

## Especificação Formal

**Prós e contras da especificação formal**  
**Desenvolvimento transformacional**  
**Especificar abstrações funcionais**  
**Abordagem algébrica**

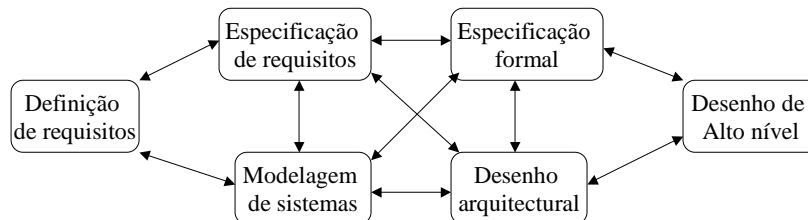
## Especificação Formal

- A maneira mais precisa de expressar uma especificação detalhada é usar uma notação matemática formal.
- Nos estágios iniciais do processo de desenvolvimento é essencial que a notação seja orientada ao cliente.
- No estágio final do processo (o qual é a construção de uma especificação precisa, consistente e completa) a notação é orientada ao analista/engenheiro de SW.

## Especificação Formal

- Enquanto a especificação é desenvolvida em detalhe, a compreensão do especificador aumenta.
- Criar uma especificação formal detalhada força uma análise do sistema detalhada que revela erros e inconsistências na especificação informal. Estas servem como feed-back para modificar especificações anteriores.
- A detecção de erro é o argumento principal da especificação formal.

## Especificação Formal no Processo de Software



## Prós e Contras da Especificação Formal

- **Contras:**
  - Os reponsáveis pela gestão de SW é conservadora e não adotam técnicas onde o custo efetivo não é obvio.
  - Muitos engenheiros de software não foram treinados para especificar formalmente.
  - Clientes não estão familiarizados com técnicas de especificação formal.
  - Algumas classes de sistema (processamento paralelo) são difíceis de especificar formalmente com as técnicas existentes.
  - Há uma vasta ignorancia da praticabilidade e aplicabilidade destas técnicas.
  - Maior foco no desenvolvimento de linguagens formais do que de métodos e ferramentas.

## Prós e Contras da Especificação Formal

- **Prós**
  - Desenvolvimento de uma especificação formal favorece uma compreensão mais profunda dos requisitos, e reduz erros e omissões.
  - Especificações formais são entidades matemáticas e podem ser analisadas utilizando-se métodos matemáticos.
  - É possível provar que uma implementação está de acordo com a especificação.
  - Especificações formais podem ser automaticamente processadas.
  - Podem ser utilizadas como guia para a fase de testes.

## Os 7 Mitos dos Métodos Formais

- Software perfeito com o uso de métodos formais
  - Não faz sentido, uma especificação formal é um modelo do mundo real e pode incorporar erros de compreensão.
- Métodos formais significam prova de programas
  - É apenas uma abordagem de MF.
- MF são caros e seu uso é só justificado em sistemas críticos.
  - Os custos são reduzidos em todas as classes de sistemas.
- MF requerem um alto nível de conhecimento matemático.
  - A matemática é relativamente simples.

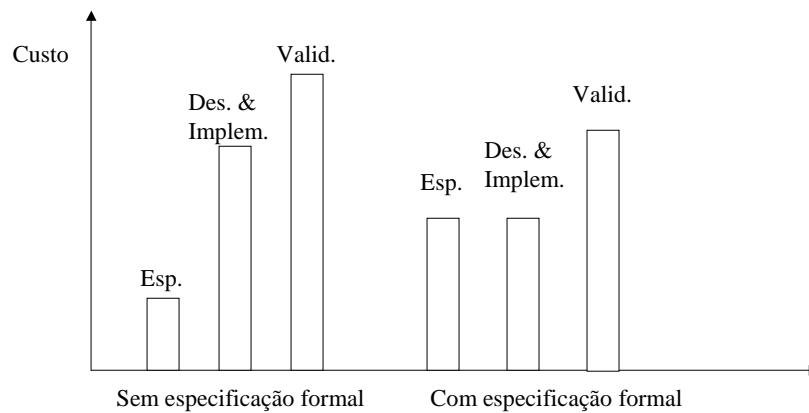
## Os 7 Mitos dos Métodos Formais

- MF aumentam os custos do desenvolvimento.
  - Não é o caso.
- Clientes não entendem MF
  - Com linguagem natural e protótipo elas podem ser compreendidas.
- MF só foram usados em sistemas triviais.
  - Ex. Sistemas de tempo real, osciloscópios, etc.

