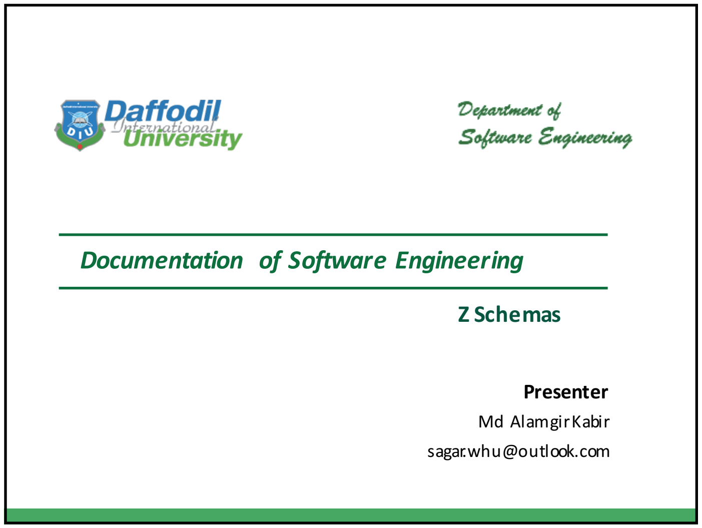


Department of Software Engineering

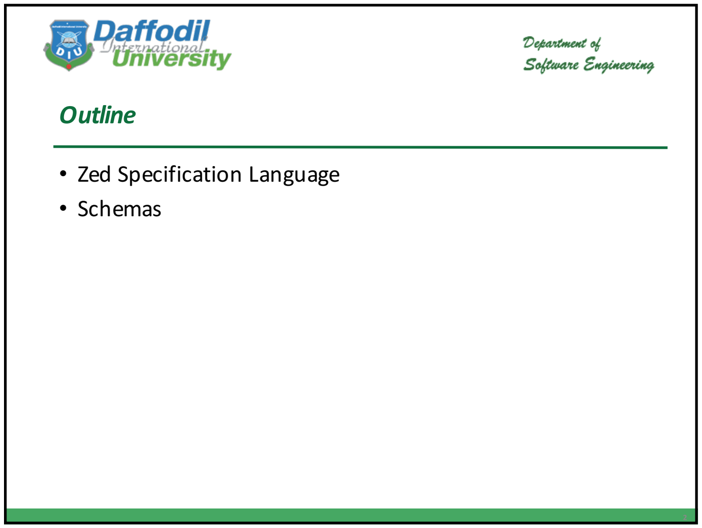
Documentation of Software Engineering

Z Schemas

Presenter

Md Alamgir Kabir

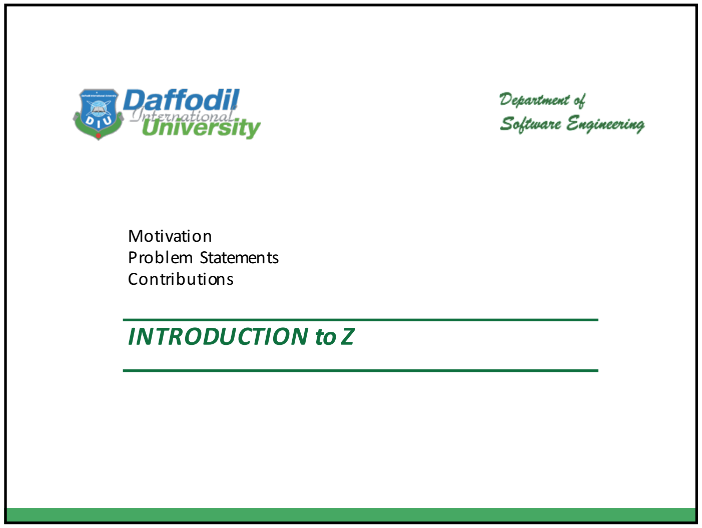
sagar.whu@outlook.com



Department of Software Engineering

Outline

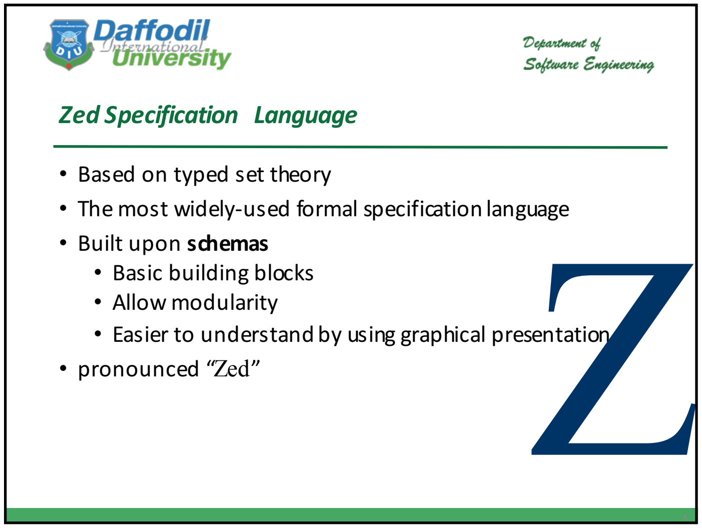
- Zed Specification Language
- Schemas



Department of Software Engineering

Motivation
Problem Statements
Contributions

INTRODUCTION to Z

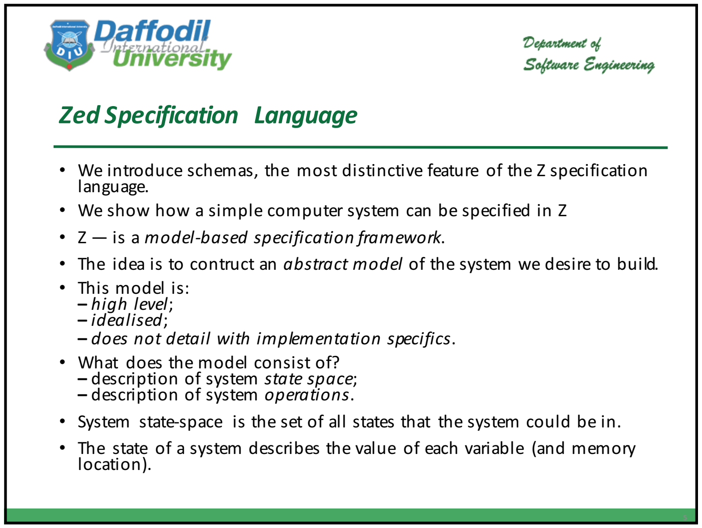


Department of Software Engineering

Zed Specification Language

- Based on typed set theory
- The most widely-used formal specification language
- Built upon **schemas**
  - Basic building blocks
  - Allow modularity
  - Easier to understand by using graphical presentation
