software Faults Are Due to:
  - Requirements defects: incomplete, ambiguous requirements or failure to specify the environment in which the software will be used.
  - Design defects: not satisfying the requirements, incomplete design or documentation defects
  - Code defects: failure of code to conform to software designs.

University of Southampton

# Software Artefact Issues

人工制品

- Use of natural languages in Requirement Engineering
  - Natural language Ambiguity
  - Inconsistencies
- Too much complexity
  - Complexity of requirements
  - Complexity of operating environment
  - Complexity of design
- Software is more error sensitive
  - Conventional engineering is tolerant
- Harder to test
  - Complete test is impossible
  - Has correlated failures

University of Southampton

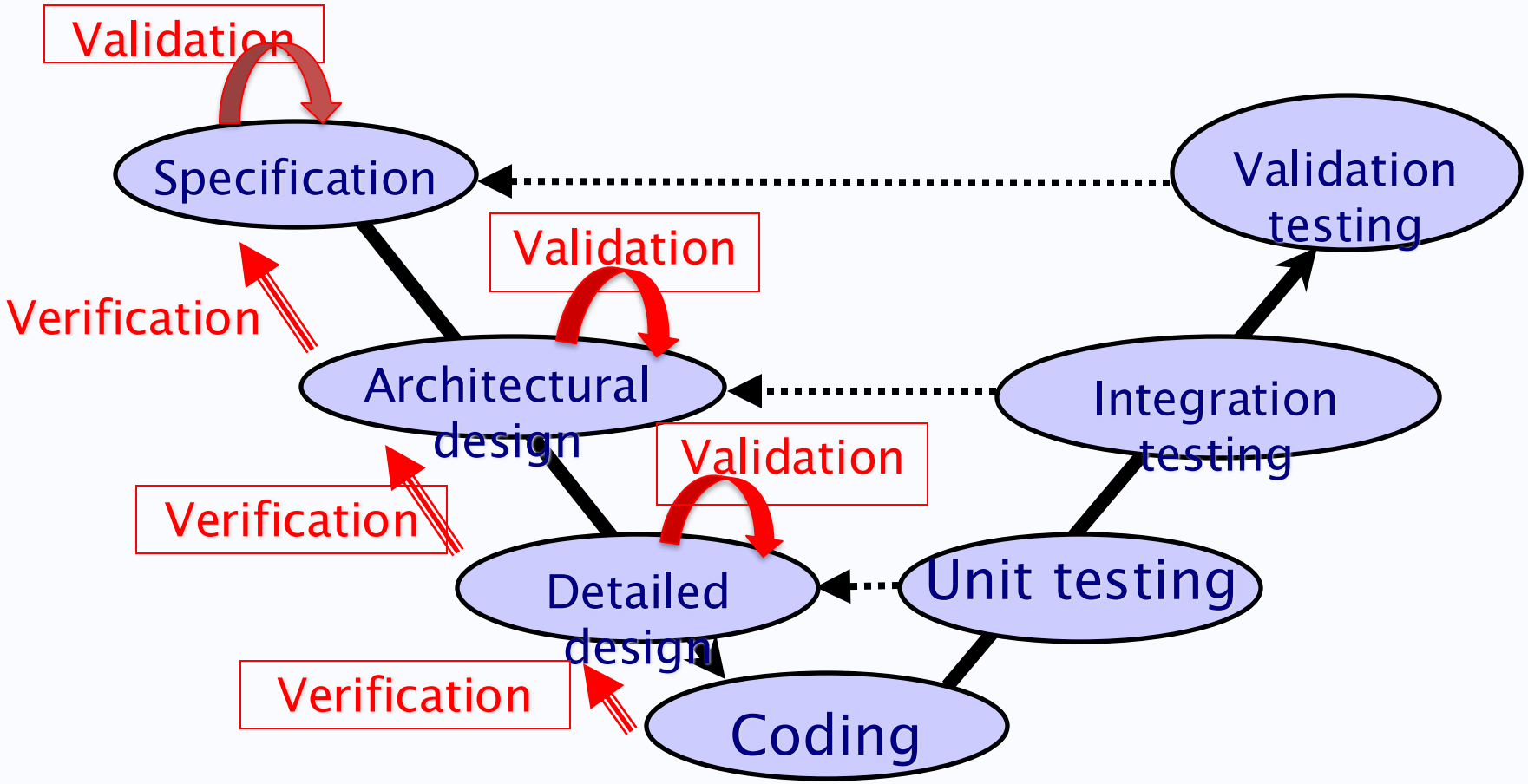
# Need for Precision and Abstraction

- Precision through early-stage models
  - Amenable to analysis by tools
  - Identify and fixing ambiguities and inconsistencies as early as possible

