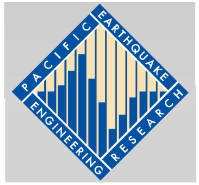
OpenSees Advanced Scripting Techniques & Tips

Frank McKenna
UC Berkeley

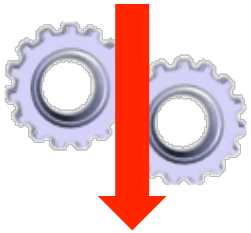


Outline of Workshop

- Programming
- Example: “How Many Fibers?”
- More on Procedures

How Do You Interact With OpenSees?

main.tcl



output

Each OpenSees Script You Write
IS A
PROGRAM

To develop scripts that are easier to generate, easier to modify and debug, easier for others to decipher (think your advisor!) and less error prone

YOU SIMPLY HAVE TO WRITE BETTER PROGRAMS

Are you a Coder or a Programmer?

- A Coder is someone who given a problem will start writing code and is capable of getting it to eventually run.

- A Programmer is someone who actively thinks about abstract solutions to a problem before even opening up a code editor. Once they have the design, they then go about implementing the design.

YOU WANT TO BE A PROGRAMMER

In 40 min I Cannot Show you how to be a Programmer But I can start you down the Path

On Program Design:

- "There are two ways of constructing a software design: one way is to make it so simple that there are *obviously* no deficiencies; the other way is to make it so complicated that there are no *obvious* deficiencies. The first method is far more difficult." **C.A. Hoare, The Emperor's Old Clothes, 1980**

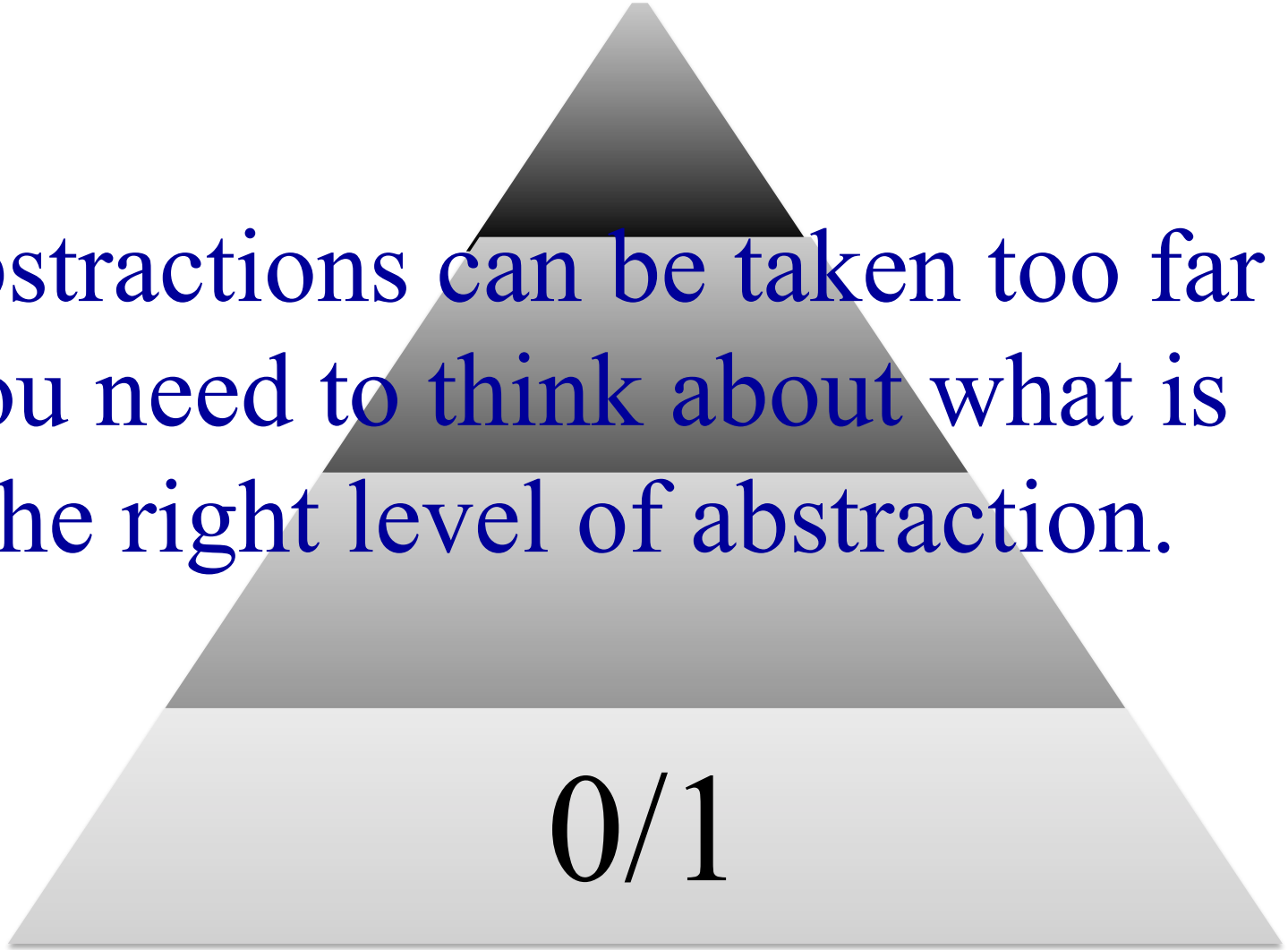
KISS is an acronym for "**Keep it simple, stupid**"

A computer is a pretty stupid machine that stores a bunch of 0/1's and can just move 0/1's around very very very quickly.It only deals with 0/1's, it knows nothing about characters, files, and OpenSees elements!

Abstraction is what makes computers usable

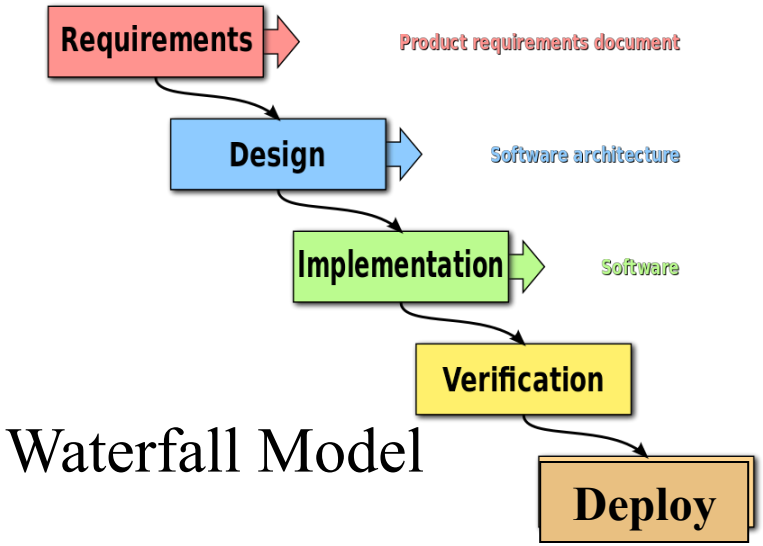
- Computing is all about constructing, manipulating, and reasoning about abstractions. An important prerequisite for writing (good) computer programs is the ability to develop your own abstractions and handle them in a precise manner.
- An abstraction captures only those details about an object that are relevant to the current perspective.
- Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

Abstractions can be taken too far
You need to think about what is the right level of abstraction.

