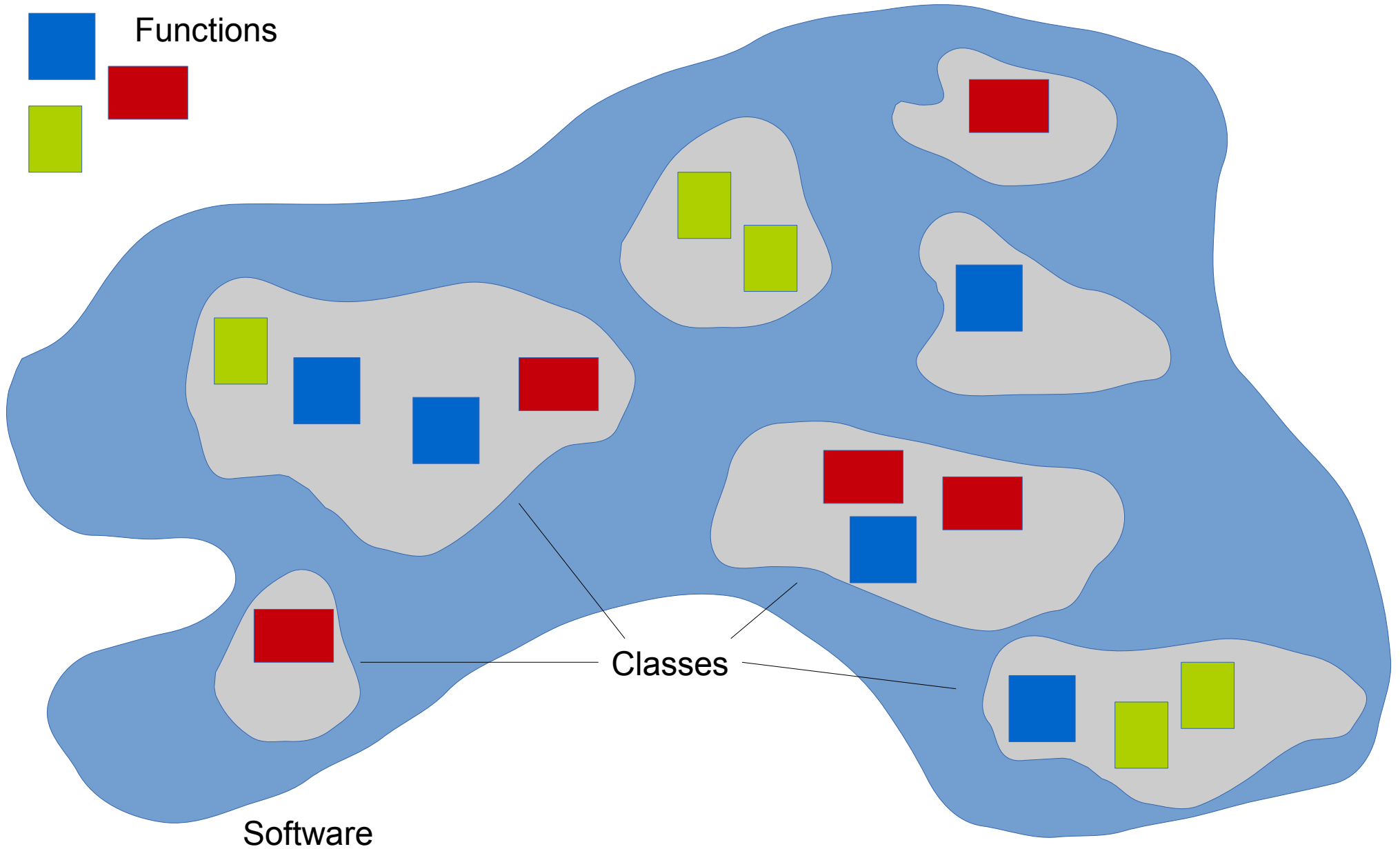
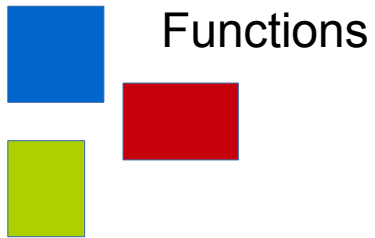


# Clean Code: Functions

01219245/01219246

Individual Software Process

# Basic building blocks



# Let's try to read the code

- You might wonder how your **update** methods are called.
- Let's try to chase it down.