- pronounced "Zed"

Z

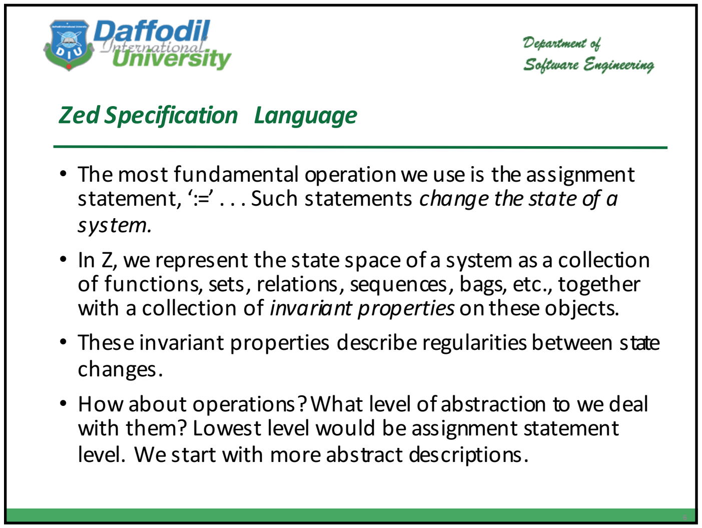


Daffodil International University

Department of Software Engineering

Zed Specification Language

- We introduce schemas, the most distinctive feature of the Z specification language.
- We show how a simple computer system can be specified in Z
- Z — is a *model-based specification framework*.
- The idea is to contruct an *abstract model* of the system we desire to build.
- This model is:
  - *high level*;
  - *idealised*;
  - *does not detail with implementation specifics*.
- What does the model consist of?
  - description of system *state space*;
  - description of system *operations*.
- System state-space is the set of all states that the system could be in.
- The state of a system describes the value of each variable (and memory location).