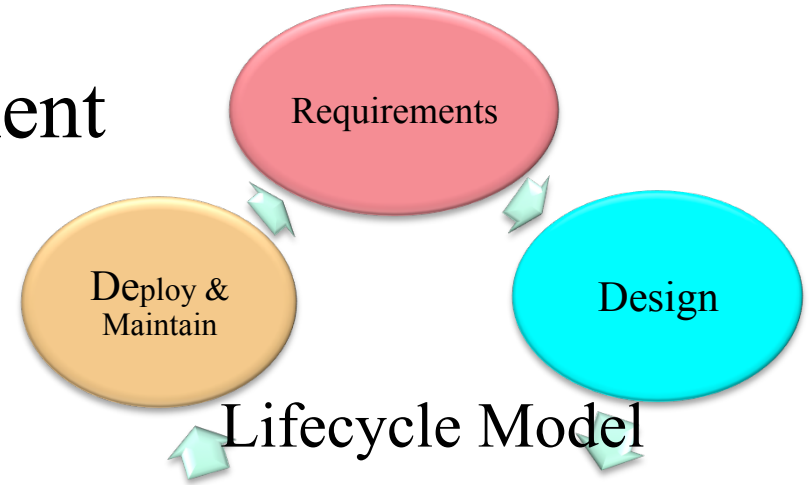


Program Development

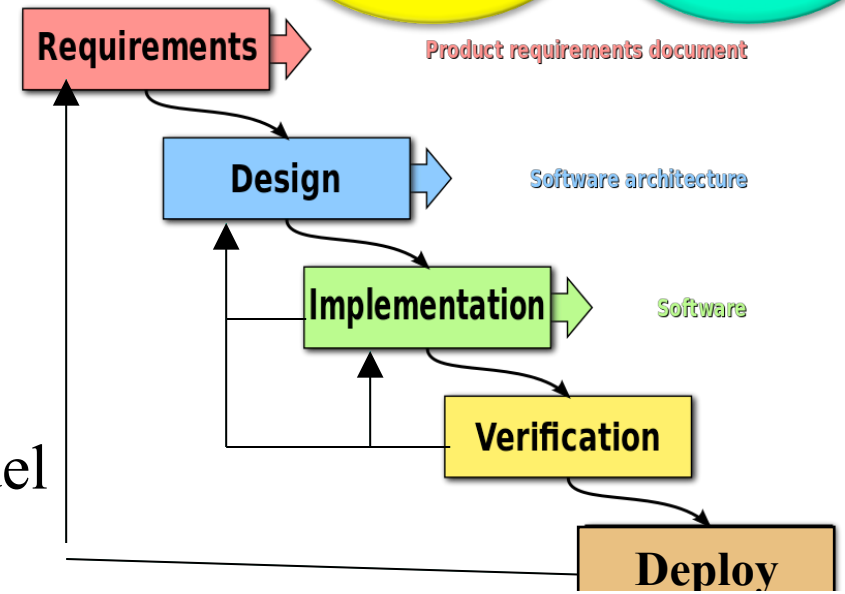
- Plan
- Implement, Test & Document
- Deploy & Maintain



Waterfall Model



Lifecycle Model



Agile Model

Writing Clear Code:

The overarching goal when writing code is to make it easy to read and to understand. **Well-written programs are easier to debug, easier to maintain, and have fewer errors.** You will appreciate the importance of good style when it is your task to understand and maintain someone else's code and even debug your own later.

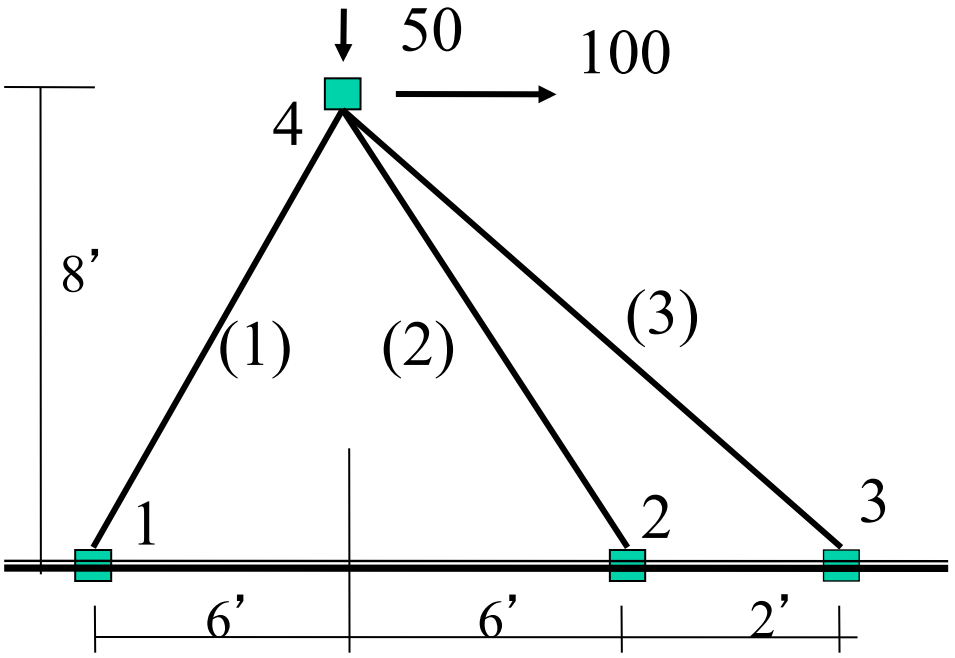
- **Coding:** Keep programs and methods short and manageable. Use straightforward logic and flow-of-control. Avoid magic numbers. Use the language. Use Functions and Variables
- **Naming Conventions** Use meaningful names that convey the purpose of the variable or procedures, use names that you can pronounce, be consistant (lowercase, upper case, …), use shorter names
- **Comments: USE THEM** (some suggest writing comments before the code), the code explains to the computer and programmer *what* is being done; the comments explain to the programmer and others *why* it is being done.

Into to Programming in Java, Robert Sedgewick and Kevin Wayne

- **Check Return Values** and **Provide Useful Error Messages**.

Truss example:

```
model Basic -ndm 2 -ndf 2
node 1 0.0 0.0
node 2 144.0 0.0
node 3 168.0 0.0
node 4 72.0 96.0
fix 1 1 1
fix 2 1 1
fix 3 1 1
uniaxialMaterial Elastic 1 3000.0
element truss 1 1 4 10.0 1
element truss 2 2 4   5.0 1
element truss 3 3 4   5.0 1
timeSeries Linear 1
pattern Plain 1 1 {
   load 4 100.0 -50.0
}
```



	E	A
1	3000	10
2	3000	5
3	3000	5

Truss example using variables:

```
model Basic -ndm 2 -ndf 2

set xCrd1 0.0
set xCrd2 144.0
set xCrd3 168.0
set xCrd4  62.0
set yCrd1 0
set yCrd2 96
set matTag1 1
set timeSeriesTag1
set patternTag1
set E1 3000.
set nodeLoadTag 4
set nodeLoadx 100.
set nodeLoadY  -50;
set A1 10
set A2 5.0
```

```
node 1 $xCrd1 $yCrd1
node 2 $xCrd2  $yCrd1
node 3 $xCrd3  $yCrd1
node 4 $xCrd4  $yCrd2

fix 1 1 1
fix 2 1 1
fix 3 1 1

uniaxialMaterial Elastic $matTag1 $E1

element truss 1 1 4  $A1 $matTag1
element truss 2 2 4  $A2 $matTag1
element truss 3 3 4  $A3 $matTag1

 timeSeries Linear $tsTag1

 pattern Plain $patternTag1 $tsTag1  {
    load 4 $nodeLoadX $nodeLoadY
 }
```

DON'T DO THIS or STOP DOING THIS

When to Use Variables

- **NOT to document the commands**

  USE comments instead

- When you have a variable that you might change at some point later in time or in the script, or for use in a mathematical expression.

USE COMMENTS INSTEAD OF VARIABLES

```
model Basic -ndm 2 -ndf 2

#node $tag $xCrd $yCrd
node 1     0.0    0.0
node 2 144.0    0.0
node 3 168.0    0.0
node 4   72.0  96.0

#fix $nodeTag $xFix $yFix
fix 1 1 1
fix 2 1 1
fix 3 1 1

#uniaxialMaterial Elastic $tag $E
uniaxialMaterial Elastic 1 3000.0

#element truss $tag $iNode $jNode $A $matTag
element truss 1 1 4 10.0 1
element truss 2 2 4   5.0 1
element truss 3 3 4   5.0 1

 timeSeries Linear 1

#pattern Plain $tag $tsTag
pattern Plain 1 1 {
   #load $nodeTag $xForce $yForce
   load 4 100.0 -50.0
}
```

Running in batch mode there are useful default variables: **argv** & **argc**

Terminal — emacs-i386 — 101×39

```
#parse input
if {$argc != 1} {
  puts "Incorrect Usage: OpenSees example2.tcl $E"
  exit
} else {
    set E [lindex $argv 0]
}

# model
model Basic -ndm 2 -ndf 2
node 1 0.0 0.0
node 2 144.0 0.0
node 3 168.0 0.0
node 4 72.0 96.0
fix 1 1 1
fix 2 1 1
fix 3 1 1
uniaxialMaterial Elastic 1 $E
element truss 1 1 4 10.0 1
element truss 2 2 4   5.0 1
timeSeries Linear 1
pattern Plain 1 1 {
   load 4 100.0 -50.0
}

#analysis
integrator LoadControl 1.0
algorithm Linear
numberer Plain
constraints Plain
system BandGeneral
analysis Static
analyze 2

#output
puts "node 4 disp [nodeDisp 4]"
```

Terminal — bash — 101×39

```
fmk:~$ OpenSees example2.tcl 3000.0


        OpenSees -- Open System For Earthquake Engineering Simulation
       Pacific Earthquake Engineering Research Center -- 2.2.1

          (c) Copyright 1999,2000 The Regents of the University of California
                                All Rights Reserved
    (Copyright and Disclaimer @ http://www.berkeley.edu/OpenSees/copyright.html)


node 4 disp              1.8750000000000000000            -0.88541666666666662966
fmk:~$ OpenSees example2.tcl 6000.0


        OpenSees -- Open System For Earthquake Engineering Simulation
       Pacific Earthquake Engineering Research Center -- 2.2.1

          (c) Copyright 1999,2000 The Regents of the University of California
                                All Rights Reserved
    (Copyright and Disclaimer @ http://www.berkeley.edu/OpenSees/copyright.html)


node 4 disp              0.93750000000000000000           -0.44270833333333331483
fmk:~$
```

