SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

17 Module Guide (H&S Chapt. 6)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 13, 2018

17 Module Guide (H&S Chapt. 6)

- Administrative details
- Index set example
- Parnas's Rational Design Process
- Module guide
- Examples
  - Solar water heating system
  - A mesh generator
  - A maze tracing robot

Administrative Details

- Assignment 2
  - Part 1: February 20, 2018
  - Partner Files: February 26, 2018
  - Part 2: March 2, 2018
- Midterm exam
  - Wednesday, February 28, 7:00 pm
  - 90 minute duration
  - This week's tutorial will cover a sample midterm

And Series

Write a function `forall(A)` that takes a list of boolean values `A` and returns $\forall(i : \mathbb{N} | i \in [0..|A| - 1] : A_i)$

```
def forall(A):
    return reduce(?, A, ?)

print(forall([True, True, True, True]))
print(forall([True, True, True, False]))
```

And Series

Write a function `forall(A)` that takes a list of boolean values `A` and returns $\forall(i : \mathbb{N} | i \in [0..|A| - 1] : A_i)$

```
def forall(A):
    return reduce(?, A, ?)

print(forall([True, True, True, True]))
print(forall([True, True, True, False]))

reduce(lambda x, y:  x and y, A, True)
```

Contains

Write a function `contains(x, xs)` that takes a value `x` and a list of values `xs` and returns $\exists(y|y \in xs : y = x)$

```
def contains(x, xs):
    return reduce(func?, seq?, False)

adjectives = ["lazy", "grouchy", "scheming"]

print(contains(4, [3, 4, 5, 6, 8]))
print(contains("happy", adjectives))
```

Contains

Write a function `contains(x, xs)` that takes a value `x` and a list of values `xs` and returns $\exists(y|y \in xs : y = x)$

```
def contains(x, xs):
    return reduce(func?, seq?, False)

adjectives = ["lazy", "grouchy", "scheming"]

print(contains(4, [3, 4, 5, 6, 8]))
print(contains("happy", adjectives))


reduce(lambda a, b:  a or b, [y==x for y in xs],
False)
```

Connection Between FP Topics

- List comprehensions can be implemented by maps and filters
- All list comprehensions can be implemented via for loops
- Not all for loops can be implemented by a list comprehension
- Details here

```
N = range(10)
dbl = lambda n: n*2
odd = lambda n: n%2==1
list(map(dbl, filter(odd, N)))
# versus
[n * 2 for n in N if n % 2 == 1]
```

Homework: Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where `B[i] = x`.

As a first step, how would you say this mathematically?

Homework: Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where `B[i] = x`.

As a first step, how would you say this mathematically?

$$\text{indexSet}(x, B) \equiv \{i : \mathbb{N} | i \in [0..|B| - 1] \wedge B_i = x : i\}$$

Homework: Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where `B[i] = x`.

As a first step, how would you say this mathematically?

$$\text{indexSet}(x, B) \equiv \{i : \mathbb{N} | i \in [0..|B| - 1] \wedge B_i = x : i\}$$

```
def indexSet(x, B):
    return [i for i in ?  if ?]
```

Homework: Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where `B[i] = x`.

As a first step, how would you say this mathematically?

$$\text{indexSet}(x, B) \equiv \{i : \mathbb{N} | i \in [0..|B| - 1] \wedge B_i = x : i\}$$

```
def indexSet(x, B):
     return [i for i in ?  if ?]
     return [i for i in range(len(B)) if B[i]==x]
```

See the range function example on 2017 midterm

"Upside Down" Tree Uses Relation

- Advantage
  - Does have fan in
- Disadvantages
  - Still limited fan in
  - Would like multiple modules to provide services at the bottom level, not just one
  - Confusing since most modules are at the top level
  - Design cannot be viewed from the top down

Parnas's Rational Design Process (RDP)

- SRS
- MG
- Uses Hierarchy (updated after writing MISes)
- For each module
  - MIS
  - MID (we will not emphasize this document)
- Implementation
- Testing
- Very successfully used on projects such as
  - The Darlington Nuclear Reactor shutdown system
  - The A7-E fighter jet