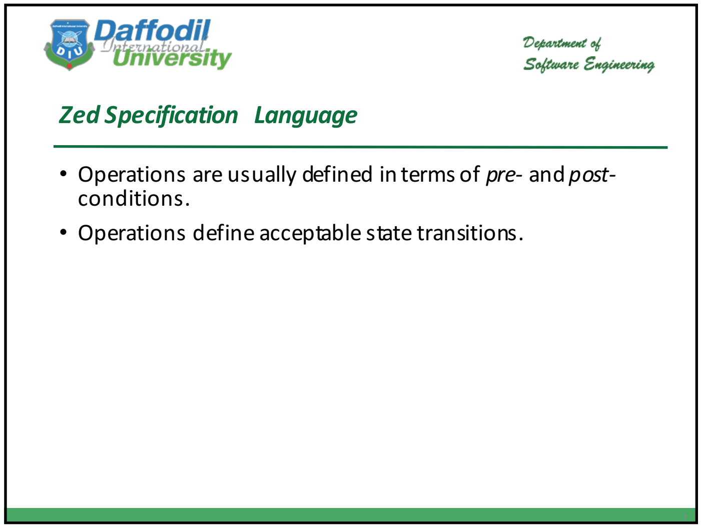


Daffodil International University

Department of Software Engineering

Zed Specification Language

- The most fundamental operation we use is the assignment statement, ':=' . . . Such statements *change the state of a system.*
- In Z, we represent the state space of a system as a collection of functions, sets, relations, sequences, bags, etc., together with a collection of *invariant properties* on these objects.
- These invariant properties describe regularities between state changes.
- How about operations? What level of abstraction to we deal with them? Lowest level would be assignment statement level. We start with more abstract descriptions.

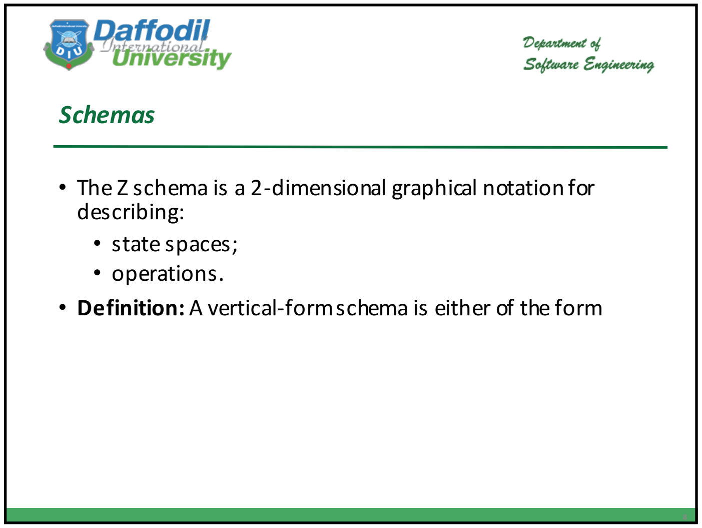


Daffodil International University

Department of Software Engineering

Zed Specification Language

- Operations are usually defined in terms of *pre*- and *post*-conditions.
- Operations define acceptable state transitions.



Daffodil International University

Department of Software Engineering

Schemas

- The Z schema is a 2-dimensional graphical notation for describing:
  - state spaces;
  - operations.
- **Definition:** A vertical-form schema is either of the form

Z Schema

$SchemaName$

$Declarations$

$Predicate_1; \cdots ; Predicate_n$

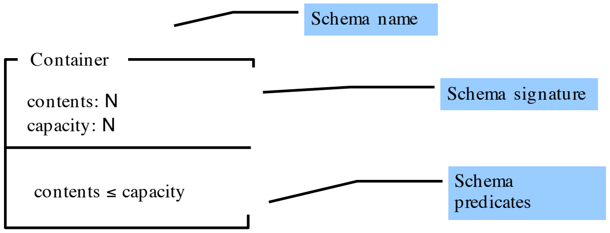
or of the form

$SchemaName$

$Declarations$

In the latter case, the predicate part is assumed to be 'true'.