Example Tcl

•*variables & variable substitution*

```
>set a 1
1
>set b a
a
>set b $a
1
```

•*expression evaluation*

```
>expr 2 + 3
5
>set b [expr 2 + $b]
 3
```

•*file manipulation*

```
>set fileId [open tmp w]
??
>puts $fileId "hello"
>close $fileID
>type tmp
hello
```

•*lists*

```
>set a {1 2 three}
1 2 three
>set la [llength $a]
3
>set start [lindex $a 0]
1
>lappend a four
1 2 three four
```

•*sourcing other files*

```
>source Example1.tcl
```

•*procedures & control structures*

```
> for {set i 1} {$i < 10} {incr i 1} {
            puts "i equals $i"
}
…
> set sum 0
foreach value {1 2 3 4} {
   set sum [expr $sum + $value]
}
>puts $sum
10
>proc guess {value} {
    global sum
    if {$value < $sum} {
       puts "too low"
    } else {
       if {$value > $sum} {
          puts "too high"
       } else { puts "you got it!"}
   }
}
 > guess 9
```

Outline of Workshop

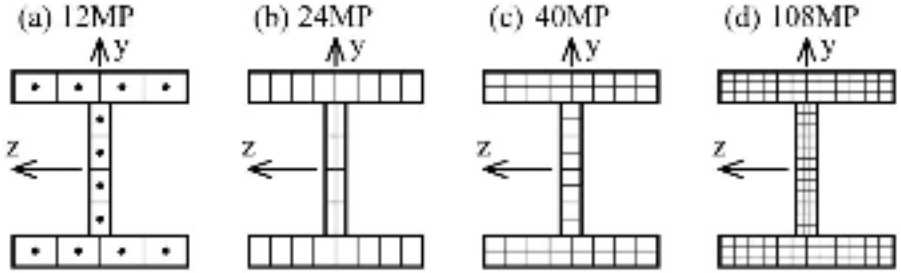
- Programming
- Example: “How Many Fibers?”
- More on Procedures

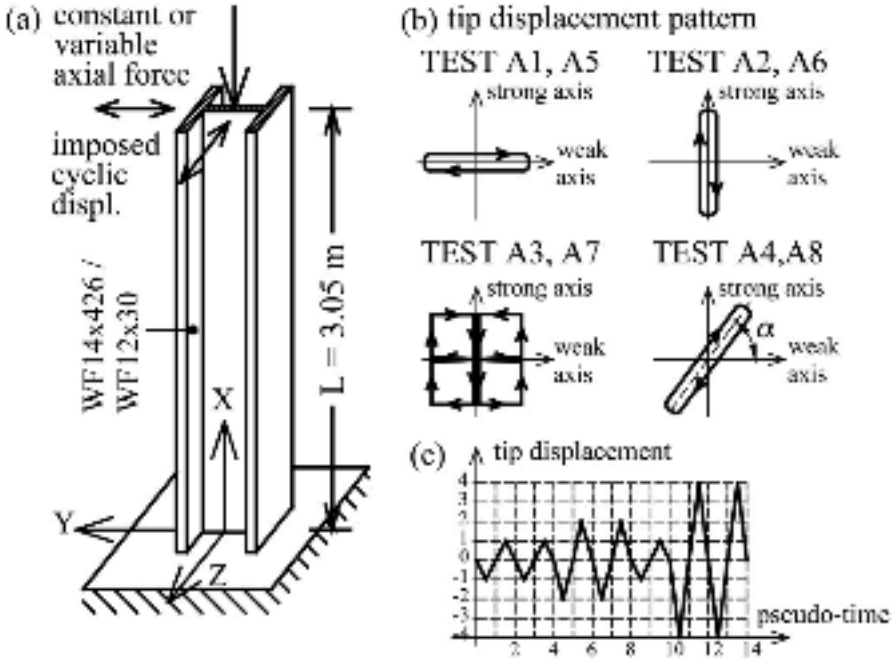
How Many Fibers in a Section?

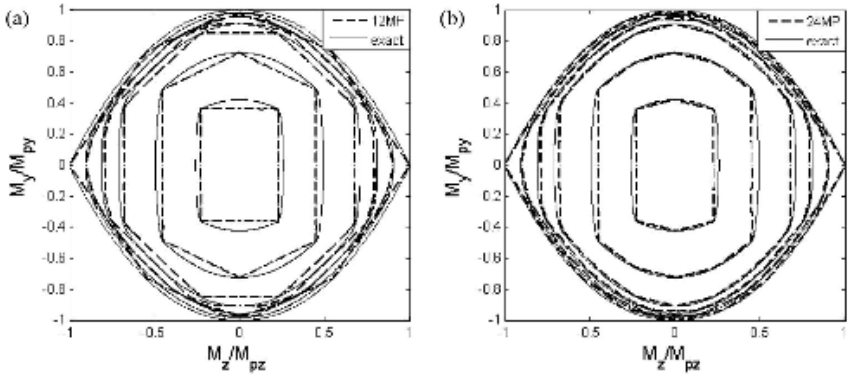
$$\mathbf{s} = \begin{pmatrix} N \\ M_z \\ M_y \end{pmatrix} = \int_A \begin{pmatrix} 1 \\ -y \\ z \end{pmatrix} \sigma \, dA \approx \sum_{i=1}^{Nfib} \begin{pmatrix} 1 \\ -y_i \\ z_i \end{pmatrix} \sigma_i A_i$$

You Don't want too many for computation and memory demands on application
And you don't want too few for accuracy

RECORD CURRENTLY is 11,210 fibers in a section







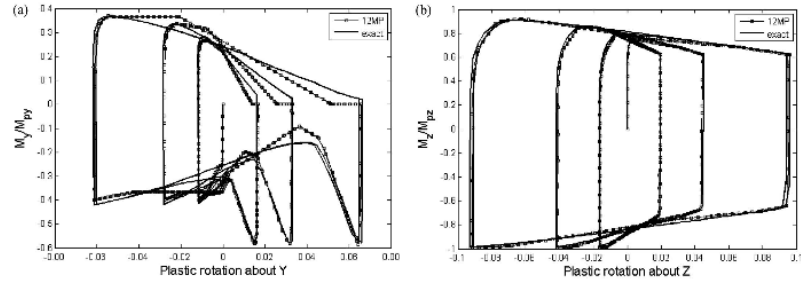
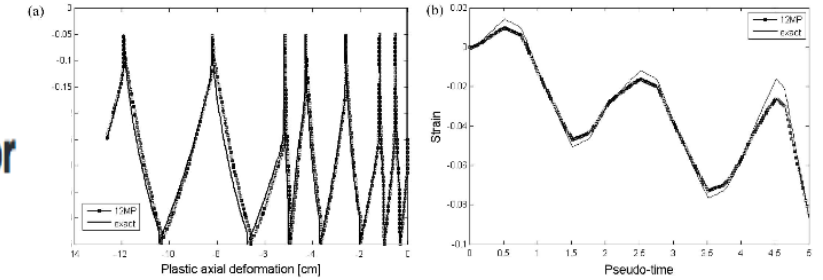


Fig. 6. Test A8_4: Normalized bending moment–plastic rotation relation for 12MP solution and "exact" solution: (a) about y-axis; (b) about z-axis

