Software Formal Engineering Methods
（软件形式化工程方法）

Shaoying Liu（刘少英）

Graduate School of Advanced Science and Engineering

Hiroshima University, Japan

Email: **sliu@hiroshima-u.ac.jp**

HP: **https://home.hiroshima-u.ac.jp/sliu/**

The greatest challenge in software engineering

How to ensure the correctness and high reliability of software systems?

Disastrous consequences of software errors

1. **Errors With Rocket Launch**

Time: 1996

Accident: a satellite's payload was not delivered into the planned Earth's orbit using a European Ariane 5 rocket. The rocket self-destructed when it started disintegrating.
Cause: a bug with the software used in launching the rocket. the problem was caused by the reuse of code from Ariane 4, the rocket's predecessor.

Disastrous consequences of software errors

2. Heathrow Terminal 5 Opening, 2008

Time: 2008

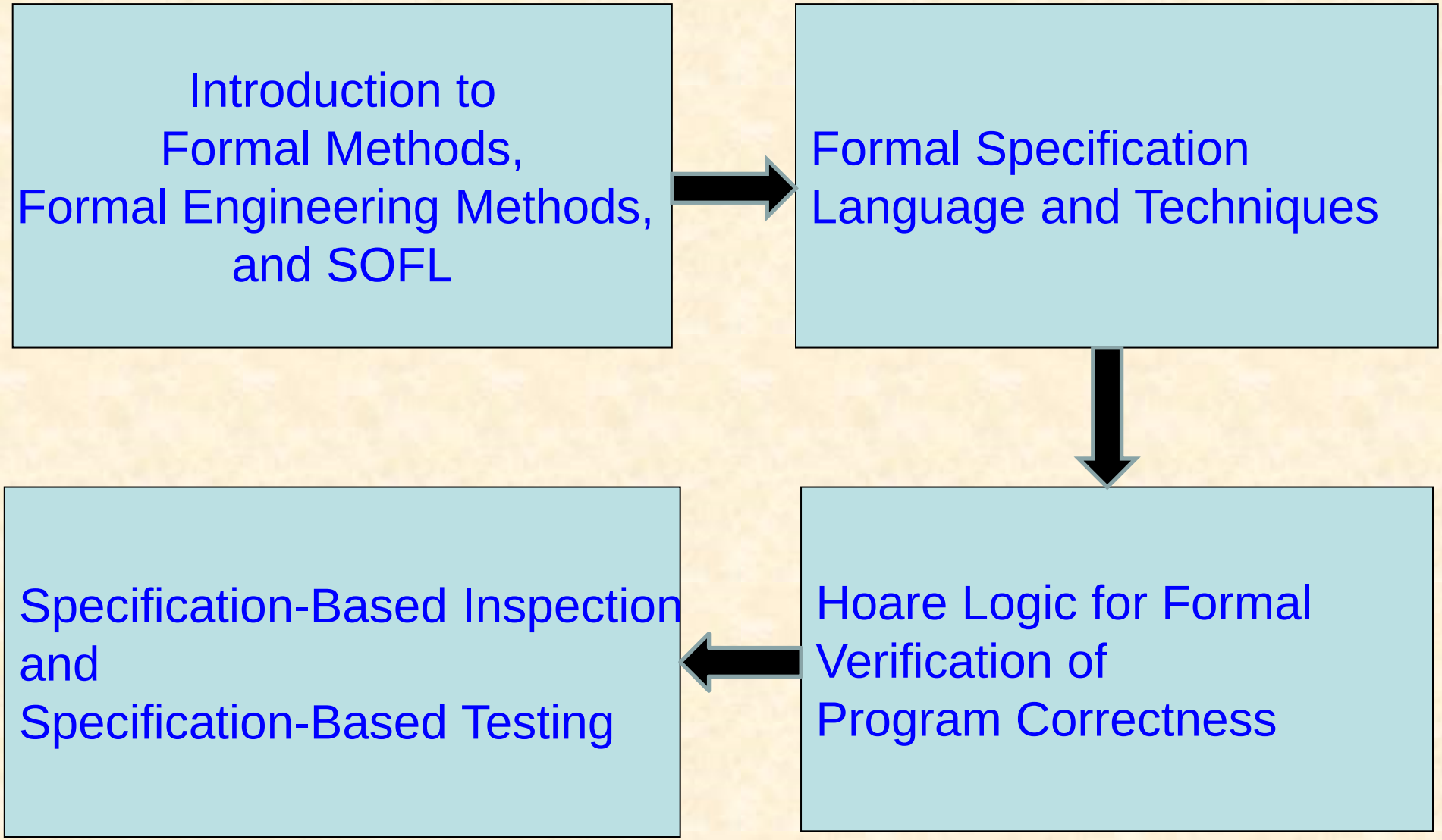
Accident: massive disruptions in managing baggage. Some 42,000 bags were lost and more than 500 flights canceled, costing more than £16 million.

Cause: a bug in the new baggage handling software system and problems with the wireless network system.

Goals of the course

- Understand what Formal Engineering Methods are and learn necessary techniques for writing formal specifications
- Understand the concept of program correctness and learn Hoare logic for formally verifying the correctness
- Learn the principles of specification-based inspection and specification-based testing as practical techniques for verifying programs

Contents of the course



Reference books

(1) C.A.R. Hoare and C.B. Jones, ``Essays in Computing Science'', Prentice Hall, 1989.

(2) Shaoying Liu, "Formal Engineering for Industrial Software Development", Springer-Verlag, 2004.

(3) Latest publications on specification-based inspection and testing.

The Textbook

"**Formal Engineering for Industrial Software Development Using the SOFL Method**",

by **Shaoying Liu**,

Springer-Verlag, 2004,

ISBN 3-540-20602-7

软件开发的形式化工程方法

——结构化+面向对象+形式化

清华大学出版社

The ways to learn

- Attend lectures
- Work on class exercises and actively join discussions
- Complete a small project
- Join the final examination

Evaluation of Performance

The evaluation of each student's performance for grading will be done based on a small project and the final examination.