# cc.Director.drawScene

```
335 drawScene: function() {
336     // calculate "global" dt
337     this.calculateDeltaTime();
338
339     //tick before glClear: issue #533
340     if (!this._paused)
341         this._scheduler.update(this._deltaTime);
342
343     this._clear();
344
345     /* to avoid flickr, nextScene MUST be here: after tick and before draw.
346     XXX: Which bug is this one. It seems that it can't be reproduced with v0.9 */
347     if (this._nextScene) {
348         this.setNextScene();
349     }
350
351     if (this._beforeVisitScene) this._beforeVisitScene();
352
353     // draw the scene
354     if (this._runningScene)
355         this._runningScene.visit();
356
357     // draw the notifications node
358     if (this._notificationNode)
359         this._notificationNode.visit();
360
361     if (this._displayStats)
362         this._showStats();
363
364     if (this._afterVisitScene) this._afterVisitScene();
365
366     this._totalFrames++;
367
368     if (this._displayStats)
369         this._calculateMPF();
370 },
```

# cc.Scheduler.update (p.1)

```
559     update:function (dt) {
560         this._updateHashLocked = true;
561
562         if (this._timeScale != 1.0) {
563             dt *= this._timeScale;
564         }
565
566         //Iterate all over the Updates selectors
567         var tmpEntry;
568         var i;
569         for (i = 0; i < this._updatesNegList.length; i++) {
570             tmpEntry = this._updatesNegList[i];
571             if ((!tmpEntry.paused) && (!tmpEntry.markedForDeletion)) {
572                 tmpEntry.target.update(dt);
573             }
574         }
575
576         // updates with priority == 0
577         for (i = 0; i < this._updates0List.length; i++) {
578             tmpEntry = this._updates0List[i];
579             if ((!tmpEntry.paused) && (!tmpEntry.markedForDeletion)) {
580                 tmpEntry.target.update(dt);
581             }
582         }
583
584         // updates with priority > 0
585         for (i = 0; i < this._updatesPosList.length; i++) {
586             tmpEntry = this._updatesPosList[i];
587             if ((!tmpEntry.paused) && (!tmpEntry.markedForDeletion)) {
588                 tmpEntry.target.update(dt);
589             }
590         }
591     }
```

# cc.Scheduler.update (p.2)

```
592 //Iterate all over the custom selectors
593 var elt;
594 for (i = 0; i < this._arrayForTimers.length; i++) {
595     this._currentTarget = this._arrayForTimers[i];
596     elt = this._currentTarget;
597     this._currentTargetSalvaged = false;
598
599     if (!this._currentTarget.paused) {
600         // The 'timers' array may change while inside this loop
601         for (elt.timerIndex = 0; elt.timerIndex < elt.timers.length; elt.timerIndex++) {
602             elt.currentTimer = elt.timers[elt.timerIndex];
603             elt.currentTimerSalvaged = false;
604
605             elt.currentTimer.update(dt);
606             elt.currentTimer = null;
607         }
608     }
609
610     if ((this._currentTargetSalvaged) && (this._currentTarget.timers.length == 0)) {
611         this._removeHashElement(this._currentTarget);
612     }
613 }
614
```

# cc.Scheduler.update (p.3)

```
615         //delete all updates that are marked for deletion
616         // updates with priority < 0
617         for (i = 0; i < this._updatesNegList.length; i++) {
618             if (this._updatesNegList[i].markedForDeletion) {
619                 this._removeUpdateFromHash(this._updatesNegList[i]);
620             }
621         }
622
623         // updates with priority == 0
624         for (i = 0; i < this._updates0List.length; i++) {
625             if (this._updates0List[i].markedForDeletion) {
626                 this._removeUpdateFromHash(this._updates0List[i]);
627             }
628         }
629
630         // updates with priority > 0
631         for (i = 0; i < this._updatesPosList.length; i++) {
632             if (this._updatesPosList[i].markedForDeletion) {
633                 this._removeUpdateFromHash(this._updatesPosList[i]);
634             }
635         }
636
637         this._updateHashLocked = false;
638         this._currentTarget = null;
639     },
```

