SMA:
Software Modeling and Analysis

Practical Session

Week 07

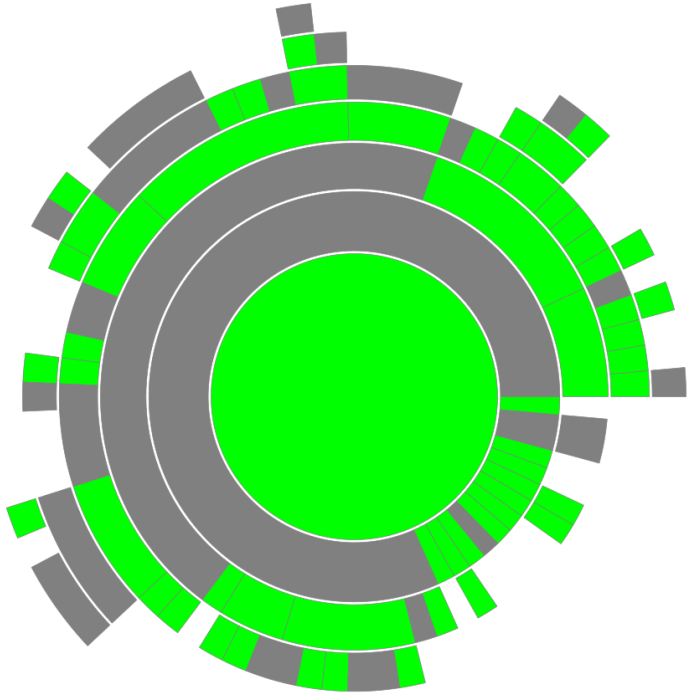
Assignment 06

Discussion

A06 - Exercise 01 | Sunburst Visualization

Smalltalk *coding*.

Build a *sunburst visualization* to analyze test coverage of the `Collection` class hierarchy.

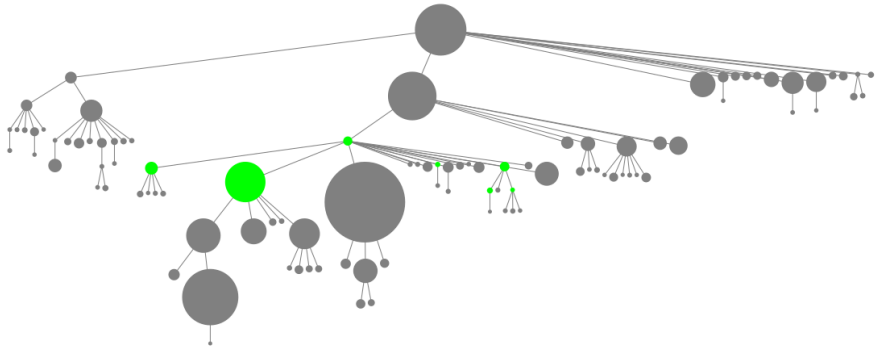


```
| sb |
sb := RSSunburstBuilder new.
sb sliceShape withBorder.
sb sliceColor: [:shape |
  (Smalltalk includesKey: (shape model name , 'Test') asSymbol)
  ifTrue: [ Color green ]
  ifFalse: [ Color gray ] ].
sb explore: Collection using: #subclasses.
RSNormalizer size
  shapes: (sb shapes select: #isSLeaf);
  normalize: #numberOfLinesOfCode.
sb build.
sb canvas @ RSCanvasController.
^ sb canvas
```

A06 - Exercise 02 | Tree Layout Visualization

Smalltalk *coding*.

Build a *tree layout visualization* to gather an overview of classes with subclasses that contain the string `Array` in their names.

