

Formal Specification

- ⊗ Techniques for the unambiguous specification of software

Objectives

- ⊗ To explain the place of formal software specification in the software process.
- ⊗ To explain when formal specification is cost-effective.
- ⊗ To describe a process model based on the transformation of formal specifications to an executable system.
- ⊗ To introduce a simple approach to formal specification based on pre and post conditions

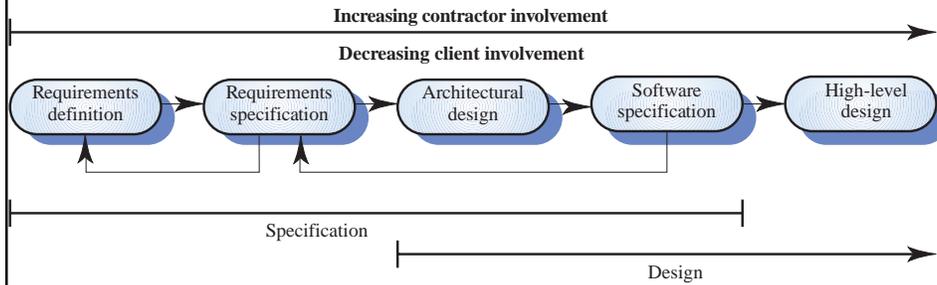
Topics covered

- ⊗ Formal specification on trial
- ⊗ Transformational development
- ⊗ Specifying functional abstractions

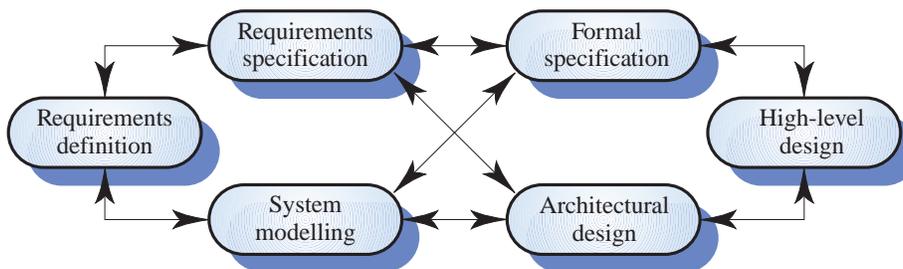
Specification in the software process

- ⊗ Specification and design are inextricably 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



Formal specification on trial

- ⊗ Formal techniques are not widely used in industrial software development
- ⊗ Given the relevance of mathematics in other engineering disciplines, why is this the case?

Why aren't formal methods used?

- ⊗ Inherent management conservatism. It is hard to demonstrate the advantages of formal specification in an objective way
- ⊗ Many software engineers lack the training in discrete math necessary for formal specification
- ⊗ System customers may be unwilling to fund specification activities
- ⊗ Some classes of software (particularly interactive systems and concurrent systems) are difficult to specify using current techniques

Why aren't formal methods used?

- ⊗ There is widespread ignorance of the applicability of formal specifications
- ⊗ There is little tool support available for formal notations
- ⊗ Some computer scientists who are familiar with formal methods lack knowledge of the real-world problems to which these may be applied and therefore oversell the technique

Advantages of formal specification

- ⊗ It provides insights into the software requirements and the design.
- ⊗ Formal specifications may be analyzed mathematically and the consistency and completeness of the specification demonstrated. It may be possible to prove that the implementation corresponds to the specification

Advantages of formal specifications

- ⊗ Formal specifications may be used to guide the tester of the component in identifying appropriate test cases
- ⊗ Formal specifications may be processed using software tools. It may be possible to animate the specification to provide a software prototype

Seven myths of formal methods

- ⊗ Perfect software results from formal methods
 - Nonsense - the formal specification is a model of the real-world and may incorporate misunderstandings, errors and omissions.
- ⊗ Formal methods means program proving
 - Formally specifying a system is valuable without formal program verification as it forces a detailed analysis early in the development process.
- ⊗ Formal methods can only be justified for safety-critical systems.
 - Industrial experience suggests that the development costs for all classes of system are reduced by using formal specification.