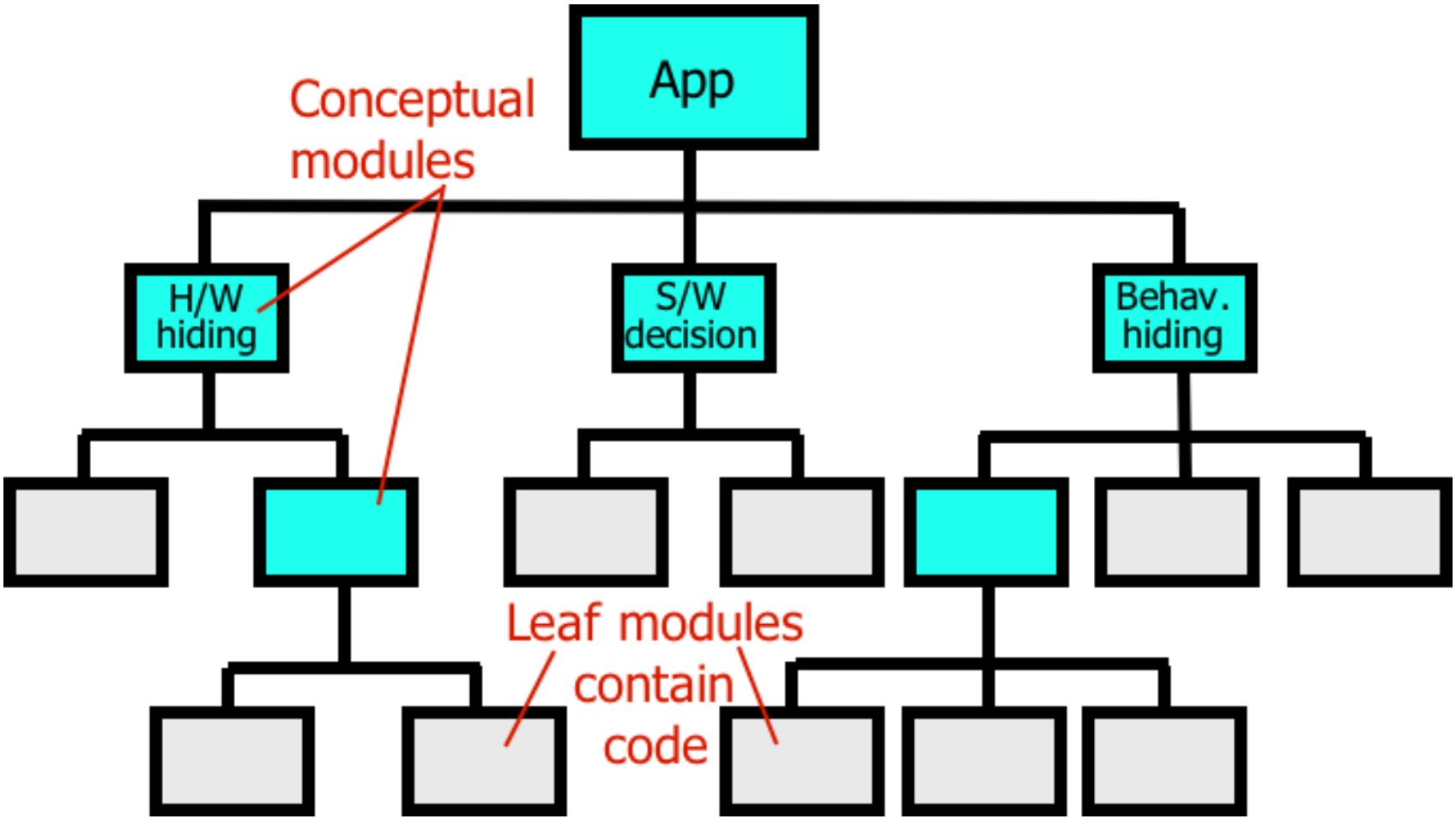
Module Guide

- Part of Parnas' Rational Design Process (RDP)
- When decomposing the system into modules, we need to document the module decomposition so that developers and other readers can understand and verify the decomposition
- Helps future maintainers find appropriate module
- Parnas proposed a Module Guide (MG) based on the decomposition module tree shown earlier
- Decomposition is usually three to five levels deep

Three Top Conceptual Modules in an RDP MG

What are the three groups of modules in a typical information-hiding decomposition?