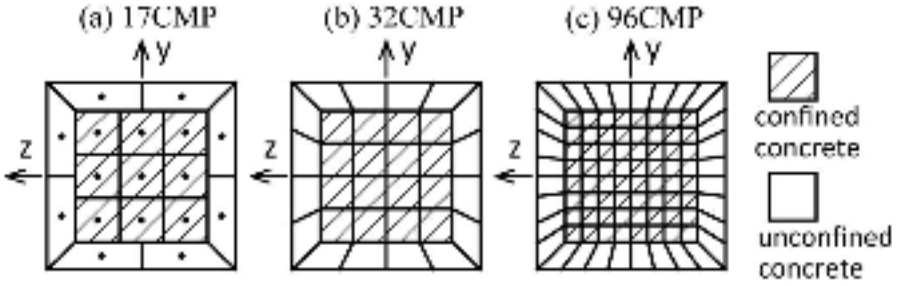


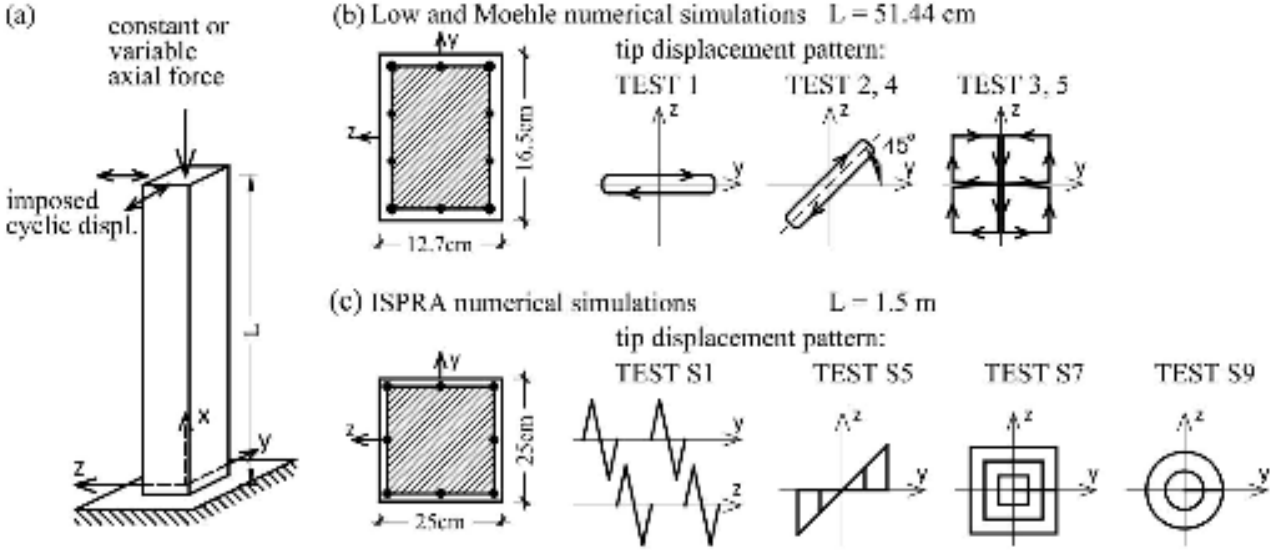
Section Discretization of Fiber Beam-Column Elements for Cyclic Inelastic Response

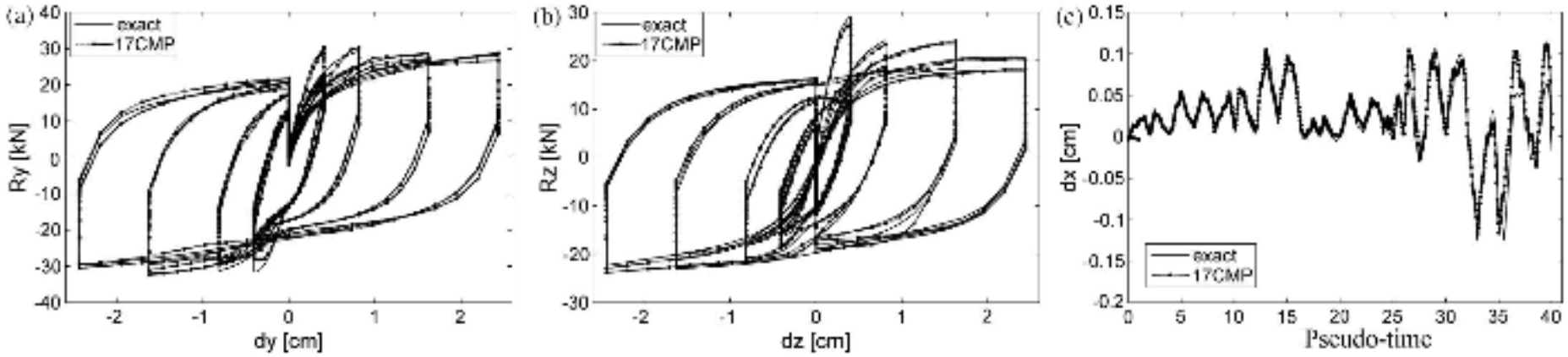
Svetlana M. Kostic[1] and Filip C. Filippou, M.ASCE[2]

Section Discretization of Fiber Beam-Column Elements for Cyclic Inelastic Response

Svetlana M. Kostic[1] and Filip C. Filippou, M.ASCE[2]