Seven myths of formal methods

- ⊗ Formal methods are for mathematicians
 - Nonsense - only simple math is needed.
- ⊗ Formal methods increase development costs
 - Not proven. However, formal methods definitely push development costs towards the front-end of the life cycle.
- ⊗ Clients cannot understand formal specifications
 - They can if they are paraphrased in natural language.
- ⊗ Formal methods have only been used for trivial systems
 - There are now many published examples of experience with formal methods for non-trivial software systems.

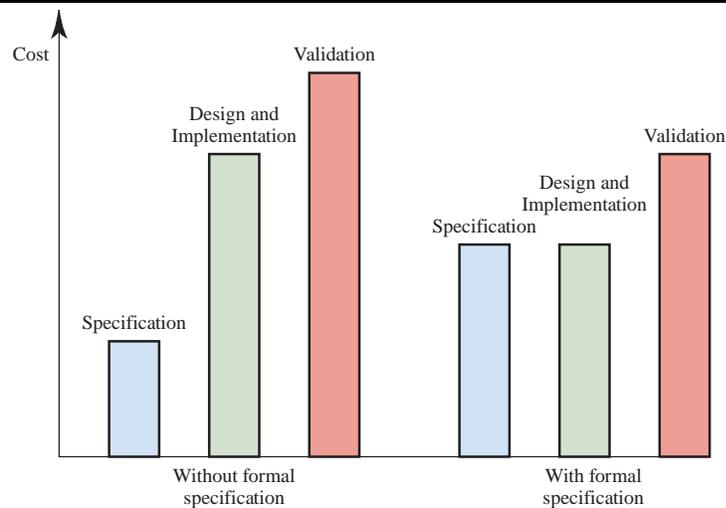
The verdict!

- ⊗ The reasons put forward for not using formal specifications and methods are weak
- ⊗ However, there are good reasons why these methods are not used
 - The move to interactive systems. Formal specification techniques cannot cope effectively with graphical user interface specification
 - Successful software engineering. Investing in other software engineering techniques may be more cost-effective

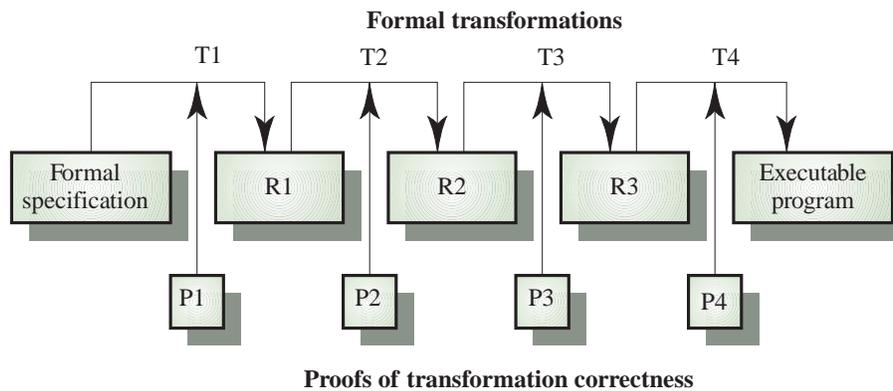
Use of formal methods

- ⊗ These methods are unlikely to be widely used in the foreseeable future. Nor are they likely to be cost-effective for most classes of system
- ⊗ They will become the normal approach to the development of safety critical systems and standards
- ⊗ This changes the expenditure profile through the software process

Development costs with formal specification



Transformational development



Specifying functional abstractions

- ⊗ The simplest specification is function specification. There is no need to be concerned with global state (assuming no side-effects)
- ⊗ The formal specification is expressed as input and output predicates (pre and post conditions)
- ⊗ Predicates are logical expressions which are always either true or false
- ⊗ Predicate operators include the usual logical operators and quantifiers such as for-all and exists