Module Decomposition (Parnas)



RDP - MG

- The MG consists of a table that documents each module's service and secret
- Conceptual modules will have broader responsibilities and secrets
- Following a particular branch, the secrets at lower levels "sum up" to the secret at higher levels
- The leaf modules that represent code will contain much more precise services and secrets
- Only the leaf modules are actually implemented
- The MG should list the likely and unlikely changes on which the design is based

Module Details

- For each module
- Module name
- Secret (informal description)
- Service or responsibility (informal description)
- For "leaf" modules add
  - Associated requirement
  - Anticipated change
  - Module prefix (optional)

RDP - MG

- Criteria for a good secret
  - One module one secret, especially for leaf modules (watch for “and”)
  - Secrets should often be nouns (data structure, algorithm, hardware, ...)
  - Secrets are often phrased as “How to ... ”

Good Secret?

Is the following a good module secret: "The file format for the map and the rules for validating that the map satisfies the environmental constraints."

Typical Modules

- What are the typical secrets for an input variable?
  - You have an input in the environment, how to get it into your system?
  - What format is the input data?
- What are the secrets for an output variable?
  - How to get an output from inside the system to the external environment?
  - How will the output be determined?
  - What format will the output have?
- What are the secrets for a state variable?
  - What rules are there governing the state transitions?
  - What data structures or algorithms are needed?

Typical Modules

- Input variables
  - Machine-hiding from hardware or OS service
  - Behaviour-hiding input format
- Output variables
  - Machine-hiding
  - Behaviour-hiding output format
  - Behaviour-hiding (calculation)
- State variables
  - Software decision hiding for data structure/algorithm
  - Behaviour-hiding state-drive
- Judgement is critical
- Often combine variables into the same module
- For non-embedded systems, machine hiding for input-output is often combined

RDP - Views

- As well as the MG, the modular decomposition should be displayed using a variety of views
- An obvious one is the Uses Hierarchy
- The Uses Hierarchy is updated once the MIS for all modules is complete
- The Uses Hierarchy can be represented
  - Graphically (if it isn't too large and complex)
  - Using a binary matrix – What would the binary matrix look like?

RDP - MIS

- For each leaf module we need to document its interface and its implementation
- In RDP, the interfaces are documented in the Module Interface Specification (MIS)
- We have already seen MIS examples specified as Module State Machines

RDP - MID

- Another document that is often helpful is the Module Internal Design (MID) for each module
- The MID provides the implementation of the module; that is, it shows how we will deliver on what is promised in the MIS
- The MID is requirements for the code represented at a higher level of abstraction than the code
- The MID uses the syntax of the selected programming language
- The MID shows decisions like whether to use a static array, or dynamic memory allocation and pointers
- We will not focus on MIDs, since for many examples it is feasible to go directly to code

MG Template

- Table of contents
- Introduction
- Anticipated and unlikely changes
- Module hierarchy
- Connection between requirements and design
- Module decomposition
  - Hardware hiding modules
  - Behaviour hiding modules
  - Software decision hiding modules
- Traceability matrices
- Uses hierarchy between modules

Traceability Matrices

- Traceability matrix help inspect the design
- Check for completeness, look at from a different viewpoint

Req.	**Modules**
R1	M1, M2, M3, M7
R2	M2, M3
...	...