(1) Small project: 40%

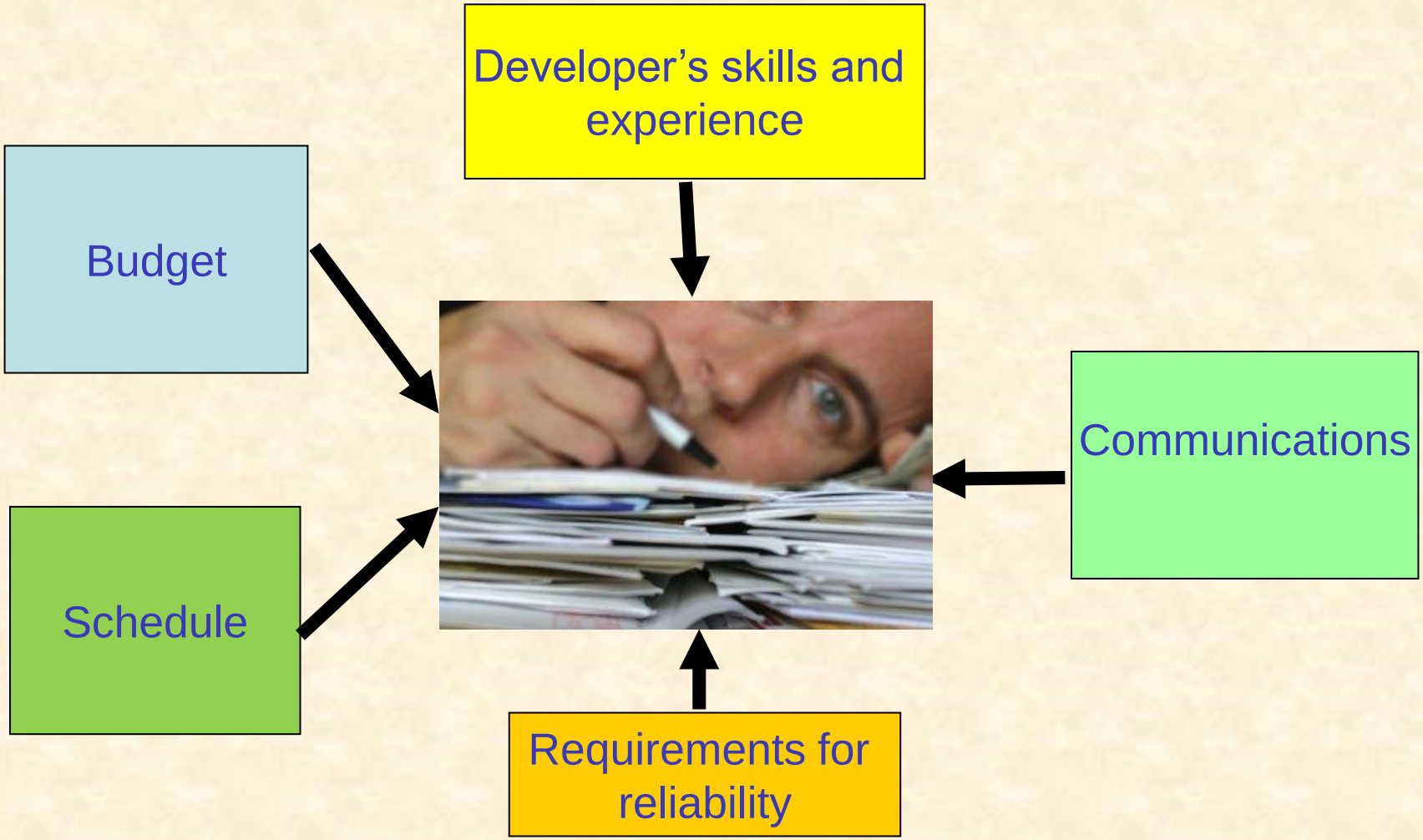
(2) Examination: 60% (90 minutes)

I. Introduction to Formal Methods, Formal Engineering Methods, and SOFL

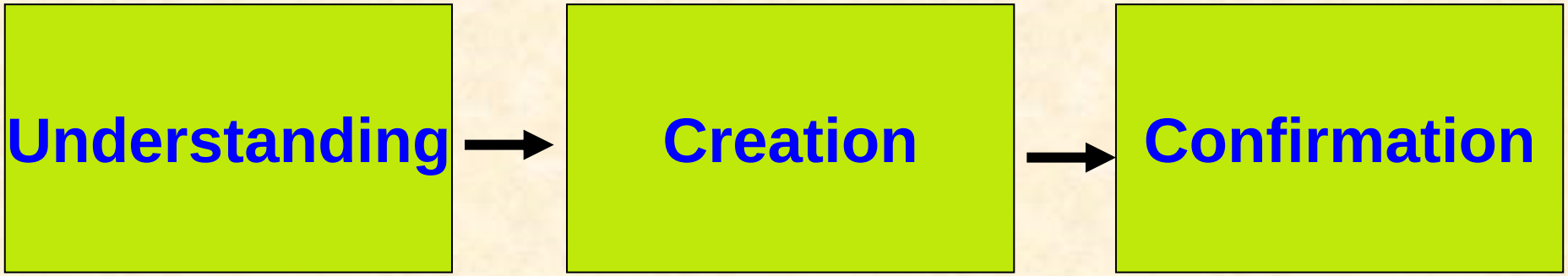
1. Challenges in traditional Software Engineering
2. Formal Methods for improvement
3. Formal Engineering Methods for practicality
4. SOFL

I.1 Challenges in traditional Software Engineering

The challenges for software projects in traditional software engineering come from many aspects.



The major tasks in software development:



Correctness &

Reliability!!!

What is a reliable system?

input

Sausage machine

output

Requirement: given a piglet as input, the machine will produce sausages automatically.

An example of unreliable system:

input

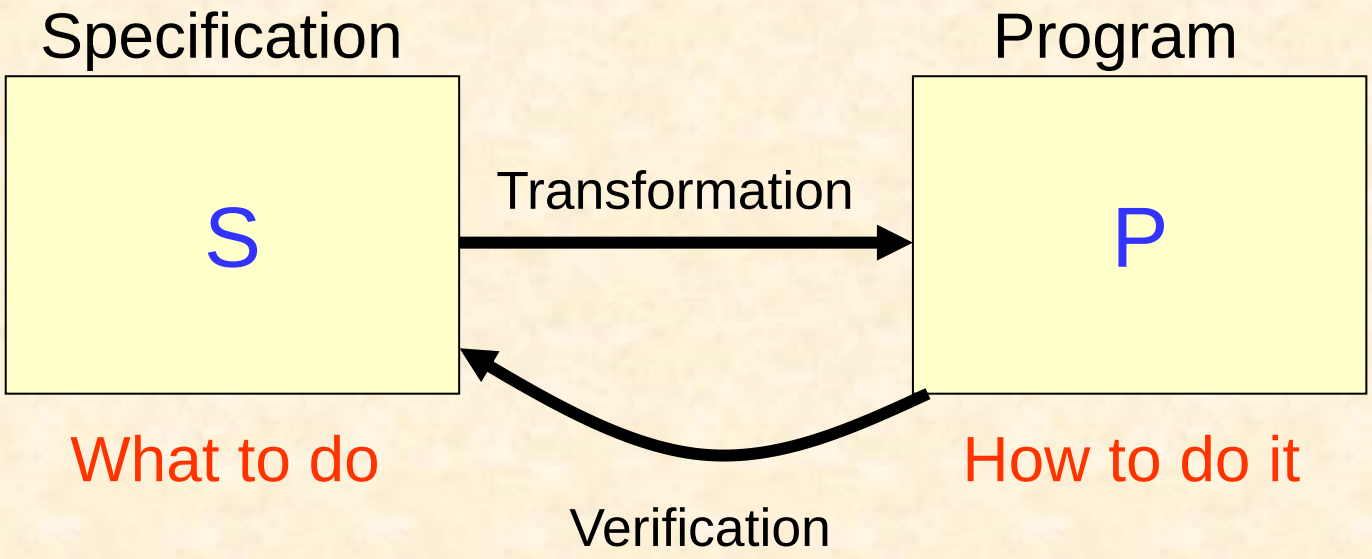
Sausage machine

beef

output

Requirement: given a piglet, the machine will produce sausages automatically.

