Formal Specification

- Techniques for the unambiguous specification of software

Objectives

- To explain why formal specification techniques help discover problems in system requirements
- To describe the use of algebraic techniques for interface specification
- To describe the use of model-based techniques for behavioural specification

Topics covered

- Formal specification in the software process
- Interface specification
- Behavioural specification

Formal methods

- Formal specification is part of a more general collection of techniques that are known as 'formal methods'
- These are all based on mathematical representation and analysis of software
- Formal methods include
  - Formal specification
  - Specification analysis and proof
  - Transformational development
  - Program verification

Acceptance of formal methods

- Formal methods have not become mainstream software development techniques as was once predicted
  - Other software engineering techniques have been successful at increasing system quality.
  - Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market
  - Formal methods are hard to scale up to large systems

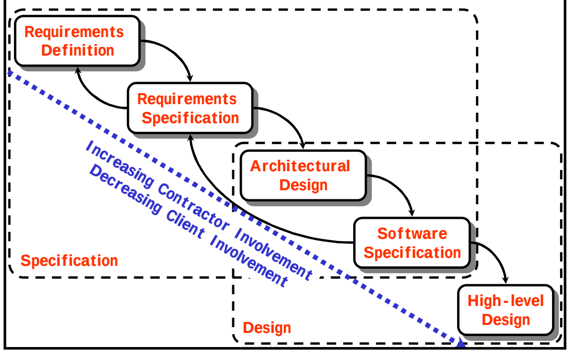
Use of formal methods

- Their principal benefits are in reducing the number of errors in systems so their main area of applicability is critical systems
- In this area, the use of formal methods is most likely to be cost-effective

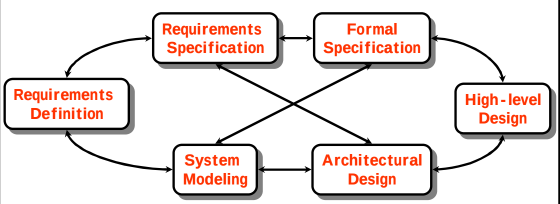
Specification in the software process

- Specification and design are intermingled.
- Architectural design is essential to structure a specification.
- Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

Specification and design



Specification in the software process



Specification techniques

- Algebraic approach
  - The system is specified in terms of its operations and their relationships
- Model-based approach
  - The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences. Operations are defined by modifications to the system's state

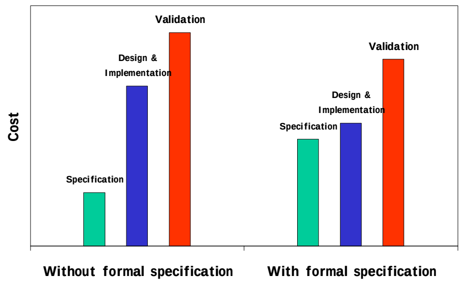
Formal specification languages

	Sequential	Concurrent
Algebraic	Larch	Lotos
Model-based	1. Z 2. VDM 3. B	1. CSP 2. Petri Nets

Use of formal specification

- Formal specification involves investing more effort in the early phases of software development
- This reduces requirements errors as it forces a detailed analysis of the requirements
- Incompleteness and inconsistencies can be discovered and resolved
- Hence, savings as made as the amount of rework due to requirements problems is reduced

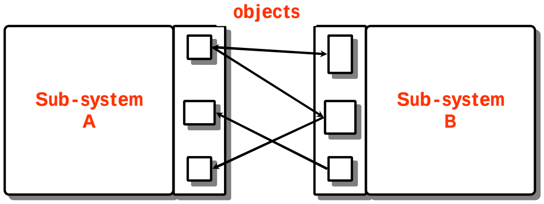
Development costs with formal specification



Interface specification

- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems
- Specification of subsystem interfaces allows independent development of the different subsystems
- Interfaces may be defined as abstract data types or object classes
- The algebraic approach to formal specification is particularly well-suited to interface specification

Sub-system interfaces



The structure of an algebraic specification

<Specification Name> (Generic Parameter)

Sort <name> **Imports** <list of specification names>
Informal description of the sort and its operations
Operation signatures setting out the names and the types of the parameters to the operations defined over the sort
Axioms defining the operations over the sort

