Typen und Konstruktoren (auch Kompatibilität zu existieren Packages (DynamicalSystems.jl etc.) testen und sicherstellen, Redundanz vermeiden wo möglich)

```
BoxCollection(center::Point,radius::Number,initialSubdivisions::Number=1)
BoxCollection(x::Point,y::Point,initialSubdivisions::Number=1)
#Sollten wir das alles als Bäume implementieren, wäre natürlich z.B.
#InitialLevels besser, also log(initialSubdivisions)

BoxCollection(center::Point,radius::Number)
BoxCollection(x::Point,y::Point)

Point(....)
```

Wichtige Funktionen:

```
subdivide(bc::BoxCovering)::BoxCovering
subidivide!(bc::BoxCovering)
subidvide(bc,testFunction)

keep!(bc,TestFunction)
discard!(bc,TestFunction)

subCovering(bc::BoxCovering, ...)::BoxCovering
subCovering(bc::BoxCovering, TestFunction)::BoxCovering

inside(b::Box,p::Point)::Boolean

intersection(bc1::BoxCovering,bc2::BoxCovering)::BoxCovering
union(bc1::BoxCovering,bc2::BoxCovering)::BoxCovering
difference(bc1::BoxCovering,bc2::BoxCovering)::BoxCovering
...

testPoints(bc::BoxCovering, ...)::Array{Point}
testPoints(box::Box)::Array{Point}

plot(bc)
```

Beispielimplementierung Subdivisions-Algorithmus:

```
F = ... #Dynamik

bc = BoxCovering(Point(0,0,0),1.0)

function hits!(box::Box,points::Array{Points})
    #Wahrscheinlich nicht in korrekter Julia Syntax
    pointsThatHit = getAll(points,inside(box,point))
    remove(points,pointsThatHit)
    return !isEmpty(pointsThatHit)
end

for 1:iterations
    subdivide!(bc)
    points = testPoints(bc)
    mappedPoints = F(points)
    keep(bc,box->hits!(box,mappedPoints))
end

plot(bc)
```

Beispielimplementierung Fortsetzungs-Algorithmus:

```
F = ...
bigBoxCovering = BoxCovering(Point(0,0,0),1.0,initialSubdivisions=1000)

manifold = SubCovering(bc,Point(0.2,0.1,0.3),0.01) #Subcovering um einen Fixpunkt
... #Subdivisionsalgorithmus mit manifold

activeBoxes = manifold
for 1:iterations
    mappedPoints = F(testPoints(activeBoxes))
    activeBoxes = SubCovering(bc,box->hits(box,mappedPoints))
    manifold = union(manifold,activeBoxes)
end

plot(manifold)
```

Beispielimplementierung invariantes Maß (vmtl. sehr naiv/ineffizient erstmal):

```
F = ...
bc = BoxCovering(....,initialSubdivisions=10)
#evtl. vorverarbeitung (boxen, die sicher ~0 sind wegwerfen)
A = zeros(length(bc),length(bc))

i = 1
for boxFrom in bc:
    mappedPoints = F(testPoints(box))
    j = 1
    for boxTo in bc:
        A[i,j]=count(inside(boxTo,mappedPoints)) / size(mappedPoints)
        j++
    end
    i++
end

v,invMeasure = eigen(A)

plot(bc,invMeasure) #hier bräuchte man noch eine nette plot methode,
                    #die das farblich darstellt o.Ä.

#Alternativ:
threshold=1/length(bc)
keep!(bc,box -> invMeasure[box.nr]>threshold)
plot(bc)
```