Examples of predicates

All variables referenced are of type INTEGER

1. The value of variable A is greater than the value of B and the value of variable C is greater than D

A > B and C > D

2. This predicate illustrates the use of the exists quantifier. The predicate is true if there are values of i, j and k between M and N such that $i^2 = j^2 + k^2$. Thus, if M is 1 and N is 5, the predicate is true as $3^2 + 4^2 = 5^2$. If M is 6 and N is 9, the predicate is false. There are no values of i, j and k between 6 and 9 which satisfy the condition.

exists i, j, k in M..N: $i^2 = j^2 + k^2$

3. This predicate illustrates the use of the universal quantifier for_all. It concerns the values of an array called Squares. It is true if the first ten values in the array take a value which is the square of an integer between 1 and 10.

for_all i in 1..10, exists j in 1..10: Squares (i) = j^2

Specification with pre and post conditions

- ⊗ Set out the pre-conditions
 - A statement about the function parameters stating what is invariably true before the function is executed
- ⊗ Set out the post-conditions
 - A statement about the function parameters stating what is invariably true after the function has executed
- ⊗ The difference between the pre and post conditions is due to the application of the function to its parameters. Together the pre and post conditions are a function specification

Specification development

- ⊗ Establish the bounds of the input parameters.
Specify this as a predicate
- ⊗ Specify a predicate defining the condition which must hold on the result of the function if it computes correctly
- ⊗ Establish what changes are made to the input parameters by the function and specify these as a predicate
- ⊗ Combine the predicates into pre and post conditions

The specification of a search

```
function Search ( X: in INTEGER_ARRAY ; Key: INTEGER )  
    return INTEGER ;
```

Pre: **exists** i **in** X'FIRST..X'LAST: X(i) = Key

Post: X" (Search (X, Key)) = Key and X = X"

Search pre-conditions

- ⊗ One of the array elements must match the key
- ⊗ Use the exists quantifier to specify that an element must exist which matches the key
 - **exists** i in X 'FIRST.. X 'LAST: $X(i) = \text{Key}$
- ⊗ Assume **FIRST** and **LAST** refer to the upper and lower bounds of the array

Search post-conditions

- ⊗ The result of Search should be the value of the array index which refers to the element containing the key
 - $X''(\text{Search}(X, \text{Key})) = \text{Key}$
- ⊗ The array after the operation is referenced by 'priming' the array name
- ⊗ The array should not be changed by the Search function
 - $X = X''$

Specification with error predicate

function Search (X: **in** INTEGER_ARRAY ; Key: INTEGER)
 return INTEGER ;

Pre: $\exists i \text{ in } X.\text{FIRST}..\text{X}.\text{LAST}: X(i) = \text{Key}$

Post: $X'' (\text{Search}(X, \text{Key})) = \text{Key} \text{ and } X = X''$

Error: $\text{Search}(X, \text{Key}) = \text{X}.\text{LAST} + 1$

Formal specification approaches

- ⊗ Algebraic approach
 - The system is described in terms of interface operations and their relationships
- ⊗ Model-based approach
 - A model of the system acts as a specification. This model is constructed using well-understood mathematical entities such as sets and sequences
- ⊗ These are covered in the following two chapters

Formal specification languages

	Sequential	Concurrent
Algebraic	Larch (Gutttag et al., 1985) OBJ (Futatsugi et al., 1985)	Lotos (Bolognesi and Brinksma, 1987)
Model-based	Z (Spivey, 1989) VDM (Jones, 1980)	CSP (Hoare, 1985) Petri Nets (Peterson, 1981)

Key points

- ⊗ Formal system specification complements informal specification techniques
- ⊗ Formal specifications are precise and unambiguous. They remove areas of doubt in a specification
- ⊗ Formal specification forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system

Key points

- ⊗ Formal specification techniques are not cost-effective for the development of interactive systems. They are most applicable in the development of safety-critical systems and standards.
- ⊗ Functions can be specified by setting out pre and post conditions for the function. However, this approach does not scale up to large or medium-sized systems.