## Veredicto

- **A favor**
  - Útil em sistemas críticos - onde segurança e fiabilidade são indispensáveis.
- **Contra**
  - Sistemas interativos - utilização de protótipos.
  - Uso de outros métodos de engenharia de software no desenvolvimento de SW aumentou a qualidade do SW

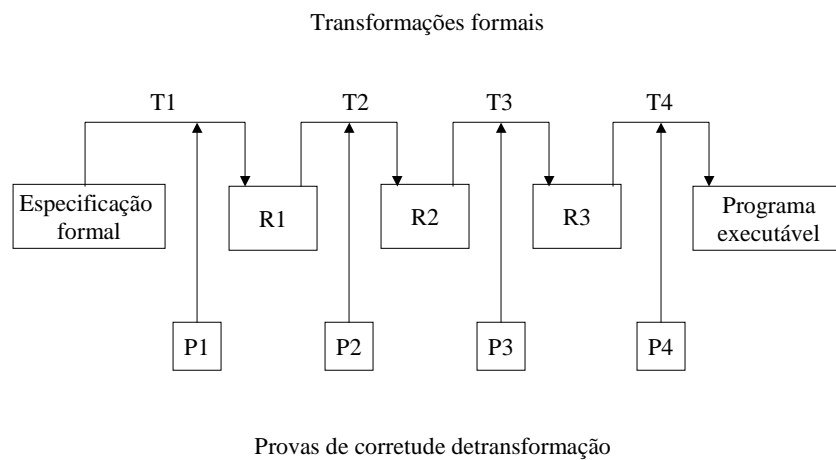
## Custos de Desenvolvimento de SW com Especificação Formal



## Desenvolvimento Transformacional

- A especificação é transformada através de uma série de passos que garantem a correção até um programa final.
- Cada transformação é suficientemente próxima da descrição anterior tal que o esforço de verificar a transformação não é excessivo.
- Pode-se garantir então que o programa é uma implementação verdadeira da especificação.
- Provas de programas são muito longas e inviáveis em sistemas de larga escala.
- Útil no desenvolvimento de sistemas críticos.

## Desenvolvimento Transformacional



## Abordagem Algébrica

- Exemplo: List(Elem)
  - Declarações
    - Create  $\rightarrow$  List
    - Cons(List, Elem)  $\rightarrow$  List
    - Tail(List)  $\rightarrow$  List
    - Head(List)  $\rightarrow$  Elem
    - Length(List)  $\rightarrow$  Integer
  - Axiomas
    - Head (Cons(L, v)) = if L = Create then v else Head(L)
    - Length(Create) = 0
    - Length(Cons(L,v)) = Length(L) + 1
    - Tail(Create) = Create
    - Tail(Cons(L, v)) = if L = Create then Create else Cons(Tail(L), v)

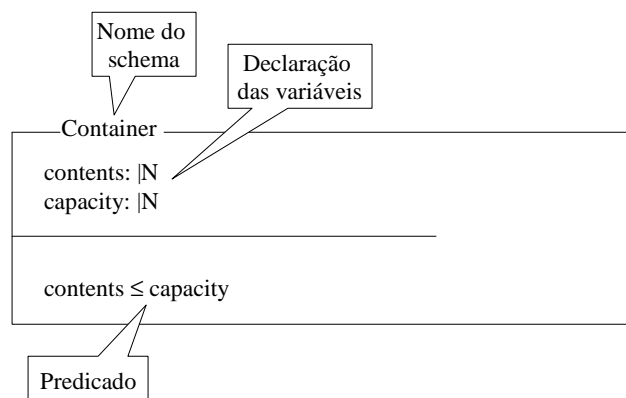
## Especificação Baseada em Modelo

**Z schemas**  
**O processo de especificação Z**  
**Especificação de um dicionário de dados**

## Especificação Baseada em Modelo

- A construção do modelo do sistema utiliza entidades matemáticas tais como conjuntos, que tem uma semântica formal
- Z baseia-se na teoria dos conjuntos com tipos
- Uma especificação em Z consiste:
  - de um modelo matemático do estado do sistema
  - da definição de operações sobre o estado
- Uma especificação em Z consiste de um número de schemas os quais possuem uma parte declarativa e uma parte de predicados
- Schemas podem ser combinados e usados em outros schemas