An abstract process model



- How to write S so that it can effectively help the developer understand the user's requirements completely and accurately?
- How to ensure that S is not ambiguous so that it can be correctly understood by all the developers involved?
- How can S be effectively used for the construction of program P, and for the verification (e.g., proof, inspection, and testing) of P?
- How can software tools effectively support the analysis of S, transformation from S to P, and verification of P against S?

An example of informal specification:

"A software system for an Automated Teller Machine (ATM) is required to provide services on various accounts. The services include operations on current account, operations on savings account, transferring money between accounts, managing foreign currency account, and change password. The operations on a current or savings account include deposit, withdraw, show balance, and print out transaction records."

A better way to write the same specification:

"A software system for an automated teller machine (ATM) is required to provide services on various accounts.

The services include
① operations on current account
② operations on savings account
③ transferring money between accounts
④ managing foreign currency account,
⑤ change password.

The operations on a current or savings account include
① deposit
② withdraw
③ show balance
④ print out transaction records."

The major problems with informal specifications:

- Informal specifications are usually ambiguous, which is likely to cause misinterpretations.
- Informal specifications are difficult to be used for implementation and for inspection and testing of programs because of the big gap between the functional descriptions in the specifications and the program structures.
- Informal specifications are difficult to be analyzed for their consistency and validity.
- Informal specifications are difficult to be supported by software tools in their analysis, transformation, and management (e.g., search, change, reuse).

2006年
4月9日の
朝日新聞

バグ頻発 デジタル製品

エンジン止まる車、電源切れぬテレビ

修正に忙殺 コスト増大

携帯1台、80年代銀行システム並み

ソフト複雑 人材は不足

機能多すぎに疑問の声

Reality of software development in practice

Manager:

Why is the project over budget and behind schedule?

Client:

Why does the software system behave differently from my requirements?

Programmer:

Why are there so many bugs remaining in the program?

Why is my own program difficult to understand even by myself?

A possible solution to these problems:

Formal Methods!!!

I.2 Formal methods for improvement

Formal methods mean **verified design**.

Formal methods =

Formal Specification (VDM, Z, B-Method)

+

Refinement (Back, Morgan)

+

Formal Verification (Floyd-Hoare, Dijkstra)

Set theory, logics, algebra, etc.

The most commonly used formal notations

(1) VDM-SL (Vienna Development Method – Specification Language),
IBM Research Laboratory in Vienna

References:

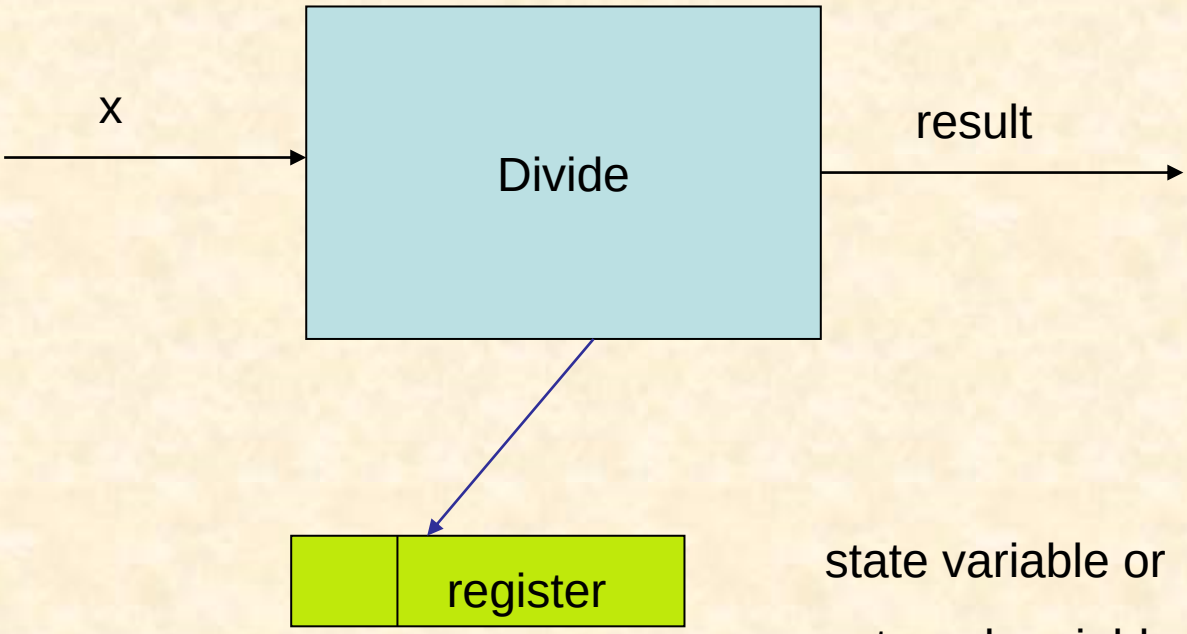
(1) "Systematic Software Development Using VDM", by Cliff B. Jones, 2nd edition, Prentice Hall,1990.

(2) "Modelling Systems", by John Fitzgerald and Peter Gorm Larsen, Cambridge University Press,1998.

The major feature of VDM-SL is the technique for operation specification.

What is an operation?

An example of operation: Divide



Operation specification:

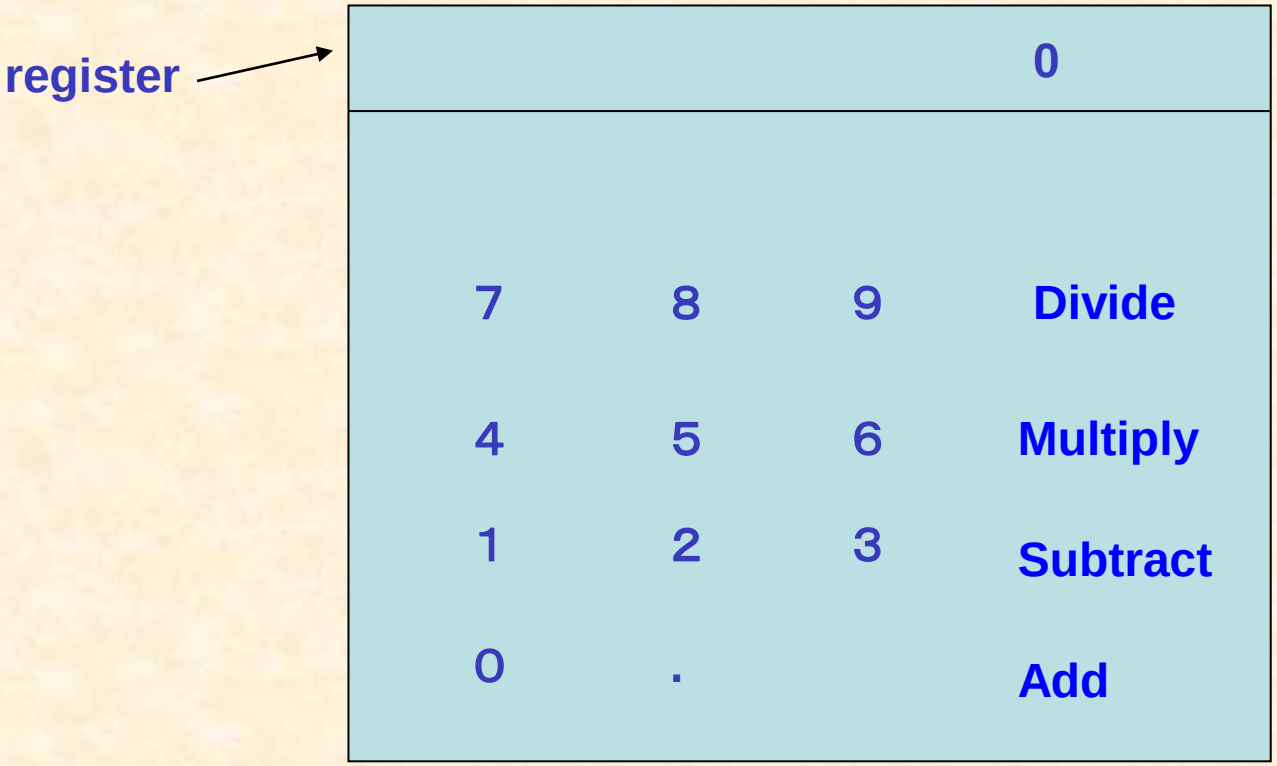
OperationName(input)output
ext State variables
pre pre-condition
post post-condition