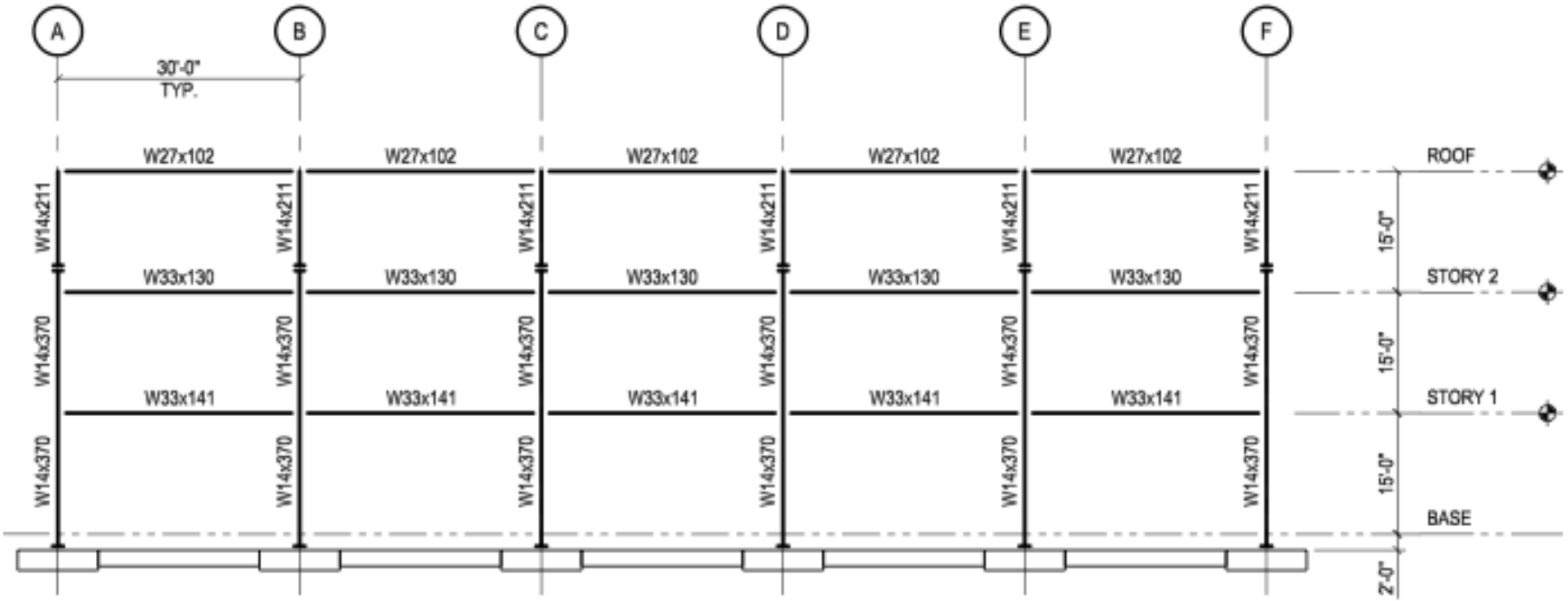






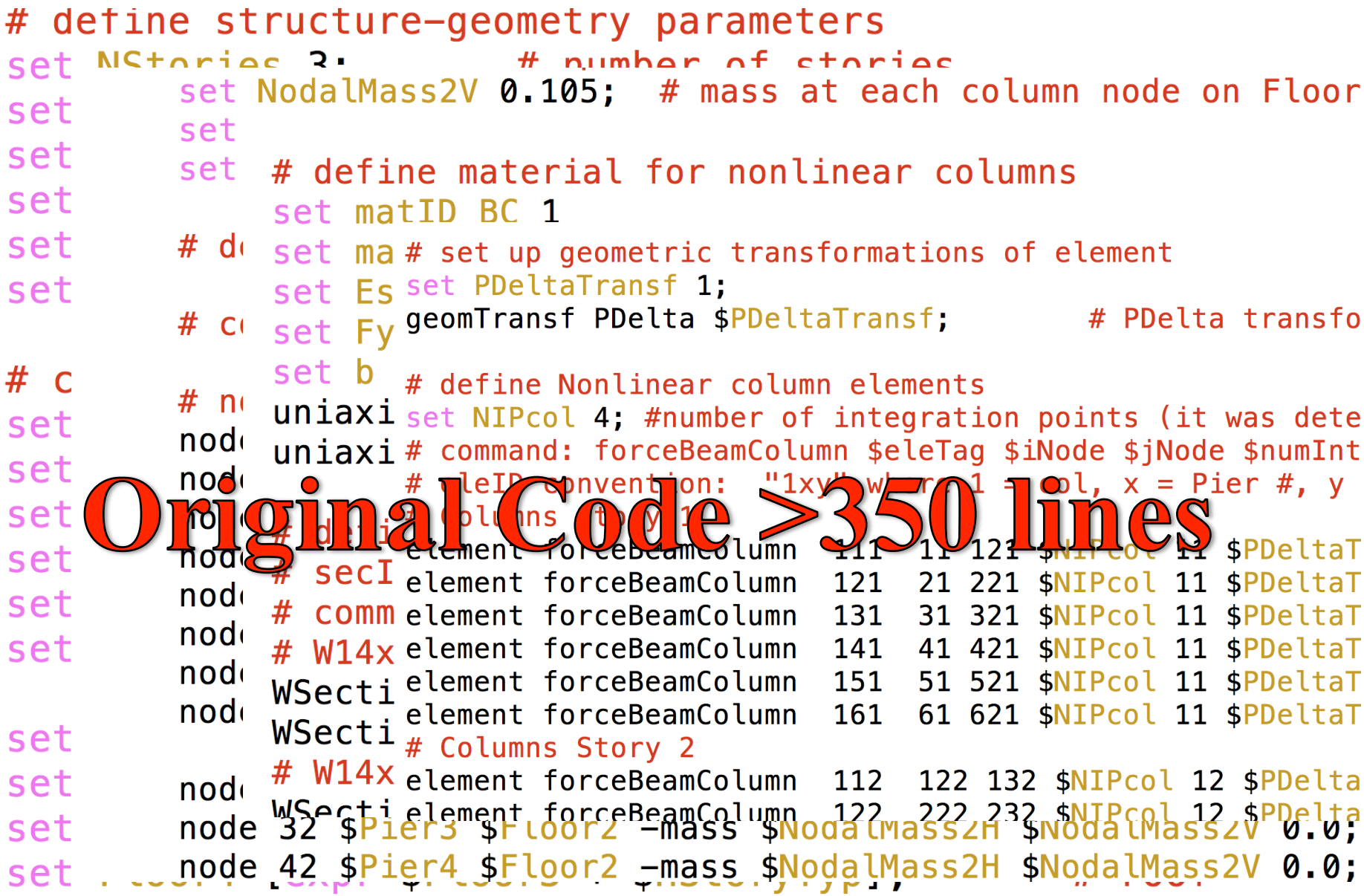
http://www.neng.usu.edu/cee/faculty/kryan/NEESTIPS/Figs%20SMRF.pdf

EXAMPLE: MRF.tcl

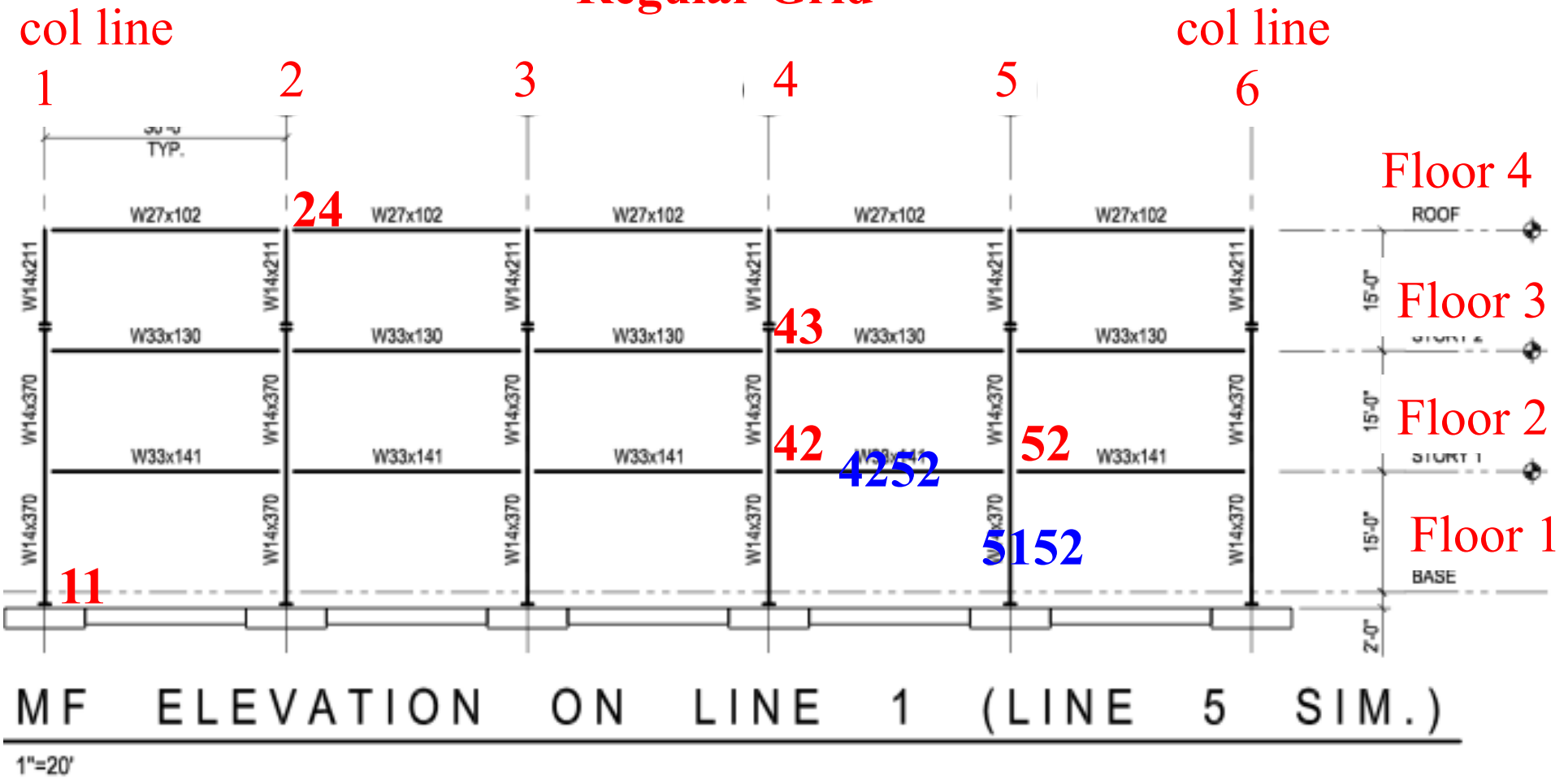


MF ELEVATION ON LINE 1 (LINE 5 SIM.)

1"=20'