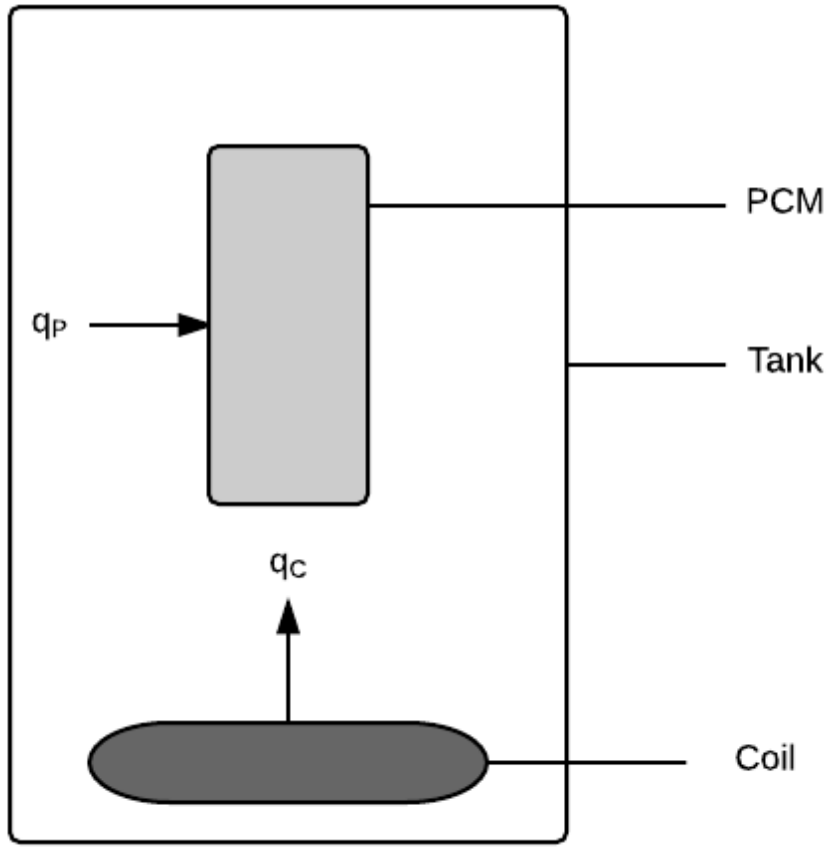
AC	**Modules**
AC1	M1
AC2	M2
...	...

Verification

- Well formed (consistent format/structure)
  - Follows template
  - Follows rules (one secret per module, nouns etc.)
- Feasible (implementable at reasonable cost)
  - Difficult to assess
  - Try sketches of MIS
- Flexible
  - Again try sketches of MIS
  - Thought experiment as if likely change has occurred
  - Low coupling
  - Encapsulate repetitive tasks
- May sometimes have to sacrifice information hiding

Solar Water Heating System Example

- https://github.com/smiths/swhs
- Solve ODEs for temperature of water and PCM
- Solve for energy in water and PCM
- Generate plots



Anticipated Changes?

What are some anticipated changes?

Hint: the software follows the Input $\rightarrow$ Calculate $\rightarrow$ Output design pattern

Anticipated Changes

- The specific hardware on which the software is to run
- The format of the initial input data
- The format of the input parameters
- The constraints on the input parameters
- The format of the output data
- The constraints on the output results
- How the governing ODEs are defined using the input parameters
- How the energy equations are defined using the input parameters
- How the overall control of the calculations is orchestrated
- The implementation of the sequence data structure
- The algorithm used for the ODE solver
- The implementation of plotting data

Module Hierarchy by Secrets

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module Output Format Module Output Verification Module Temperature ODEs Module Energy Equations Module Control Module Specification Parameters Module
Software Decision Module	Sequence Data Structure Module ODE Solver Module Plotting Module

Example Modules from SWHS

Hardware Hiding Modules

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

Example Modules from SWHS

Input Parameters Module

Secrets: The data structure for input parameters, how the values are input and how the values are verified. The load and verify secrets are isolated to their own access programs (like submodules). This, combined with the fact that all of the services are invoked together, suggests that the one module one secret rule can be relaxed here.

Services: ...