Z Schema



Container

contents: N
capacity: N

contents ≤ capacity

Schema name

Schema signature

Schema predicates

Schema predicates are always true
Predicates can refer only to elements in the signature

- *SchemaName* will be associated with the schema proper, which is the contents of the box.
- The declarations part of the schema will contain:
  - a list of variable declarations; and
  - references to other schemas (this is called schema inclusion).
- Variable declarations have the usual form
  - $x_1,x_2,\ldots,x_n : T;$
- The predicate part of a schema contains a list of predicates, separated either by semi-colons or new lines.

State Space Schemas

- Here is an example state-space schema, representing part of a system that records details about the phone numbers of staff. (Assume that *NAME* is a set of names, and *PHONE* is a set of phone numbers.)

$PhoneBook$

$known : \mathbb{P}\, NAME$

$tel : NAME \nrightarrow PHONE$

$\mathrm{dom}\ tel = known$

State Space Schemas

- The declarations part of this schema introduces two variables: ***known*** and ***tel***.
- The value of ***known*** will be a subset of *NAME*, i.e., a set of names. [This variable will be used to represent all the names that we know about — those that we can give a phone number for.]
- The value of *tel* will be a partial function from *NAME* to *PHONE*, i.e., it will associate names with phone numbers.
- The domain of *tel* is always

equal to the set *known*.

```
PhoneBook ____________________
known : ℙ NAME
tel : NAME ⇸ PHONE
______________________________
dom tel = known
______________________________
```

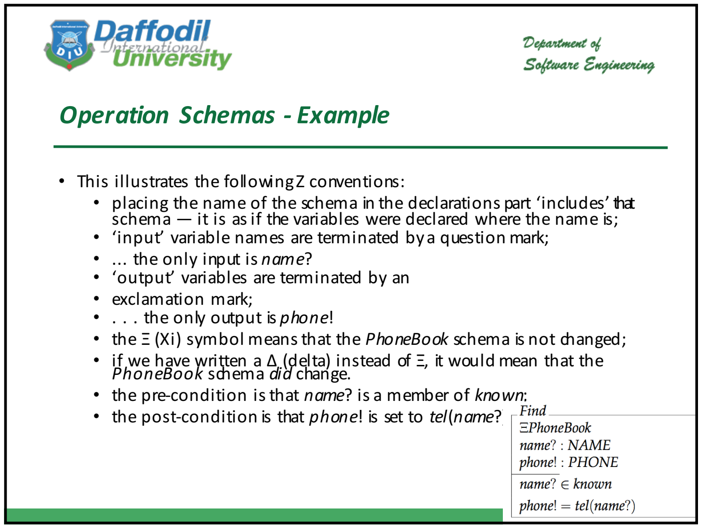
THE FURUTE OF BANKING

Operation Schemas

- In specifying a system operation, we must consider:
  - the objects that are *accessed* by the operation, and of these:
    - the objects that are *known to remain unchanged* by the operation (cf. value parameters);
    - the objects that *may be altered* by the operation (cf. variable parameter);
  - the *pre-conditions* of the operation, i.e., the things that must be true for the operation to succeed;
  - the *post-conditions* — the things that will be true after the operation, if the pre-condition was satisfied before the operation.

Operation Schemas - Example

- Return to the telephone book example, and consider the ‘lookup’ operation: we put a name in, and get a phone number out.
  - this operation accesses the *PhoneBook* schema;
  - it does not change it;
  - it takes a single ‘input’ — a name for which we want to find a phone number;
  - it produces a single output — a phone number.
  - it has the pre-condition that the name is known to the database.



Department of Software Engineering

Operation Schemas - Example

- This illustrates the following Z conventions:
  - placing the name of the schema in the declarations part ‘includes’ that schema — it is as if the variables were declared where the name is;
  - ‘input’ variable names are terminated by a question mark;
  - ... the only input is *name*?
  - ‘output’ variables are terminated by an
  - exclamation mark;
  - . . . the only output is *phone*!
  - the Ξ (Xi) symbol means that the *PhoneBook* schema is not changed;
  - if we have written a Δ (delta) instead of Ξ, it would mean that the *PhoneBook* schema *did* change.
  - the pre-condition is that *name*? is a member of *known*;
  - the post-condition is that *phone*! is set to *tel*(*name*?