Example:

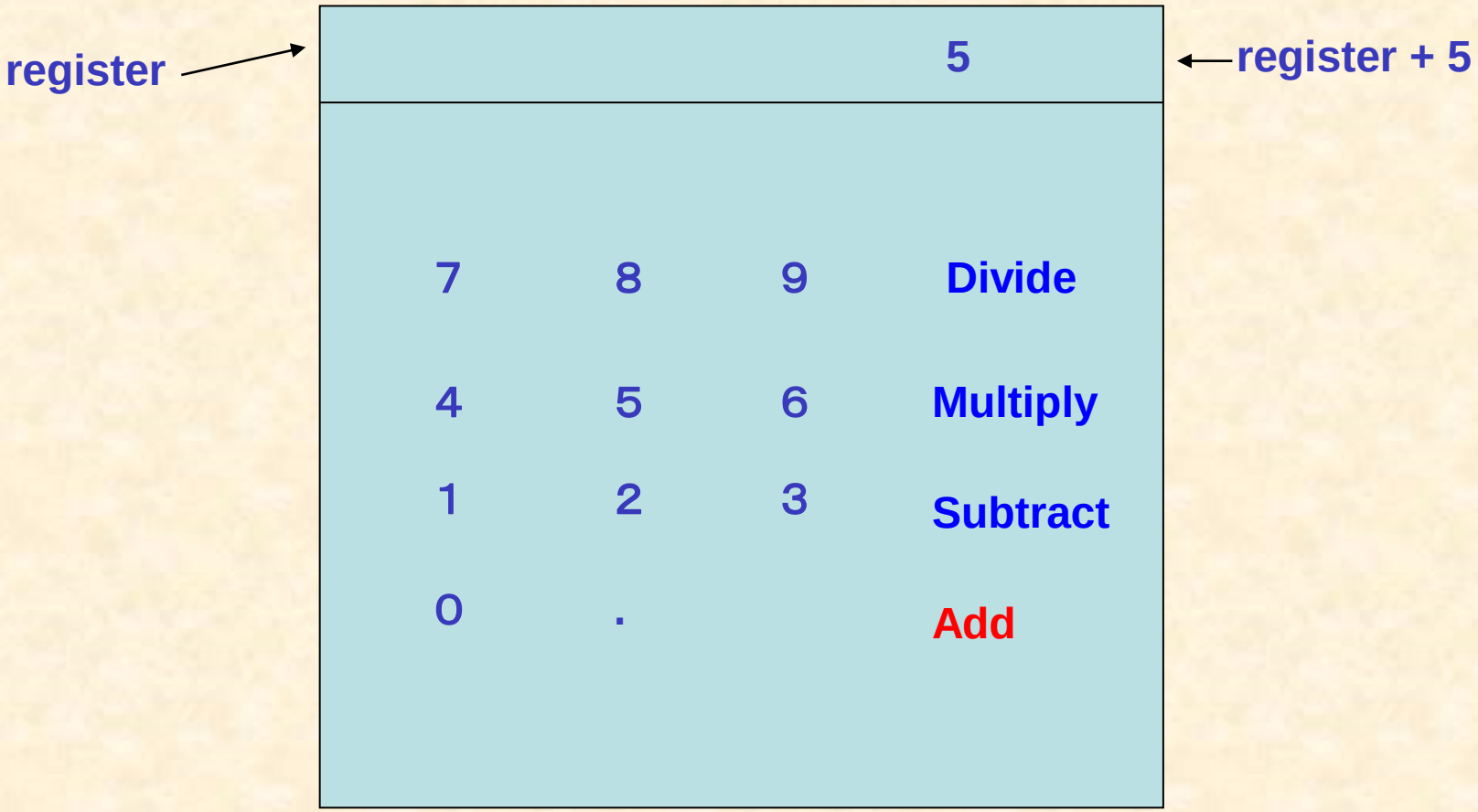
Divide(x: int) result: real
ext wr register: real
pre x <> 0
post register = register~ / x
and
result = register

The difference and similarity between a VDM operation specification and program

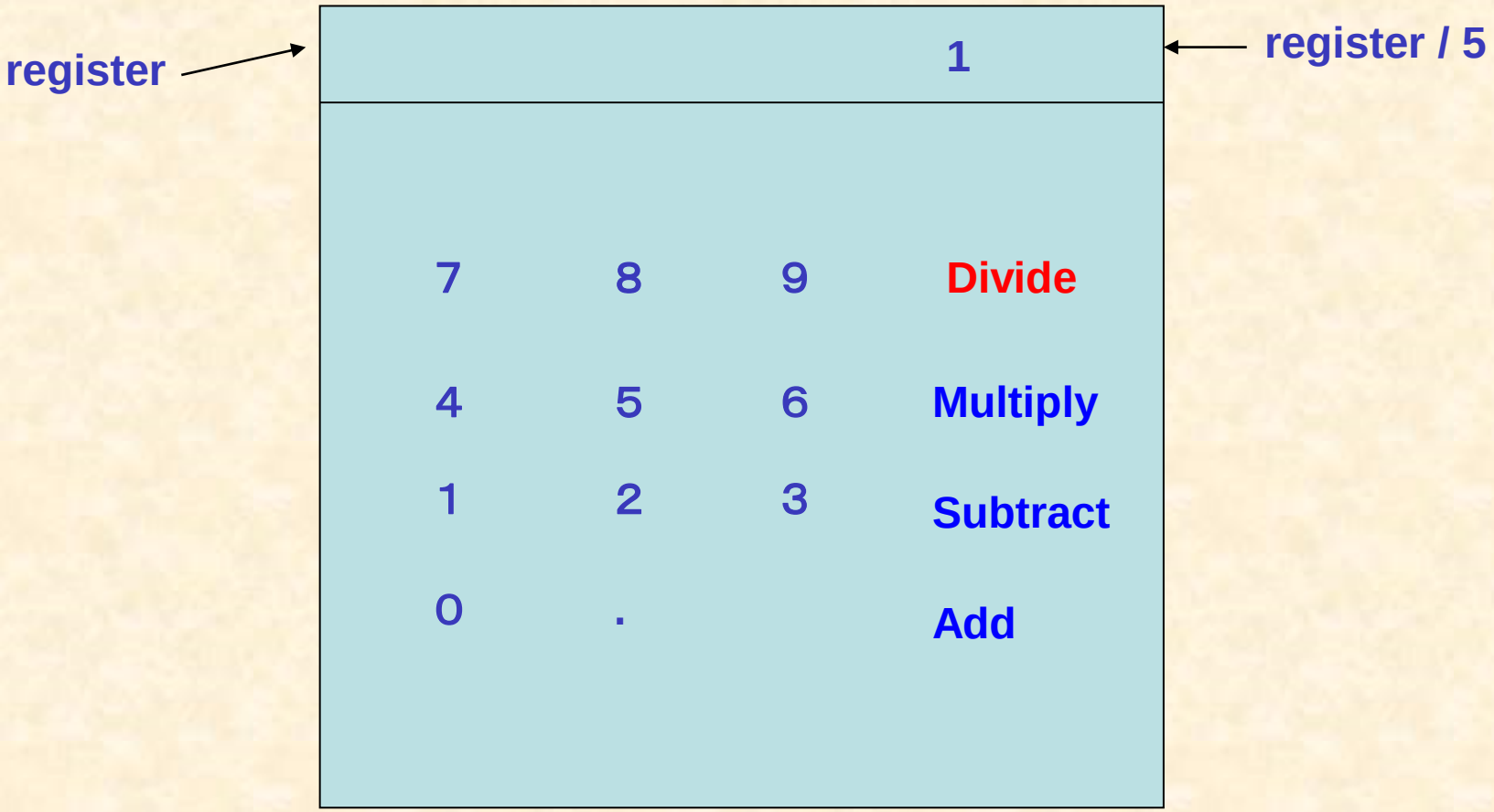
An example of a simplified calculator:



The difference and similarity between a VDM operation specification and program



The difference and similarity between a VDM operation specification and program



The difference and similarity between a VDM operation specification and program

Let's consider the following program:

```
import …;

class Calculator {
 int  register := 0;

int Add(int x) {
 register := register + x;
 return register;
}

double Divide(int x) {
 register := register / x;
 return register;
}
……
}
```

```
main(String arg[]) {
 int input;
 double divisionResult;
 Calculator myCal :=
 new Calculator();
 read(input); //reading from GUI
 divisionResult :=
        myCal.Divide(input);
 System.out.println("Division result:" +
divisionResult);
} //end of the main method
```

The difference and similarity between a VDM operation specification and program

The specification defines the relation between input and output, while the program defines the process of computing the output from the input.

```
import …;

class Calculator {
 int  register := 0;

……

double Divide(int x)
pre x != 0
{
 register := register / x;
 return register;
}
post register = register~ / x and
        result = register
}
```

```
main(String arg[]) {
 int input;
 double divisionResult;
 Calculator myCal :=
 new Calculator();
 read(input); //reading from GUI
 divisionResult :=
     myCal.Divide(input);
 System.out.println("Division result:" +
divisionResult);
} //end of the main method
```

The difference and similarity between a VDM operation specification and program

Omit the program and only use the specification to define the function of the program.

```
import …;

class Calculator {
 int  register := 0;

……

double Divide(int x)
pre x != 0
post register~ / x = register
      and
      register = result
}
```

```
main(String arg[]) {
 int input;
 double divisionResult;
 Calculator myCal :=
 new Calculator();
 read(input); //reading from GUI
 if (input != 0) {
 divisionResult := myCal.Divide(input);
 System.out.println("Division result:" +
divisionResult);}
 else
 System.out.println("The denominator is zero.");
} //the end of the main method.
```

The difference and similarity between a VDM operation specification and program

Specification in VDM-SL:

```
module Calculator

……

Divide(x: int) result: real
ext wr register: real
pre x <> 0
post register = register~ / x
      and
      result = register
```

```
main(String arg[]) {
 int input;
 double divisionResult;
 Calculator myCal :=
 new Calculator();
 read(input); //reading from GUI
 if (input != 0) {
 divisionResult := myCal.Divide(input);
 System.out.println("Division result:" +
divisionResult);}
 else
 System.out.println("The denominator is zero.");
} //the end of the main method.
```

(2) Z, PRG (Programming Research Group),
Oxford University, UK

A Z specification is composed of a set of schemas and possibly their sequential compositions. A schema can be used to define global variables, state variables, and operations.

References:

(1) "The Z Notation", by J.M. Spivey,
Prentice Hall, 1989.

(2) "Using Z: Specification, Refinement, and Proof'', by Jim Woodcock and Jim Davies,
Prentice Hall, 1996.

(3) B-Method,

Jean-Raymond Abrial, France

Reference:

(1) “The B-Book: Assigning Programs to Meanings”, by J-R Abrial, Cambridge University Press,1996,

A B specification is composed of a set of related abstract machines. Each abstract machine is a module that contains a set of operation definitions. Each operation is defined using pre- and post-conditions.

Class dicsussion

(1) Tell the difference between the assignment

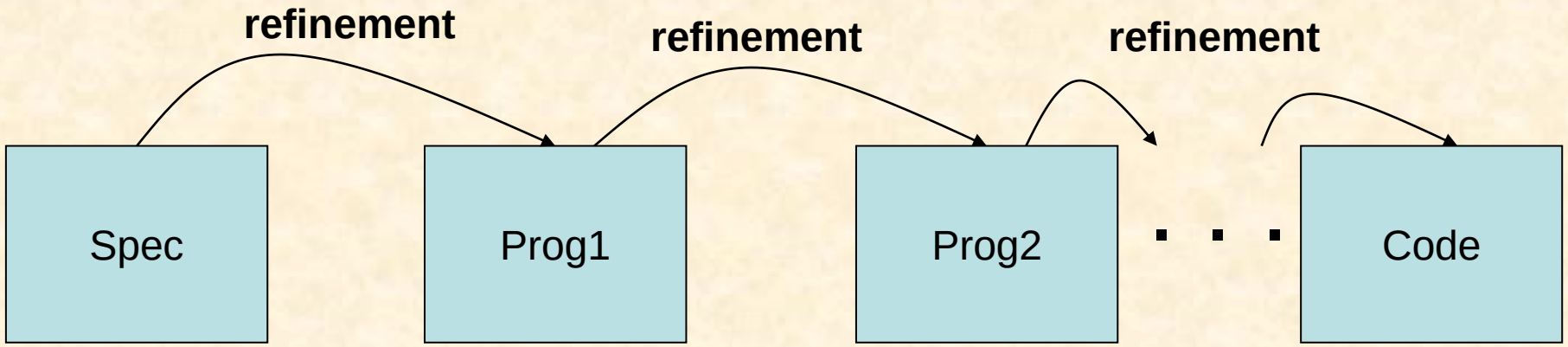
x := y + z (or x = y + z in Java)

in a program and the equality in mathematics:

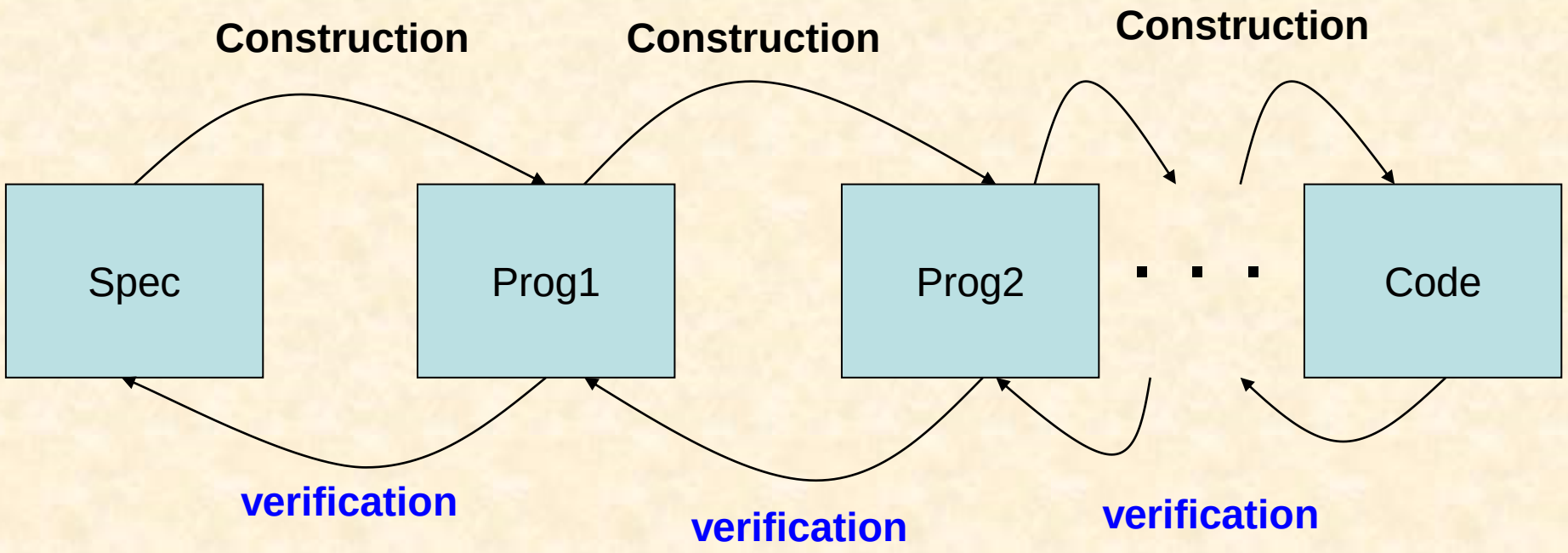
x = y + z

(2) What is the relation between them?

Software development by refinement



Software development based on formal verification



Application examples of formal methods

1 The company Praxis in the UK has used VDM, CSP, and Z to develop a toolset for the **SSADM** development method.

2. The IBM Hursley in the UK has collaborated with the PRG of Oxford university to apply Z to the development of a **Customer Information Control System**（CICS）.

3. The Darlington Nuclear Generator System (Ontario Hydro) in Canada has used Parnas' SCR (Software Cost Reduction) to verify the Shutdown System.

4. The company CSK in Japan has used VDM to develop a stock processing system.

5. The company FeliCa has applied VDM++ to the development of a mobile phone IC chip.

Challenges for formal methods

- The complexity of formal specifications grows rapidly as their scale increases. This makes evolution of specifications, which is an necessary activity in real projects, very hard and costly.
- Formal methods only offer notations and rules, but not tell how each kind of technique (e.g., specification, refinement, verification) can be effectively applied in the context of practical software development.

- Formal verification for program correctness can only be applied to small and simple programs. There is a lack of effective techniques for dealing with large scale systems.
- The tool support is not good enough to make formal methods practical in industry.

- There are many practical constraints as well.

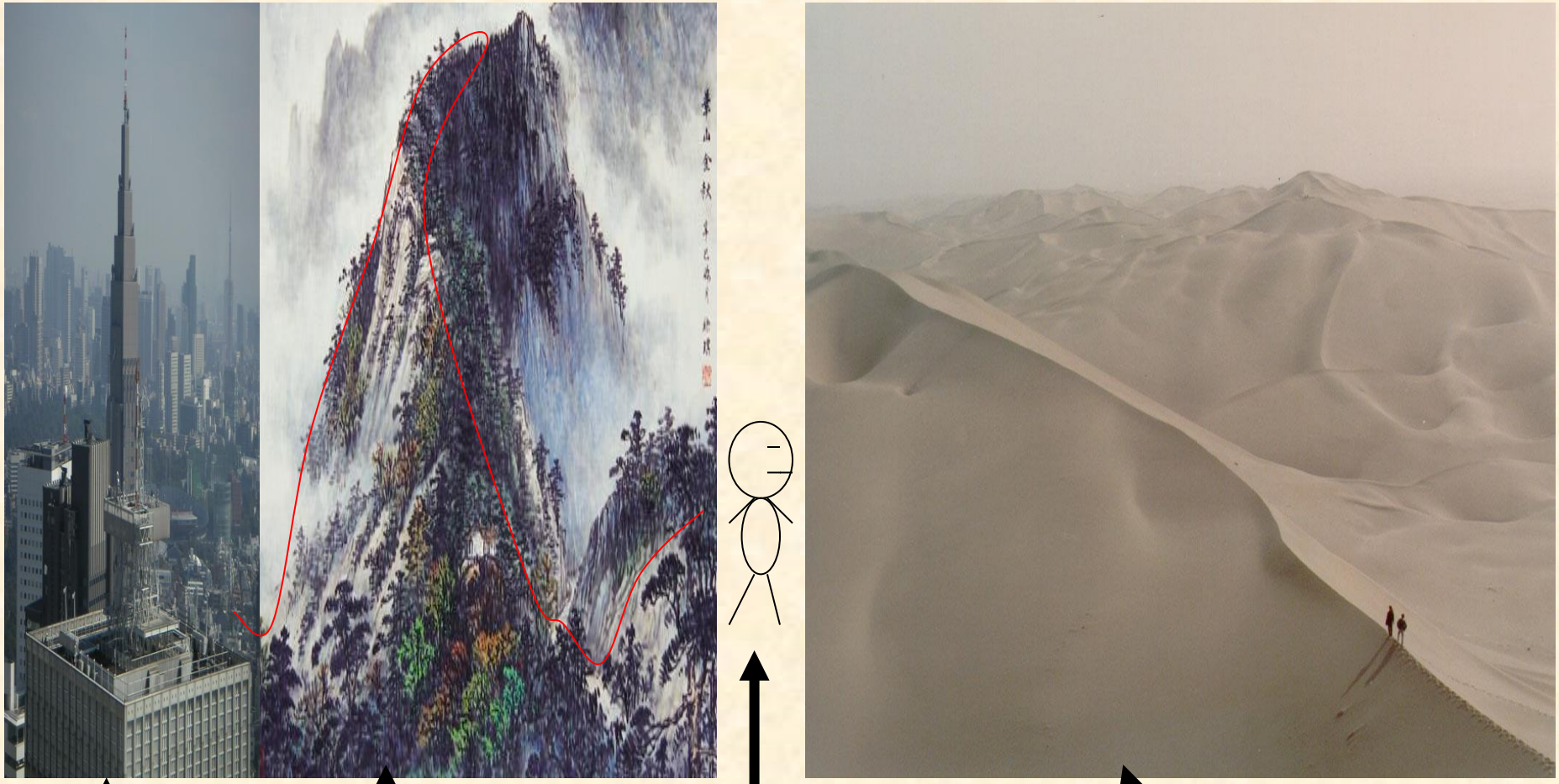
Examples:

(1) Practitioners may lack the skills for abstraction in writing formal specifications.