# SpriteFrameCache

- I am trying to work with spritesheet, but the TexturePacker does not work on my machine.
- I tried open-source darkFunction Editor. It is written in Java, and it runs fine on my machine.
- But the information file it generates is in a different, unusable format. T\_T

```
<?xml version="1.0"?>
<!-- Generated by darkFunction Editor (www.darkfunction.com) -->
<img name="newSpriteSheet.png" w="1024" h="1024">
  <definitions>
    <dir name="/">
      <spr name="0" x="1" y="1" w="40" h="40"/>
      <spr name="1" x="42" y="1" w="40" h="40"/>
    </dir>
  </definitions>
</img>
```



# What should I do?

- I am planning to write a converter.
- So I need to see how Cocos2d-html5 works with their **plist** file.
  - Can I do that? Yes. Cocos2d-html5 is open-source software.
  - This is the beauty of open-source software.

# cc.SpriteFrameCache. addSpriteFrames (p.1)

```
169  /**
170  * <p>
171  *   Adds multiple Sprite Frames from a plist or json file.<br/>
172  *   A texture will be loaded automatically. The texture name will be composed by replacing the .plist or .json s
173  *   If you want to use another texture, you should use the addSpriteFrames:texture method.<br/>
174  * </p>
175  * @param {String} filePath file path
176  * @param {HTMLImageElement|cc.Texture2D|string} texture
177  * @example
178  * // add SpriteFrames to SpriteFrameCache With File
179  * cc.SpriteFrameCache.getInstance().addSpriteFrames(s_grossiniPlist);
180  * cc.SpriteFrameCache.getInstance().addSpriteFrames(s_grossiniJson);
181  */
182  addSpriteFrames: function (filePath, texture) {
183      if (!filePath)
184          throw "cc.SpriteFrameCache.addSpriteFrames(): plist should be non-null";
185
186      var fileUtils = cc.FileUtils.getInstance(), dict;
187      var ext = filePath.substr(filePath.lastIndexOf(".", filePath.length) + 1, filePath.length);
188      if (ext == "plist") {
189          var fullPath = fileUtils.fullPathForFilename(filePath);
190          dict = fileUtils.dictionaryWithContentsOfFileThreadSafe(fullPath);
191      }
192      else {
193          dict = JSON.parse(fileUtils.getTextFileData(filePath));
194      }
195  }
```

# cc.SpriteFrameCache. addSpriteFrame (p.2)

```
196 switch (arguments.length) {
197     case 1:
198         if (!cc.ArrayContainsObject(this._loadedFileNames, filePath)) {
199             var texturePath = "";
200             var metadataDict = dict["metadata"] || dict["meta"];
201             if (metadataDict) {
202                 // try to read texture file name from meta data
203                 texturePath = metadataDict["textureFileName"] || metadataDict["image"];
204             }
205
206             if (texturePath != "") {
207                 // build texture path relative to plist file
208                 texturePath = fileUtils.fullPathFromRelativeFile(texturePath, filePath);
209
210             } else {
211                 // build texture path by replacing file extension
212                 texturePath = filePath;
213
214                 // remove .xxx
215                 var startPos = texturePath.lastIndexOf(".", texturePath.length);
216                 texturePath = texturePath.substr(0, startPos);
217
218                 // append .png
219                 texturePath = texturePath + ".png";
220             }
221
222             var getTexture = cc.TextureCache.getInstance().addImage(texturePath);
223             if (getTexture)
224                 this._addSpriteFramesWithDictionary(dict, getTexture);
225             else
226                 cc.log("cocos2d: cc.SpriteFrameCache: Couldn't load texture");
227         }
228     break;
```