Original Code >350 lines

Steel W Sections & Regular Grid



Nodes # colfloor

Elements # $iNode$jNode

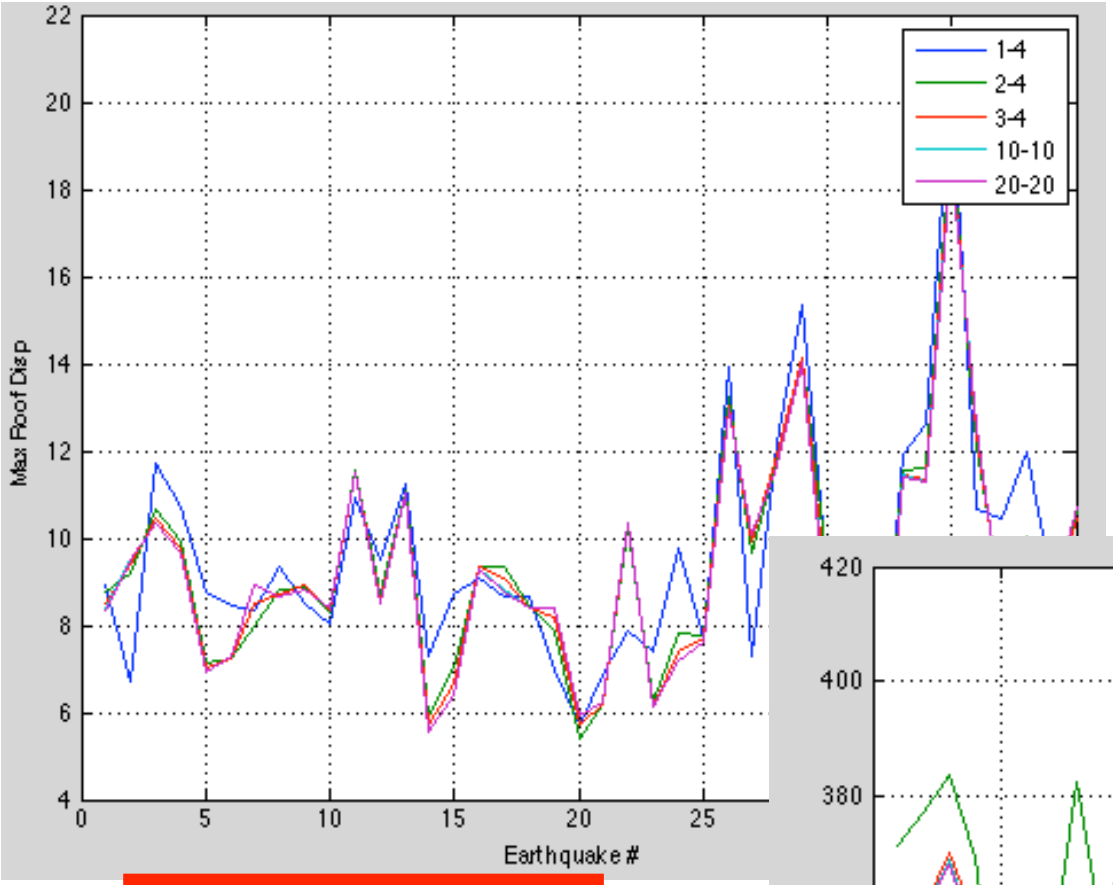
(if more than 10 col lines or floors, start numbering at 10, if > 100, at 100, ….)

```
model Basic –ndm 2 –ndf 3
source SteelWSections.tcl

# add nodes & bc
set floorLocs {0. 204. 384. 564.};                  # floor locations
set colLocs {0. 360. 720. 1080. 1440. 1800.}; #column line locations
set massXs {0. 0.419 0.419 0.430}; # mass at nodes on each floor in x dirn
set massYs {0. 0.105 0.105 0.096} ; #    “     “     “     “       “       “    in y dirn
foreach floor {1 2 3 4} floorLoc $floorLocs massX $massXs massY $massYs {
    foreach colLine {1 2 3 4 5 6} colLoc $colLocs {
       node $colLine$floor $colLoc $floorLoc -mass $massX $massY 0.
        if {$floor == 1} {fix $colLine$floor 1 1 1}
    }
 }

#uniaxialMaterial Steel02 $tag $Fy $E $b $R0 $cr1 $cr2
uniaxialMaterial Steel02 1 50.0 29000. 0.003  20 0.925 0.15
uniaxialMaterial Fatigue 2 1

set colSizes  {W14X370 W14X370 W14X211};     #col sizes stories 1, 2 and 3
set beamSizes {W33X141 W33X130 W27X102};  #beams sizes floor 1, 2, and 3

# add columns
geomTransf PDelta 1
foreach colLine {1 2 3 4 5 6} {
    foreach floor1 {1 2 3} floor2 { 2 3 4} {
       set theSection [lindex $colSizes [expr $floor1 -1]]
       forceBeamWSection2d $colLine$floor1$colLine$floor2 $colLine$floor1 $colLine$floor2 $theSection 2 1
    }
 }
#add beams
geomTransf Linear 2
 foreach colLine1 {1 2 3 4 5} colLine2 {2 3 4 5 6} {
    foreach floor {2 3 4} {
       set theSection [lindex $beamSizes [expr $floor -2]]
       forceBeamWSection2d $colLine1$floor$colLine2$floor $colLine1$floor $colLine2$floor $theSection 2 2
    }
 }
```

SteelWSections.tcl

```
proc elasticBeamWSection2d {eleTag iNode jNode sectType E transfTag {Orient XX}} {
    global WSection
    global in
    set found 0
    foreach {section prop} [array get WSection $sectType] {
        set propList [split $prop]
        set A [expr [lindex $propList 0]*$in*$in]
        set Ixx [expr [lindex $propList 5]*$in*$in*$in*$in]
        set Iyy [expr [lindex $propList 6]*$in*$in*$in*$in]
        if {$Orient == "YY" } {
            puts "element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iyy $transfTag"
            element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iyy $transfTag
        } else {
            puts "element elasticBeamColumn $eleTag $iNode $jNode $A $E $Ixx $transfTag"
            element elasticBeamColumn $eleTag $iNode $jNode $A $E $Ixx $transfTag
        }
    }
}

#Winxlb/f "Area(in2) d(in) bf(in) tw(in) tf(in) Ixx(in4) Iyy(in4)"
array set WSection {
    W44X335     "98.5 44.0 15.9 1.03 1.77 31100 1200 74.7"
    W44X290     "85.4 43.6 15.8 0.865 1.58 27000 1040 50.9"
    W44X262     "76.9 43.3 15.8 0.785 1.42 24100 923 37.3"
    W44X230     "67.7 42.9 15.8 0.710 1.22 20800 796 24.9"
    W40X593     "174 43.0 16.7 1.79 3.23 50400 2520 445"
    W40X503     "148 42.1 16.4 1.54 2.76 41600 2040 277"
```

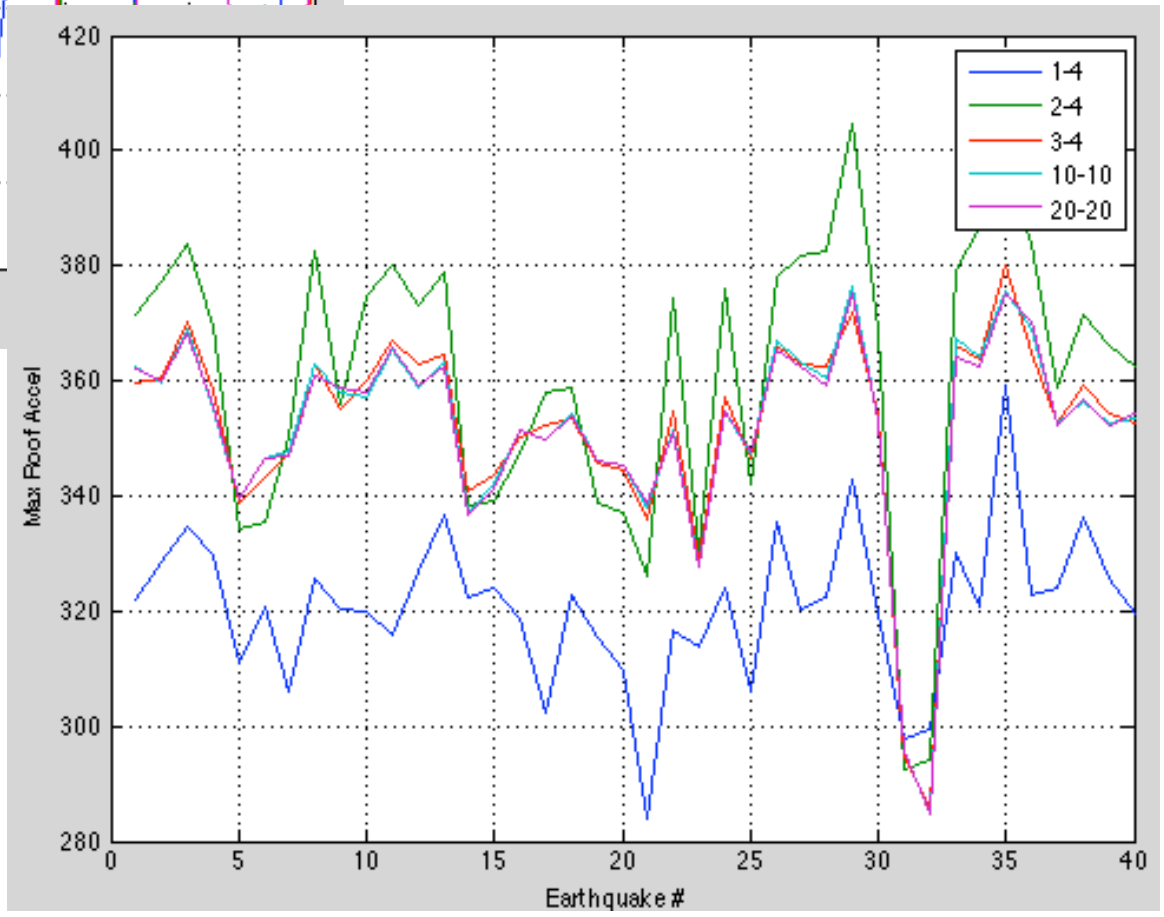
```
 proc forceBeamWSection2d {eleTag iNode jNode sectType matTag transfTag {Orient XX}} {

    global FiberSteelWSection2d
    set nFlange 3
    set nWeb    4
    set nip 4

    FiberSteelWSection2d $eleTag $sectType $matTag $nFlange $nWeb
    element forceBeamColumn $eleTag $iNode $jNode $nip $eleTag $transfTag
}

proc dispBeamWSection2d {eleTag iNode jNode sectType matTag transfTag {Orient XX}} {

    global FiberSteelWSection2d
    set nFlange 3
    set nWeb    4
    set nip 4

    FiberSteelWSection2d $eleTag $sectType $matTag $nFlange $nWeb
    element dispBeamColumn $eleTag $iNode $jNode $nip $eleTag $transfTag
}


proc FiberSteelWSection2d {sectTag sectType matTag nFlange nWeb {Orient XX}} {
    global WSection
    global in
    set found 0
    foreach {section prop} [array get WSection $sectType] {
        set propList [split $prop]

        set  d [expr [lindex $propList 1]*$in]
        set bf [expr [lindex $propList 2]*$in]
        set tw [expr [lindex $propList 3]*$in]
        set tf [expr [lindex $propList 4]*$in]
        Wsection $sectTag $matTag $d $bf $tf $tw $nFlange 1 1 $nWeb $Orient
        set found 1
    }
    if {$found == 0} {
         puts "FiberSteelWSection2d sectType: $sectType not found for sectTag: $sectTag"
    }
}
```