(2) Managers may not want to introduce formal methods before seeing hard evidence of the effectiveness of formal methods.

(3) Budget, schedule, and company's environment may not allow practitioners to use formal methods.

Software Practitioners' dilemma



Happy world

Formal Methods

Software practitioner

Traditional SE

I.3 Formal Engineering Methods for Practicality

Formal Engineering Methods (FEM) provide ways to integrate Formal Methods into the commonly used software engineering methods and approaches to improve their rigor, comprehensibility, effectiveness, and tool supportability for software productivity and quality.

Formal Methods

Application of Formal Methods in Software Engineering

Formal Engineering Methods

The difference between FM and FEM

FM answers the question:

what should we do and why?

FEM answers the question:

what can we do and how?

I.4 SOFL

SOFL stands for Structured Object-Oriented Formal Language

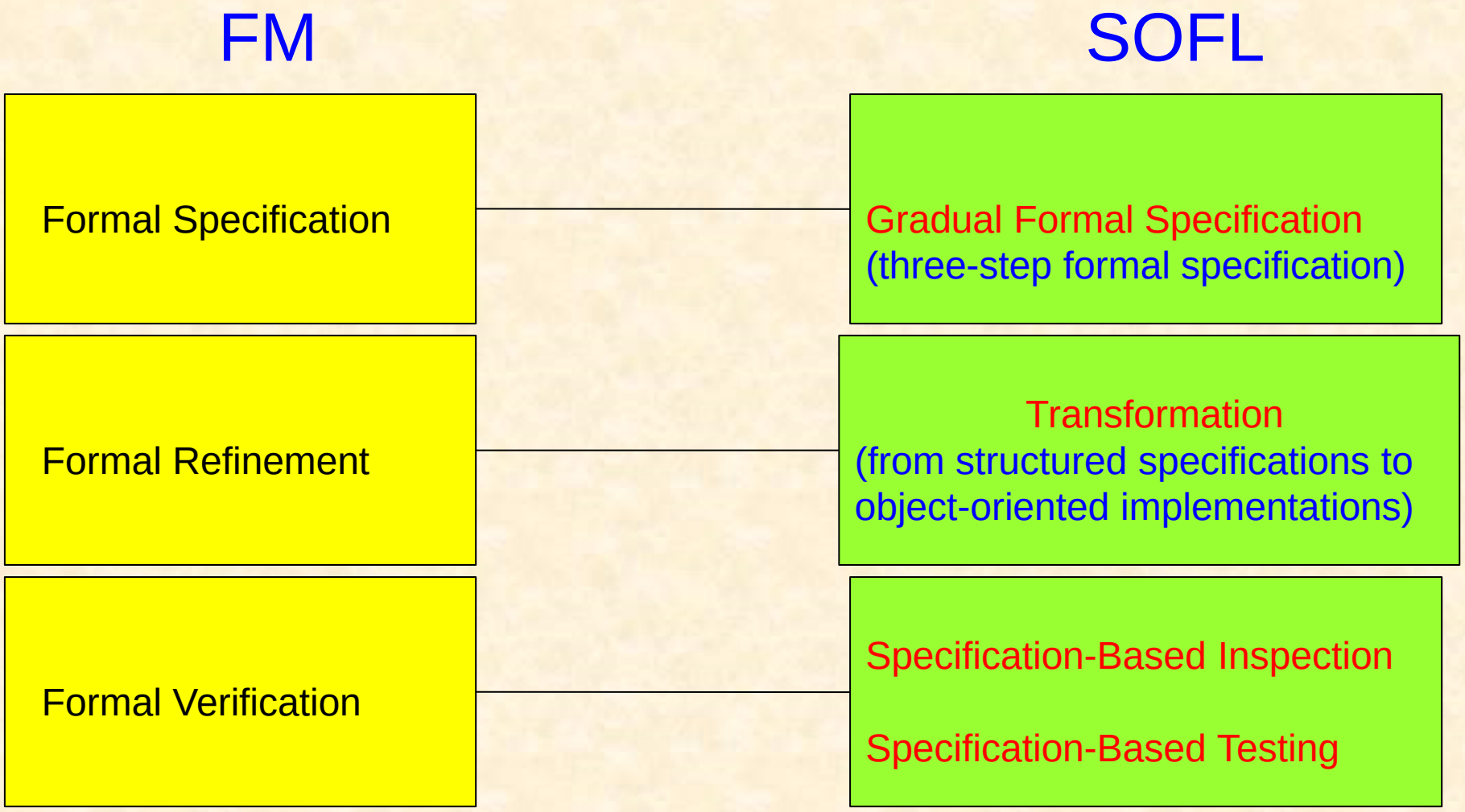
SOFL method is a specific and representative formal engineering method. The research on it started at the University of Manchester, UK in 1989.

Completed at Hiroshima City University in 1998.

Finalized at Hosei University in 2002.

Further developments of SOFL technologies after 2002.

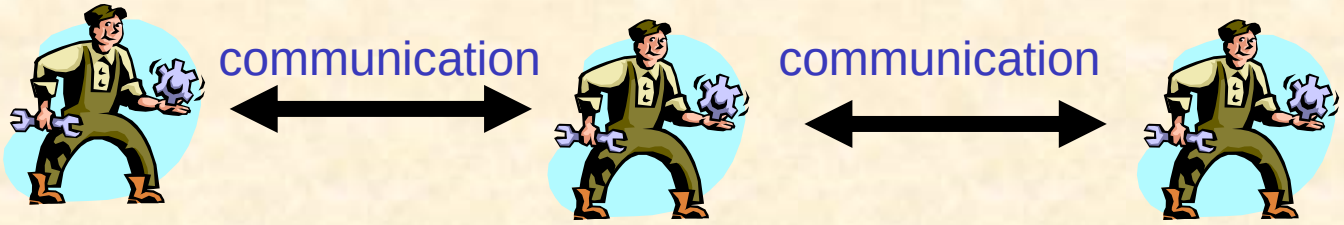
Comparison between FM and the SOFL method