- Mastering complexity through abstraction
  - Focus on **what** a system does (its purpose) at early stages of modelling
  - Incremental analysis and design – answering the the question of **how** later on

University of Southampton

# The Need for Early-stage Analysis



# Testing vs Proving

don't guarantee the correctness

- The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements.
  - Testing provides evidence of software faults (negative evidence)
- What happen if we can ascertain or establish the correctness or validity of solutions; to verify; to prove certain aspects or properties of software
  - If you prove that something is true or correct, you provide evidence showing that it is definitely true or correct.
  - Proof provides positive evidence of software correctness

证明 n.

- Which one is better, testing or proof?
  - We need both

do test and prove together

conducting the prove early is better

but definiry correctness is not easy.

University of Southampton

# Formal Methods – Definition

textual based

- Formal methods refers to mathematically based techniques for the specification, development and verification of software and hardware systems. (From Wikipedia)
- What kind of mathematics?
  - Discrete mathematics, set theory, logic
- Advantages of formal methods
  - Encourages us to think before coding
  - Do some reasoning
  - Help us to remove ambiguities
  - Enforce the consistency of the specification and modelling

University of Southampton

# Question

- UML is a formal Language
  - True not mathematical
  - False ☑ based

公开的

The **Object Constraint Language** (**OCL**) is a declarative language describing rules applying to Unified Modeling Language (UML) models developed at IBM and is now part of the UML standard.

University of Southampton

# Event-B in Software Development

- **Event-B**: A formal language for writing high-level specifications of computer systems
  - System specifications are derived from requirements (abstraction)
  - System specification can be gradually turned into design (refinement)
- Event-B language includes first order logic and set theory
  - Formal specification is more precise and consistent than an informal (natural language) specification.
- Event-B typically used in safety-critical, security-critical or mission-critical applications.

before programming



# Events In Event-B Formalism

suitable for event-driven

- A System model is represented by a construct called "Machine"
- A Machine is made of a number of Events representing the behaviour of the system
- An Events is made of guards and actions
- The Guards denote the enabling condition of the event
- The Actions denote the way the state is modified by the event
- Guards and actions are written using set-theoretic expressions
- The state of model is represented using variables

# A Simple Event-B Example: CoffeeClub

- For a coffee club, we require a Moneybank that stores money used by the coffee club.
- A Few requirement of the Moneybank:
  - REQ1: We need a money bank for storing and reclaiming finite, non-negative funds for a coffee club.
  - REQ2: We need an operation for adding money to the money bank.
  - REQ3: We need an operation for removing money from the money bank; but it cannot remove more money than the money bank contains.

# Modelling Requirements in Event-B

- REQ1: We need a money bank for storing and reclaiming finite, non-negative funds for a coffee club.

```
machine CoffeeClub
variables
moneybank   // The machine state is represented by the variable
            // moneybank, denoting the money bank for the coffee club.

invariants
@inv1: moneybank ∈ ℕ  //REQ1: moneybank must not be negative
```

The invariants specify the properties that the variables (the state) must satisfied by your system all the time.

# Notation Used

- Some Keywords:

  **Machine**, **variables**, **Invariants**

- Mathematical notations

| **Notation** | | |
|---|---|---|
| math | ascii | |
| $\in$ | : | set membership |
| $\mathbb{N}$ | NAT | the set of natural numbers = non-negative integers |

# Events

- **Events** model possible behaviour of the system presented in a machine
- Events include:
  - the conditions under which the behaviour can occur (called guards)
  - and how the state of the machine is changed (named as actions)
- The machine always should start from a known state (called initial state)
- Thus, we need an ***Initialisation*** event, a special unconditional event
  - occurs in a machine once only
  - before any other event
  - ***initialises*** the machine's variables to values that establishes the invariant.

University of Southampton

# Initialisation Event

- Variables do not have any known value before initialisation.

```
events
    event INITIALISATION
    then
      @act1: moneybank ≔ 0   //moneybank could be initialised
                             //to any natural number
    end
```

# Modelling REQ2

- REQ2: We need an operation for adding money to the money bank.