Example Modules from SWHS

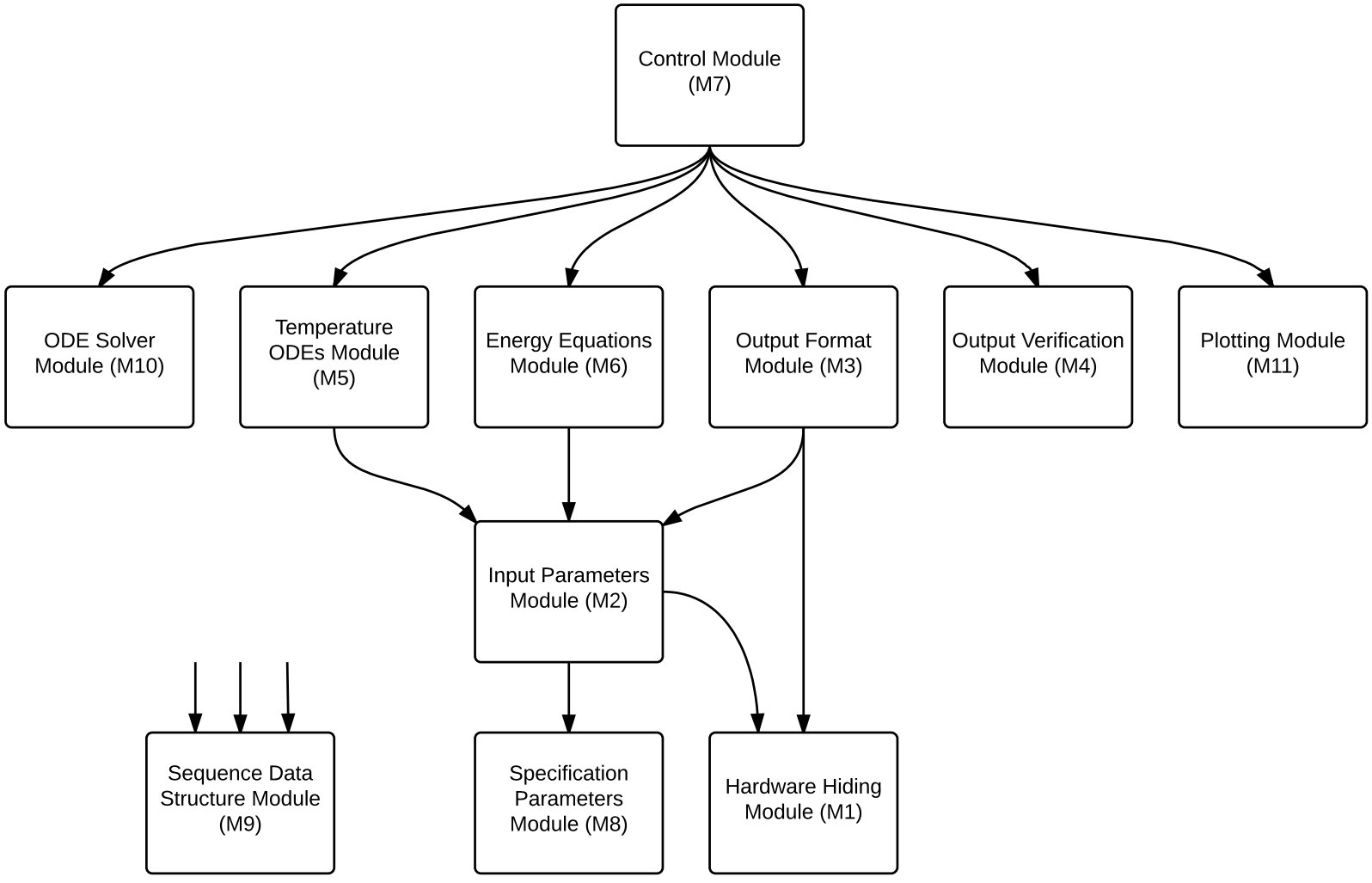
ODE Solver Module

Secrets: The algorithm to solve a system of first order ODEs.