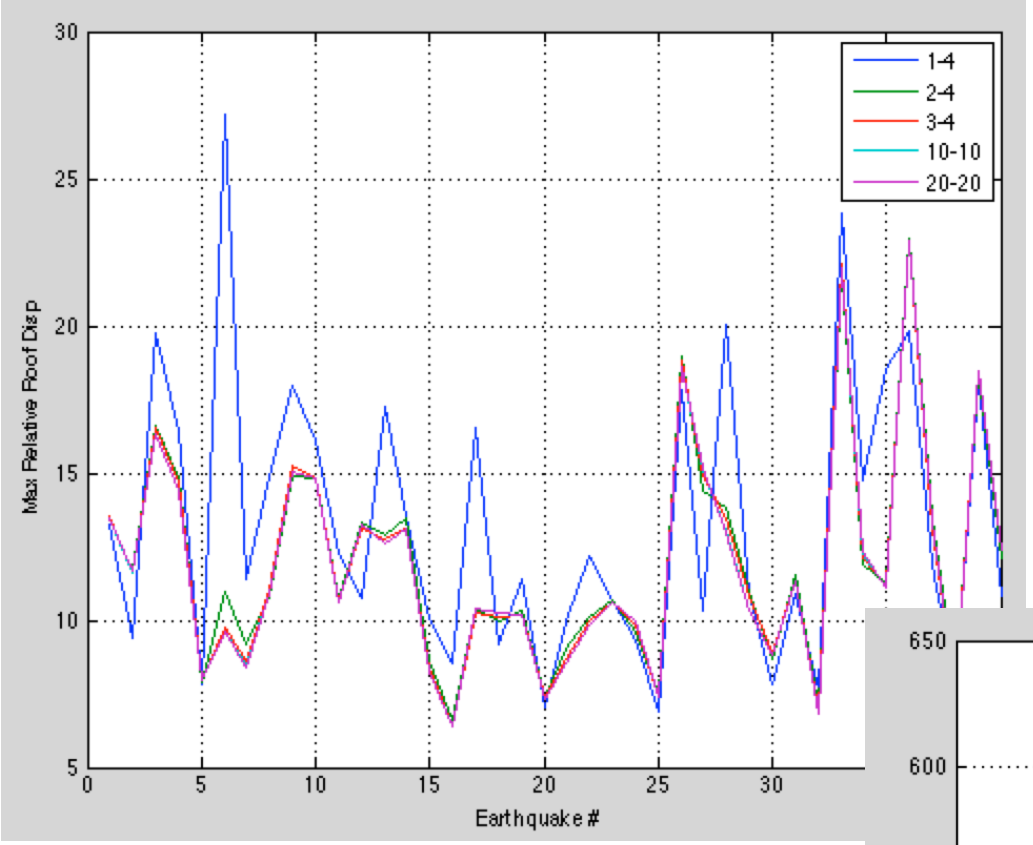


fibers flange- # fibers web

Results 10 % in 50year

	Time
1-4	292sec
2-4	302sec
3-4	309sec
10-10	578sec
20-20	1001sec
35-30	1305sec

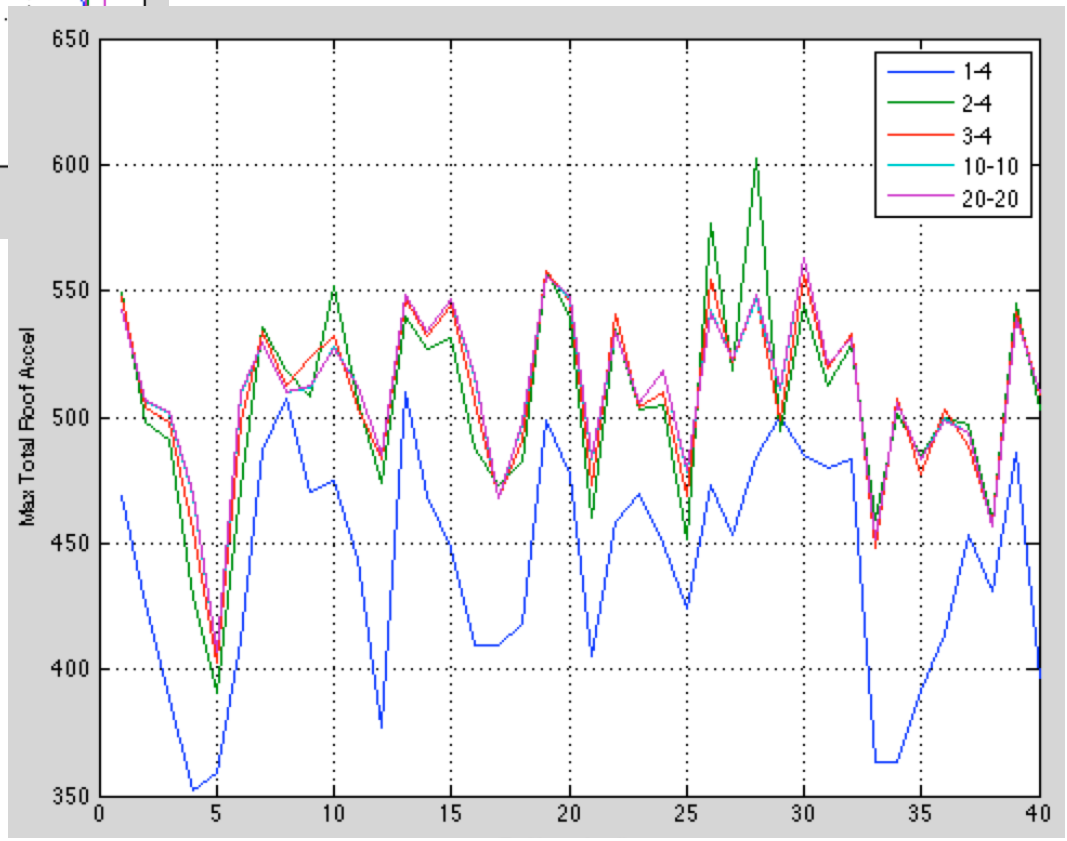




fibers flange- # fibers web

Results 2% in 50year

	Time
1-4	254sec
2-4	265sec
3-4	272sec
10-10	506sec
20-20	879sec
35-30	1539sec



When DONE with a Program

Critique It

(if you have the time this will make you a better programmer)

```
proc forceBeamWSection2d {eleTag iNode jNode sectType matTag transfTag {Orient XX}} {

    global FiberSteelWSection2d
    set nFlange 1
    set nWeb    4
    set nip 4

    FiberSteelWSection2d $eleTag $sectType $matTag $nFlange $nWeb
    element forceBeamColumn $eleTag $iNode $jNode $nip $eleTag $transfTag
}

proc dispBeamWSection2d {eleTag iNode jNode sectType matTag transfTag {Orient XX}} {

    global FiberSteelWSection2d
    set nFlange 1
    set nWeb    4
    set nip 4

    FiberSteelWSection2d $eleTag $sectType $matTag $nFlange $nWeb
    element dispBeamColumn $eleTag $iNode $jNode $nip $eleTag $transfTag
}


proc FiberSteelWSection2d {sectTag sectType matTag nFlange nWeb {Orient XX}} {
    global WSection
    global in
    set found 0
    foreach {section prop} [array get WSection $sectType] {
        set propList [split $prop]

        set  d [expr [lindex $propList 1]*$in]
        set bf [expr [lindex $propList 2]*$in]
        set tw [expr [lindex $propList 3]*$in]
        set tf [expr [lindex $propList 4]*$in]
        Wsection $sectTag $matTag $d $bf $tf $tw $nFlange 1 1 $nWeb $Orient
        set found 1
    }
    if {$found == 0} {
         puts "FiberSteelWSection2d sectType: $sectType not found for sectTag: $sectTag"
    }
}
```

I DON'T LIKE THIS

Outline of Workshop

- Programming
- Example: “How Many Fibers?”
- More on Procedures

Basic Procedure

- A procedure can be defined with a set number of required arguments

```
proc sum {x y} {
   return [expr $x + $y]
}

puts “sum 3 4 [sum 3 4]”
```

```
puts sum 3 4: 7
```

More Advanced Procedures I

- Variable in the procedure can be defined with default values.

```
proc sum {x {y 0}} {
    return [expr $x + $y]
}

puts "sum 3: [sum 3]"
puts "sum 3 4: [sum 3 4]"
```

```
sum 3: 3
sum 3 4: 7
```