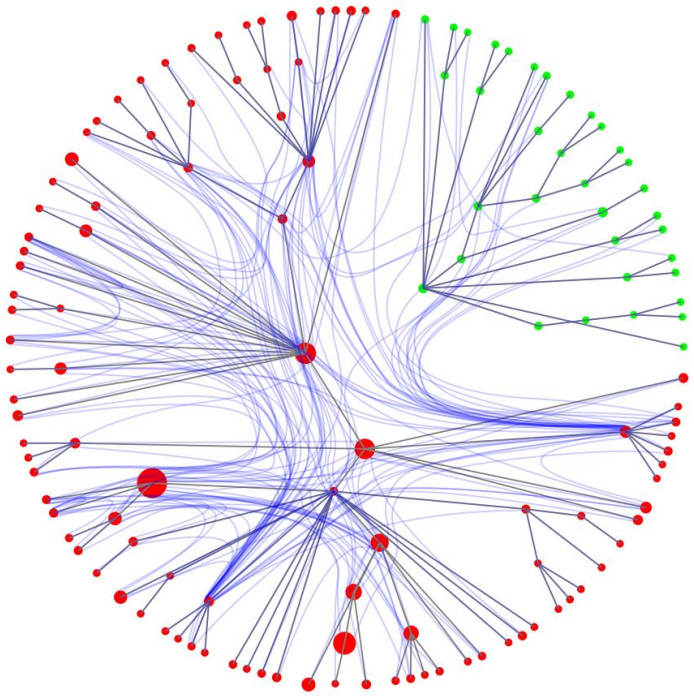


```
| canvas shapes |
canvas := RSCanvas new.
shapes := (Collection withAllSubclasses ) collect:  [ :class |
  (class subclasses notEmpty and:  ['*Array*' match:  class name])
  ifTrue:  [RSEllipse new model:  class; color:  Color green; popup ]
  ifFalse:  [RSEllipse new model:  class; color:  Color gray; popup ]].
canvas addAll:  shapes.
RSEdgeBuilder line
  canvas:  canvas;
  connectFrom:  [ :class | class superclass ].
RSNormalizer size
  shapes:  shapes;
  normalize:  [ :cls | cls numberOfMethods].
RSTreeLayout on:  shapes.
canvas edges pushBack.
canvas zoomToFit.
^ canvas
```

A06 - Exercise 03 | Node-link Visualization

Smalltalk *coding*.

Create a visualization to analyze the class dependencies between the `Collection` class hierarchy and the `RSLayout` class hierarchy.



```
| c classes |
c := RSCanvas new.
classes := RSLayout withAllSubclasses , Collection withAllSubclasses.
classes := classes asSet
  collect: [ :cls | (RSLayout withAllSubclasses includes: cls)
    ifTrue: [RSEllipse new model: cls; color: Color green]
    ifFalse: [RSEllipse new model: cls; color: Color red]]
  as: RSGroup.
c addAll: classes.
RSEdgeBuilder line
  color: Color gray;
  canvas: c;
  shapes: classes;
  connectFrom: #superclass.
RSNormalizer size
  shapes: classes;
  to: 20;
  normalize: #numberOfMethods.
RSClusterLayout on: classes.
RSMultiBezierEdgeBuilder multiBezier
  borderColor: (Color blue alpha: 0.2);
  canvas: c;
  shapes: classes;
  withBorderAttachPoint;
  following: #superclass;
  connectToAll: #dependentClasses.
c @ RSCanvasController.
classes @ RSPopup.
^ c
```

A06 - Exercise 04 | Discussion

Visualization *reasoning*.

Comment on the strenghts and limitations of each visualization you just created.

Sunburst visualization

Strengths: nice for hierarchies, tiles reveal relationship to parent tile
Limitations: hard to evaluate manually (circular sections are hard to estimate)

Tree layout visualization

Strengths: advantages like sunburst + area easier to estimate
Limitations: requires much space

Node-link visualization

Strengths: advantages from sunburst and tree layout,
supports multiple overlaid relation visualizations
Limitations: very high resolution needed for further manual inspection

Assignment 07

Preview

A07 - Exercise 01 | Code metrics in theory

General *knowledge*.

a) What is the cyclomatic complexity? Explain!
b) Which other metrics do you know?
c) Do metrics always express problems?
d) How and when are nowadays metrics integrated into development processes?

A07 - Exercise 02 | Simple code metrics in practice

Writing **code.**

Find all classes that have > 100 methods in `modelArgo`.

A07 - Exercise 03 | Advanced code metrics in practice

***Writing* code and *interpretation* of the results.**

a) Find all methods in `modelArgo` that have:

   1) > 150 lines of code, and

   2) an acyclomatic complexity of < 4

b) Apply your implementation to `modelSolr`.
   Which differences can you see in the result?

c) Is it appropriate to use the same thresholds for any models?
   Justify!

A07 - Exercise 04 | Expert code metrics in practice

Writing **code.**

Add a method to `FAMIXType` to obtain the ATFD metric for its instances.

Important:

1) ATFD counts the attributes from other classes used by a class.

2) We only care for methods that begin with "`set`" or "`get`".