**event** addMoney
Parameter **any** amount
**where**
Guard @grd1: amount ∈ ℕ1 //ℕ1 is used to prevent the event firing
// uselessly for amount = 0
**then**
Action @act1: moneybank ≔ moneybank + amount //add amount to
//moneybank
**end**

University of Southampton

# Notation Used

- Some Keywords:
  - **Event**, **any**, **where**, **then**, **end**
- Mathematical notations

**Notation**

| math | ascii | |
|---|---|---|
| $:=$ | := | "becomes equal to": where $x := e$ means assign to the variable $x$ the value of the expression $e$ |
| $\mathbb{N}1$ | NAT1 | the set of non-zero natural numbers |

# Modelling REQ3

- REQ3: We need an operation for removing money from the money bank;
  - **but it cannot remove more money than the money bank contains.**

```
event withdrawMoney
any amount
where
@grd1: amount ∈ 1 .. moneybank //The amount must not exceed the
                                // contents of moneybank.
                                // There is no need to remove an
                                // amount of 0
then
@act1: moneybank ≔ moneybank − amount //subtract amount from
                                      // moneybank
end
```

**What happens if we remove this constraint?**

# Events – Reminder

- **Events** model possible behaviour of the system presented in a machine
- **Events** include:
  - the conditions under which the behaviour can occur (guards)
  - and how the state of the machine is changed (actions)
- The machine always should start from a known state
- Thus, we need an Initialisation event, a special unconditional event
  - occurs in a machine once only
  - before any other event
  - initialises the machine's variables to values that establishes the invariant.

# Event – Operational Interpretation 不用解决多线程问题

- An event execution is supposed to take no time
  - Thus, no two events can occur simultaneously
- When all events have false guards, the discrete system stops
- When more than one event have true guards, one of them is chosen non-deterministically and its action modifies the state
- The previous phase is repeated (if possible)

different between system and software.

```
Initialize;
while (some events have true guards) {
   Choose one such event;
   Modify the state accordingly;
}
```

# Other Formalisms

- VDM (Bjørner & Jones , 1970s)
  - IBM Vienna Labs: Vienna Development Method
  - Designed for defining programming languages
  - Extended to specify sequential programs
- Z Notation (Oxford group , 1980s)
  - Specification of software systems
  - Makes extensive use of set theory and logic
- B Method (Abrial, 1990s)
  - Evolved from Z, and aimed at software modelling software with emphasis on tools (proof + code generation)
  - Mainly used in railway industry
- Alloy (Jackson, 1990s/2000s)
  - Focus on modelling and automated verification

# Event-B an evolved version of B-Method

- B-Method was designed for *software* development
- Realisation that it is important to reason about system behaviour, not just software
- Event-B is intended for modelling and reasoning about system behaviour
- Event-B is supported by Rodin tool (www.event-b.org)
- Rodin is an Open source, Eclipse-based, open architecture tool
  - Range of plug-in tools (provers, ProB model checker, UML- B,...)

# Summary

- A model is representation of the system we want to build
- An effective modelling approach should allow us to reason about the system during its specification & design
- **Formal methods** as a modelling tool allow us
  - To reason about the intended behaviour of the system under consideration
  - Define its behaviour (what it does)
  - Incorporate constraints (what it must not do)
- Event-B is a mathematical technique for system level modelling & analysis
  - Supported by Rodin tools-set – an Eclipse-based IDE

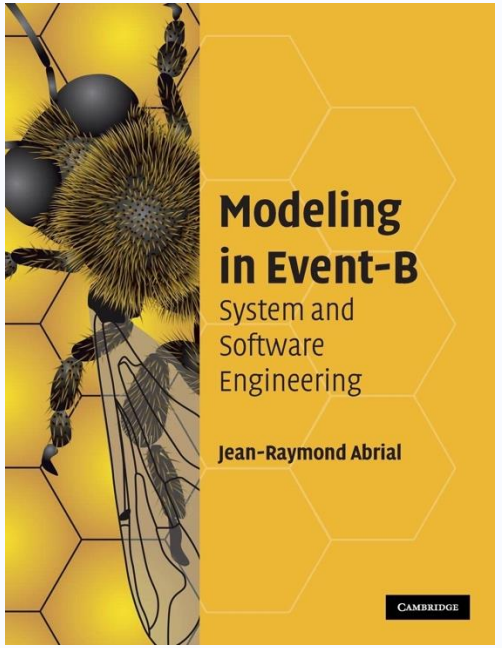
University of Southampton

# Reference

- *System Modelling & Design in Event-B* by Ken Robinson
  - Read Chapter 2 : System Modelling & Design
  - and Chapter 3: Section 3.1: p. 5-7 excluding proof obligations

    https://wiki.event-b.org/images/SM%26D-KAR.pdf

- *Modelling in Event-B: System and Software Engineering* by Jean-Raymond Abrial
  - Read Chapter 1 from p 12.
- Event-B and Rodin Documentation Wiki



University of Southampton

# YOUR QUESTIONS