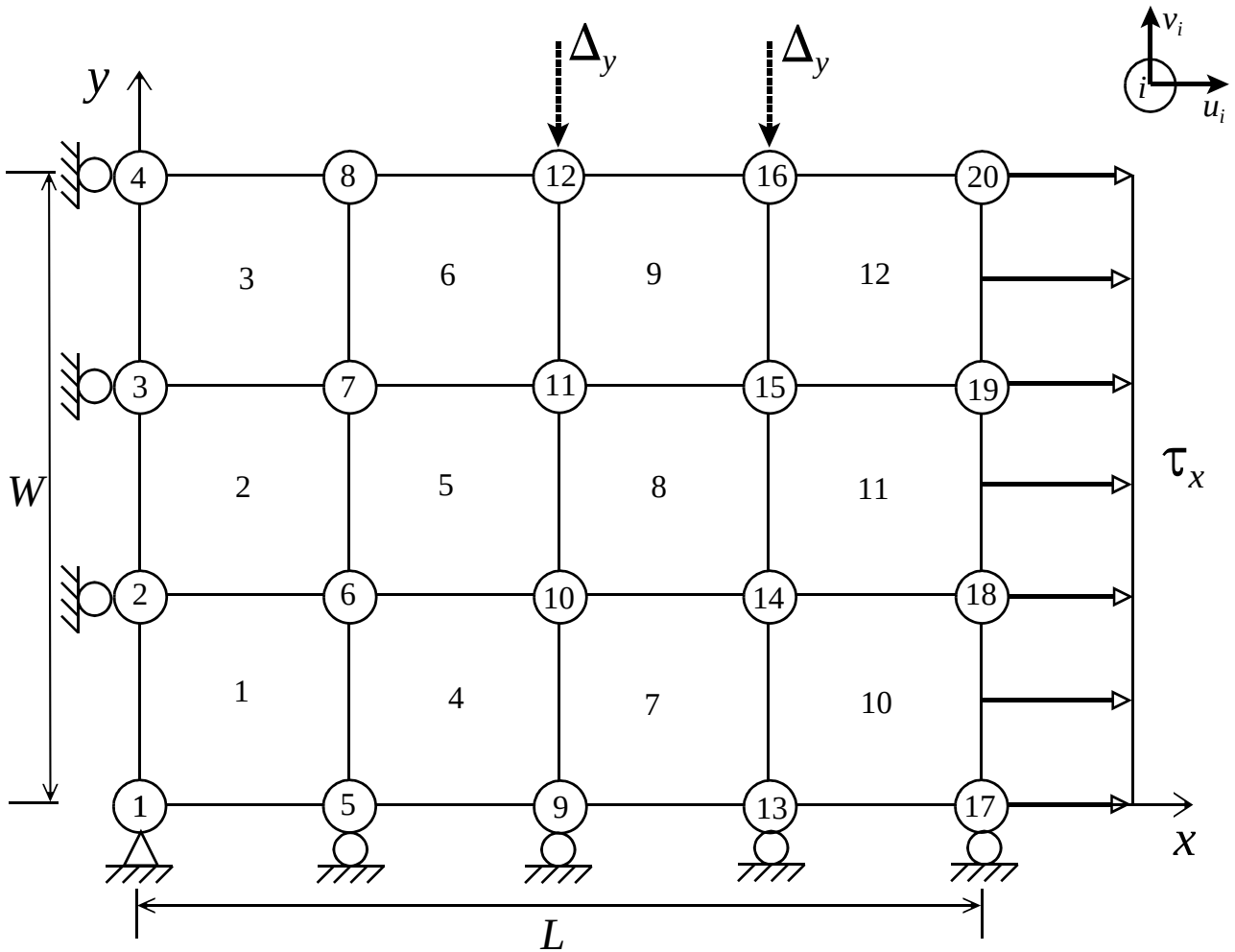
Services: Provides solvers that take the governing equation, initial conditions, and numerical parameters, and solve them.