The feature of SOFL

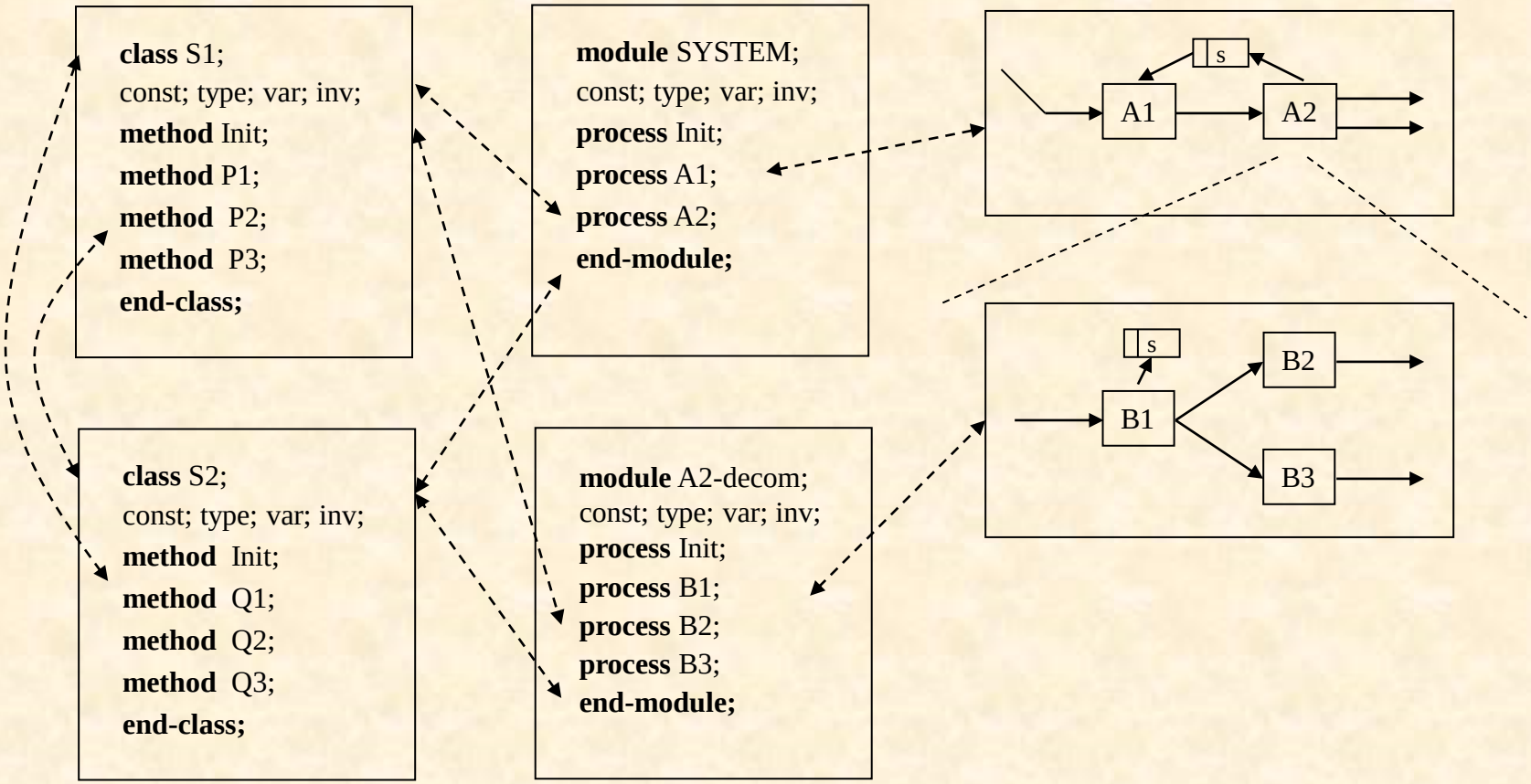
The SOFL method strikes a good balance among the three qualities:

Simplicity, Visualization, Precision



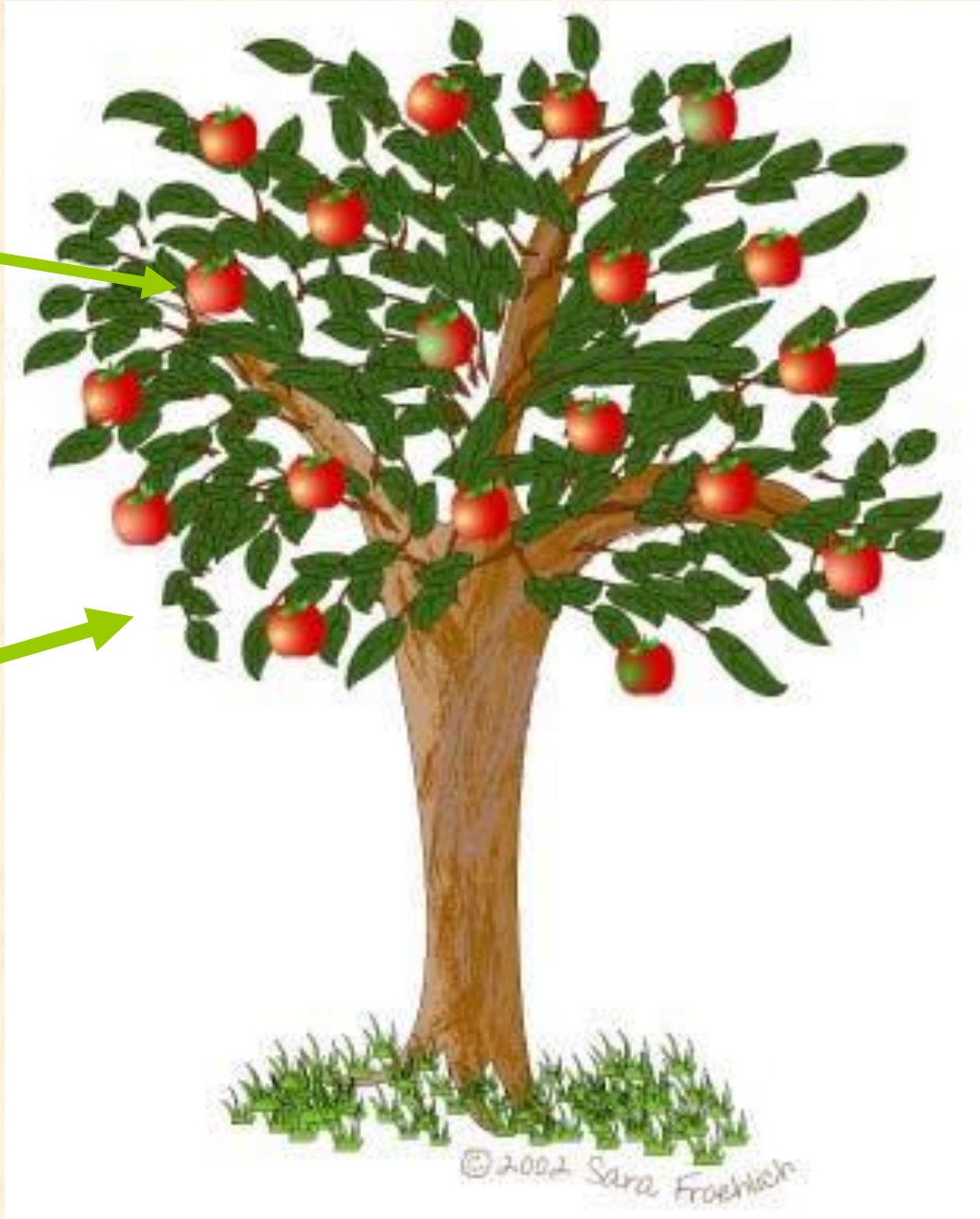
The structure of a SOFL specification:

CDFDs + modules + classes



Component

Architecture



Basic Components of the SOFL Specification Language

- The SOFL logic
- Module
- Condition Data Flow Diagrams
- Process specification
- Function definition and specification
- Data types
- Process decomposition
- Other issues