# cc.SpriteFrameCache. addSpriteFrames (p.3)

```
229         case 2:
230             if (texture instanceof cc.Texture2D) {
231                 /** Adds multiple Sprite Frames from a plist file. The texture will be associated with the
created sprite frames. */
232                 this._addSpriteFramesWithDictionary(dict, texture);
233             } else {
234                 /** Adds multiple Sprite Frames from a plist file. The texture will be associated with the
created sprite frames.
235                 @since v0.99.5
236                 */
237                 var textureFileName = texture;
238                 if (!textureFileName)
239                     throw "cc.SpriteFrameCache.addSpriteFrames(): texture name should not be null";
240                 var gTexture = cc.TextureCache.getInstance().addImage(textureFileName);
241
242                 if (gTexture) {
243                     this._addSpriteFramesWithDictionary(dict, gTexture);
244                 } else {
245                     cc.log("cocos2d: cc.SpriteFrameCache: couldn't load texture file. File not found " +
textureFileName);
246                 }
247             }
248             break;
249         default:
250             throw "Argument must be non-nil ";
251     }
252 },
```

# cc.SpriteFrameCache. \_addSpriteFramesWithDictionary (1)

```
49  /**
50  * Adds multiple Sprite Frames with a dictionary. The texture will be associated with the created sprite frames.
51  * @param {object} dictionary
52  * @param {cc.Texture2D} texture
53  */
54  _addSpriteFramesWithDictionary: function (dictionary, texture) {
55      var metadataDict = dictionary["metadata"] || dictionary["meta"];
56      var framesDict = dictionary["frames"];
57
58      var format = 0;
59      // get the format
60      if (metadataDict) {
61          var tmpFormat = metadataDict["format"];
62          format = (tmpFormat.length <= 1) ? parseInt(tmpFormat) : tmpFormat;
63      }
64
65      // check the format
66      if (format < 0 || format > 3) {
67          cc.log("format is not supported for cc.SpriteFrameCache.addSpriteFramesWithDictionary");
68          return;
69      }
70
71      for (var key in framesDict) {
72          var frameDict = framesDict[key];
73          if (frameDict) {
74              var spriteFrame = this._spriteFrames[key];
75              if (spriteFrame) {
76                  continue;
77              }
78
79              if (format == 0) {
80
81              }
82              else if (format == 1 || format == 2) {
83
84              }
85              else if (format == 3) {
86
87              }
88              else {
89
90              }
91
92              if (cc.renderContextType === cc.CANVAS && spriteFrame.isRotated()) {
93                  //clip to canvas
94                  var locTexture = spriteFrame.getTexture();
95                  if (locTexture.isLoaded()) {
96                      var tempElement = spriteFrame.getTexture().getHtmlElementObj();
97                      tempElement = cc.cutRotateImageToCanvas(tempElement, spriteFrame.getRectInPixels());
98                      var tempTexture = new cc.Texture2D();
99                      tempTexture.initWithElement(tempElement);
100                     tempTexture.handleLoadedTexture();
101                     spriteFrame.setTexture(tempTexture);
102
103                     var rect = spriteFrame._rect;
104                     spriteFrame.setRect(cc.rect(0, 0, rect.width, rect.height));
105                 }
106             }
107
108             // add sprite frame
109             var keyName = (filename != null) ? filename : key;
110             this._spriteFrames[keyName] = spriteFrame;
111         }
112     }
113 },
```

# cc.SpriteFrameCache. \_addSpriteFramesWithDictionary (2)

```
79         if (format == 0) {
80             var x = parseFloat(frameDict["x"]);
81             var y = parseFloat(frameDict["y"]);
82             var w = parseFloat(frameDict["width"]);
83             var h = parseFloat(frameDict["height"]);
84             var ox = parseFloat(frameDict["offsetX"]);
85             var oy = parseFloat(frameDict["offsetY"]);
86             var ow = parseInt(frameDict["originalWidth"]);
87             var oh = parseInt(frameDict["originalHeight"]);
88             // check ow/oh
89             if (!ow || !oh) {
90                 cc.log("cocos2d: WARNING: originalWidth/Height not found on the
91             }
92             // Math.abs ow/oh
93             ow = Math.abs(ow);
94             oh = Math.abs(oh);
95             // create frame
96             spriteFrame = cc.SpriteFrame.createWithTexture(texture, cc.rect(
97         }
```

# Small

- Function should be small.
- `cc.Directory.drawScene` is not very small, but small enough to fit this presentation screen.
- `cc.Scheduler.update`, `cc.SpriteFrameCache.addSpriteFrames`, and `cc.SpriteFrameCache._addSpriteFramesWithDictionary` are not.

# Why small?

- What do you think?