Implemented By: Matlab

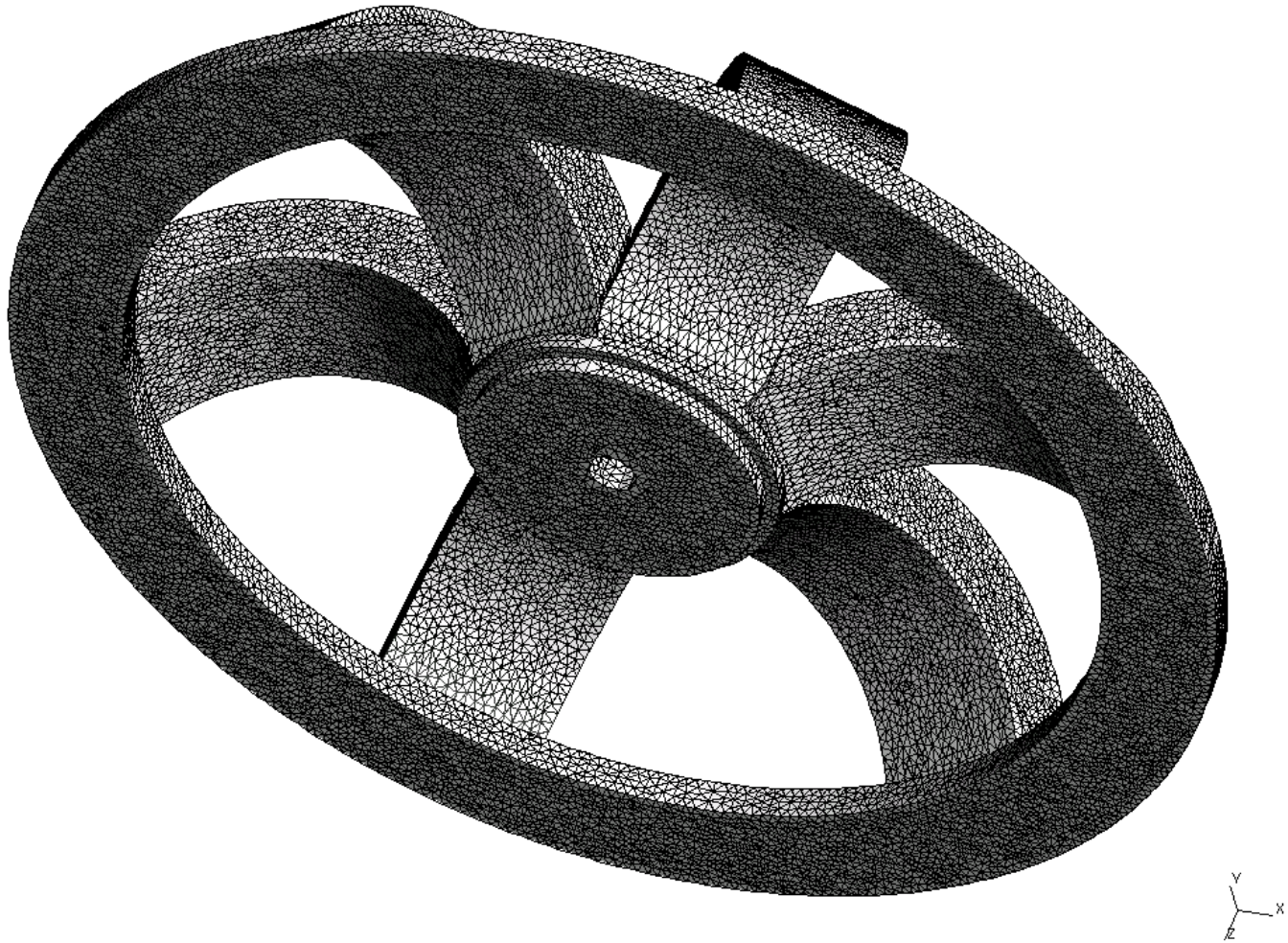
SWHS Uses Hierarchy



Mesh Generator Example



Mesh Generator Complex Circular Geometry



Mesh Generator Example: Design Goals

- Independent and flexible representation for each mesh entity
- Complete separation of geometric data from the topology
- The mesh generator should work with different coordinate systems
- A flexible data structure to store sets of vertices, edges and triangles
- Different mesh generation algorithms with a minimal amount of local changes

Example Mesh Gen Modular Decomposition

Link

Another Mesh Generator Uses Hierarchy