Specification components

- Introduction
  - Defines the sort (the type name) and declares other specifications that are used
- Description
  - Informally describes the operations on the type
- Signature
  - Defines the syntax of the operations in the interface and their parameters
- Axioms
  - Defines the operation semantics by defining axioms which characterise behaviour

Systematic algebraic specification

- Algebraic specifications of a system may be developed in a systematic way
  - Specification structuring.
  - Specification naming.
  - Operation selection.
  - Informal operation specification
  - Syntax definition
  - Axiom definition

Specification operations

- Constructor operations. Operations which create entities of the type being specified
- Inspection operations. Operations which evaluate entities of the type being specified
- To specify behaviour, define the inspector operations for each constructor operation

Interface specification in critical systems

- Consider an air traffic control system where aircraft fly through managed sectors of airspace
- Each sector may include a number of aircraft but, for safety reasons, these must be separated
- In this example, a simple vertical separation of 300m is proposed
- The system should warn the controller if aircraft are instructed to move so that the separation rule is breached

A sector object

- Critical operations on an object representing a controlled sector are
  - Enter. Add an aircraft to the controlled airspace
  - Leave. Remove an aircraft from the controlled airspace
  - Move. Move an aircraft from one height to another
  - Lookup. Given an aircraft identifier, return its current height

Primitive operations

- It is sometimes necessary to introduce additional operations to simplify the specification
- The other operations can then be defined using these more primitive operations
- Primitive operations
  - Create. Bring an instance of a sector into existence
  - Put. Add an aircraft without safety checks
  - In-space. Determine if a given aircraft is in the sector
  - Occupied. Given a height, determine if there is an aircraft within 300m of that height

Sector specification

```
SECTOR
sort Sector
imports INTEGER, BOOLEAN

Enter - adds an aircraft to the sector if safety conditions are satisfed
Leave - removes an aircraft from the sector
Move - moves an aircraft from one height to another if safe to do so
Lookup - Finds the height of an aircraft in the sector

Create - creates an empty sector
Put - adds an aircraft to a sector with no constraint checks
In-space - checks if an aircraft is already in a sector
Occupied - checks if a specified height is available

Enter (Sector, Call-sign, Height) → Sector
Leave (Sector, Call-sign) → Sector
Move (Sector, Call-sign, Height) → Sector
Lookup (Sector, Call-sign) → Height

Create → Sector
Put (Sector, Call-sign, Height) → Sector
In-space (Sector, Call-sign) → Boolean
Occupied (Sector, Height) → Boolean

Enter (S, CS, H) =
   if     In-space (S, CS ) then S exception (Aircraft already in sector)
   elsif  Occupied (S, H) then S exception (Height conflict)
   else   Put (S, CS, H)

Leave (Create, CS) = Create exception (Aircraft not in sector)
Leave (Put (S, CS1, H1), CS) =
     if CS = CS1 then S else Put (Leave (S, CS), CS1, H1)

Move (S, CS, H) =
   if     S = Create then Create exception (No aircraft in sector)
   elsif  not In-space (S, CS) then S exception (Aircraft not in sector)
   elsif  Occupied (S, H) then S exception (Height conflict)
   else   Put (Leave (S, CS), CS, H)

-- NO-HEIGHT is a constant indicating that a valid height cannot be returned

Lookup (Create, CS) = NO-HEIGHT exception (Aircraft not in sector)
Lookup (Put (S, CS1, H1), CS) =
    if CS = CS1 then H1 else Lookup (S, CS)

Occupied (Create, H) = false
Occupied (Put (S, CS1, H1), H) =
    if    (H1 > H and H1 - H ≤ 300) or (H > H1 and H - H1 ≤ 300) then true
    else  Occupied (S, H)

In-space (Create, CS) = false
In-space (Put (S, CS1, H1), CS ) =
    if CS = CS1 then true else In-space (S, CS)
```

Specification commentary

- Use the basic constructors Create and Put to specify other operations
- Define Occupied and In-space using Create and Put and use them to make checks in other operation definitions
- All operations that result in changes to the sector must check that the safety criterion holds