## Z schemas



## Z schemas

Indicator

light: {on, off}  
reading: |N  
dangerLevel: |N

light = on  $\leftrightarrow$  reading  $\leq$  dangerLevel

StorageTank

Container  
Indicator

reading = contents  
capacity = 5000  
dangerLevel = 50

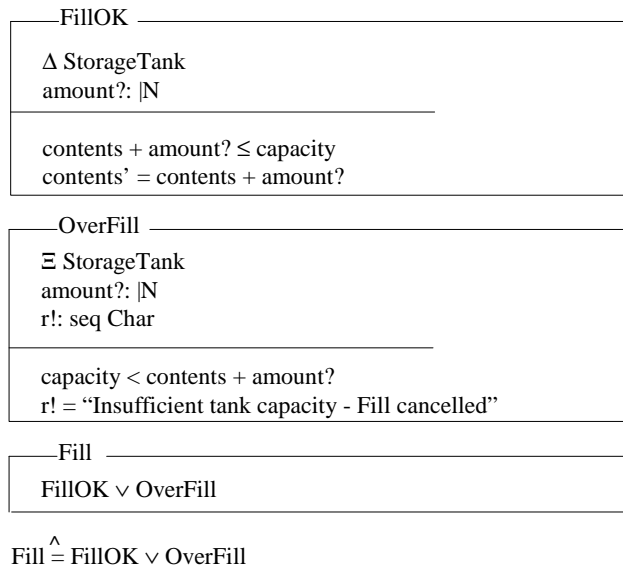
## Z schemas

StorageTank

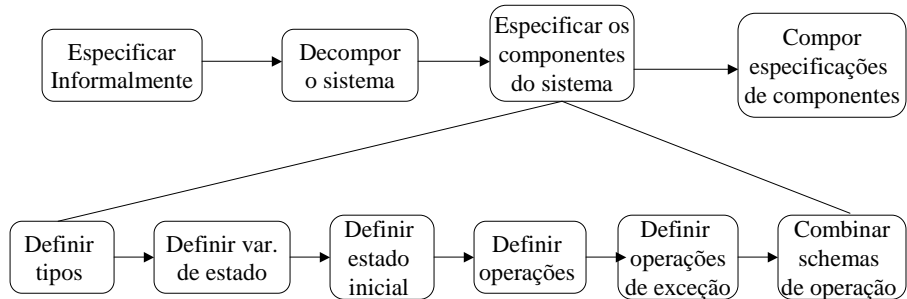
contents: |N  
capacity: |N  
light: {on, off}  
reading: |N  
dangerLevel: |N

contents  $\leq$  capacity  
light = on  $\leftrightarrow$  reading  $\leq$  dangerLevel  
reading = contents  
capacity = 5000  
dangerLevel = 50

## Z schemas (operações)



## O Processo de Especificação Z



## Especificação de um Dicionário de Dados

- Campos:
  - Item name, Description, Type, Date
- Operações:
  - Add, Delete, Lookup, Replace
- Given sets
  - [NAME, DATE]
- Tipos enumerados
  - SemModelTypes = {relation, entity, attribute}

## Especificação de um Dicionário de Dados

DDEntry

entry: NAME  
description: seq char  
type: SemModelType  
creationDate: DATE

#description  $\leq$  2000

DD

DDEntry  
ddict:NAME  $\rightarrow$  {DDEntry}

InitDD

DD'

ddict' =  $\emptyset$

## Schemas para Operações (1)

AddOK

$\Delta$  DD

name?: NAME

entry?: DDEntry

name?  $\notin$  dom ddict

ddict' = ddict  $\cup$  {name?  $\mapsto$  entry?}

LookupOK

$\exists$  DD

name?: NAME

entry!: DDEntry

name?  $\in$  dom ddict

entry! = ddict (name?)

## Schemas para Operações (2)

AddError

$\exists$  DD

name?: NAME

error!: seq char

name?  $\in$  dom ddict

error! = "Name already in the dictionary"

LookupError

$\exists$  DD

name?: NAME

error!: seq char

name?  $\notin$  dom ddict

error! = "Name not in the dictionary"

Add  $\hat{=}$  AddOK  $\vee$  AddError

Lookup  $\hat{=}$  LookupOK  $\vee$  LookupError

## Schemas para Operações (3)

ReplaceOK

$\Delta$  DD

name?: NAME

entry?: DDEntry

name?  $\in$  dom ddict

ddict'  $\oplus$  {name?  $\mapsto$  entry?}

DeleteOK

$\Delta$  DD

name?: NAME

name?  $\in$  dom ddict

ddict' = {name?} y ddict

Replace  $\hat{=}$  ReplaceOK  $\vee$  LookupError

Delete  $\hat{=}$  DeleteOK  $\vee$  LookupError

## Schema para o Sistema

TheDataDictionary

DD

InitDD

Add

Lookup

Delete

Replace