Find

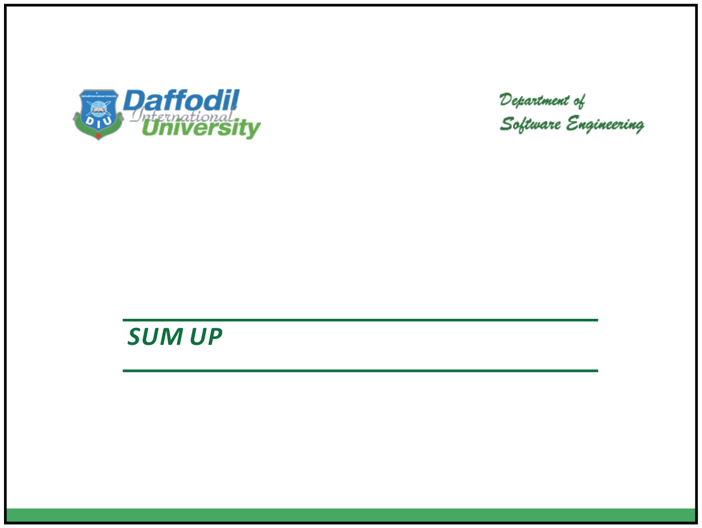
$\Xi PhoneBook$

$name? : NAME$

$phone! : PHONE$

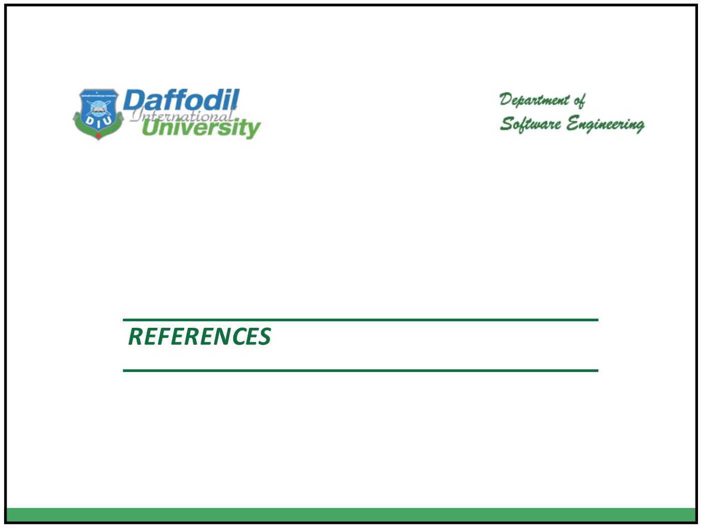
$name? \in known$

$phone! = tel(name?)$



Department of Software Engineering

SUM UP

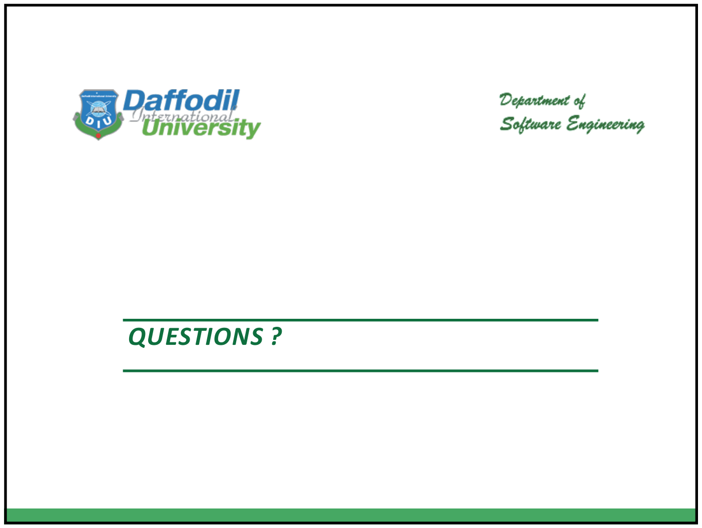


Department of Software Engineering

REFERENCES

Department of Software Engineering

References



Daffodil International University

Department of Software Engineering

QUESTIONS ?