More Advanced Procedures II

- By declaring **args** as the last argument a procedure can take a variable number of arguments

```
proc sum {x {y 0} args} {
    set sum [expr $x + $y]
    foreach arg $args {
        set sum [expr $sum + $arg]
    }
    return $sum
}


puts "sum 3: [sum 3]"
puts "sum 3 4: [sum 3 4]”
puts “sum 3 4 5 6: [sum 3 4 5 6]”
```

```
sum 3: 3
sum 3 4: 7
sum 3 4 5 6: 18
```

SteelWSection revisited:

```
proc forceBeamWSection2d {eleTag iNode jNode sectType matTag transfTag args} {

   global FiberSteelWSection2d

   set Orient XX
   if {[lsearch $args “Orient"] != -1} {
      set Orient YY
   }

   set nFlange 3
   if {[lsearch $args "nFlange"] != -1} {
      set loc [lsearch $args "nFlange"]
      set nFlange [lindex $args [expr $loc+1]]
   }

   set nWeb   4
   if {[lsearch $args "nWeb"] != -1} {
      set loc [lsearch $args "nWeb"]
      set nWeb [lindex $args [expr $loc+1]]
   }

   set nip 4
   if {[lsearch $args "nip"] != -1} {
      set loc [lsearch $args "nip"]
      set nip [lindex $args [expr $loc+1]]
   }

   FiberSteelWSection2d $eleTag $sectType $matTag $nFlange $nWeb $Orient
   element forceBeamColumn $eleTag $iNode $jNode $nip $eleTag $transfTag
}
```

forceBeamWSection2d $colLine$floor1$colLine$floor2 $colLine$floor1 $colLine$floor2 $theSection 2 1 nip 5

Procedure Pointers

- Passing procedure pointers around.

```
#define my procedure
proc eat    x {puts "eating $x"}
proc drink  x {puts "drinking $x"}
proc inhale x {puts "inhaling $x"}

#build a mapping table
array set fp {solid eat liquid drink gaseous inhale}

# testing:
foreach {state matter} {solid bread  liquid wine  gaseous perfume} {
    $fp($state) $matter
}
```

```
eating bread
drinking wine
inhaling perfume
```

How You Might Use It

- Passing procedure pointers around.

```
#SteelWSections.tcl
proc elasticElement   {secType …}  { …..}
proc colForceElement {secType ….} { ….}
proc colDispBasedElement {secType …} { …..}
…
#build a mapping table in main script
array set eleType {colEle colForceEle beamEle elasticElement }

 # add column elements:
 $eleType(colEle) {……..}
```

In Conclusion

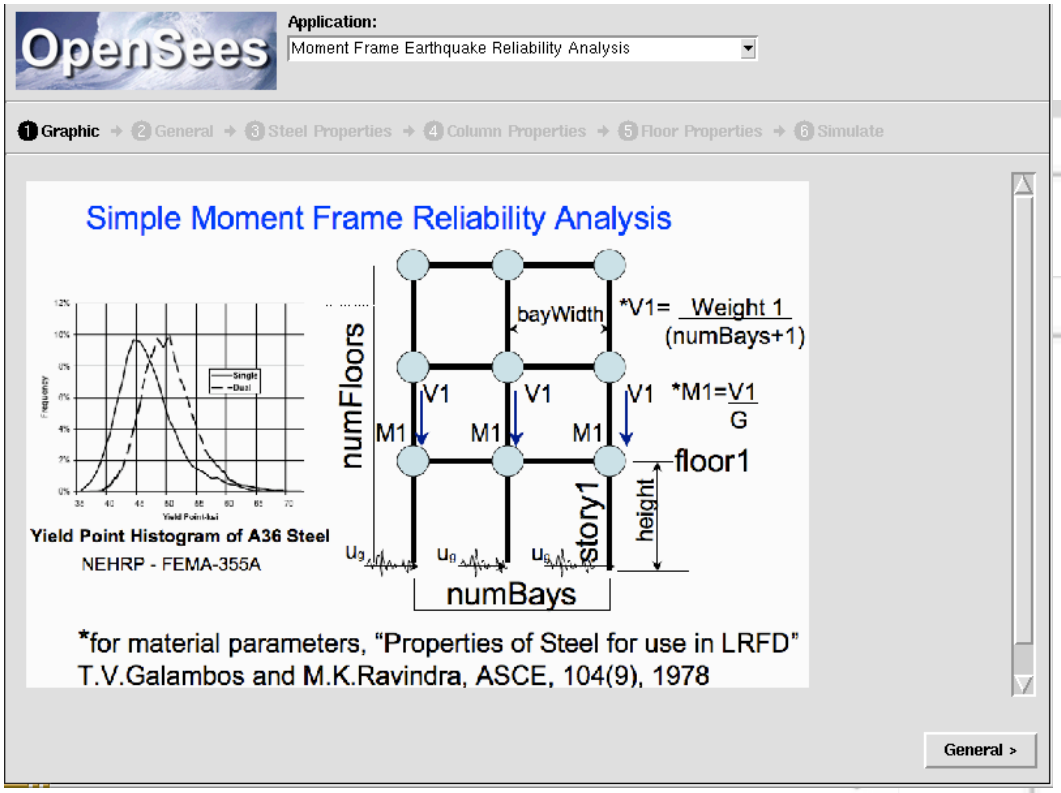
BECOME A PROGRAMMER:

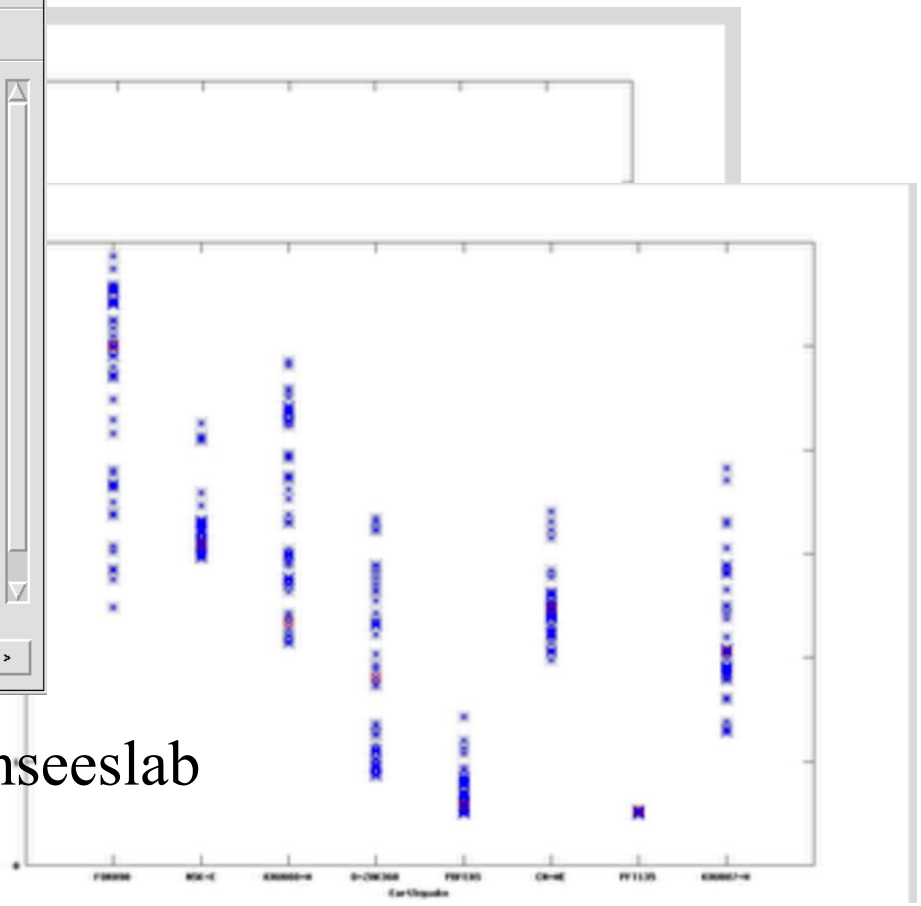
- **DESIGN FIRST, THINK ABSTRACTIONS & KISS**
- **Do Format** and **Be consistent** with formatting (spaces or tabs for indentation, but DO indent)
- **Use Variables/Procedures** and **Be consistent** with naming convention and **DO use meaningful names**. **BUT DON'T OVERDO IT** and **DON'T USE VARIABLES TO COMMENT.**
- **COMMENT** (what are the variables, what the code block will do in plain language). DON'T OVERDO IT! Some people suggest writing the comments before any code!
- **Check return values.**
- **Provide useful error messages.**
- **Use the features of the language you have chosen.**

Upcoming

- March 27: OpenSees Reliability Analysis

Prof Michael Scott, Oregon State University





http://nees.org/resources/tools/openseeslab

Any Questions?