

---

## LeadingAgile Code Analysis Tool Data Files

- 1. Code Analysis Data
- 2. Metrics Tool Data
  - 2.1. metrics\_data.json
- 3. Statistics Derived from Metrics
  - 3.1. Statistics
  - 3.2. Trends Over Time
  - 3.3. Counts Over Limit
- 4. Language Distribution
- 5. Commit History Information
  - 5.1. File Modification Frequency
  - 5.2. Files Most-Modified
- 6. GQM Answers
  - 6.1. Answers to Questions
  - 6.2. Team Summary

## 1. Code Analysis Data

The Code Analysis Gather tools create and use data files that are stored as collections in a single folder. The Gather metrics tools and the Gather CLI create this folder structure. Each tool within the suite that creates a data file will use a format specific to that tool. All of the data files are JSON and all follow a general form with some sections within the file specific to the tool at hand.

The basic format is a JSON dictionary object with the basic elements of meta and data.

The meta item contains process and system settings recorded at the time the last data collection that updates the file happens.

The data section contains the data in a form specific to the tool that created the data.

The data files are created in a folder organization to contain data for each Repository, each data run for a repository, and the data for the tools collected in the run.

Each repository analysed will have a top-level folder named after the base folder of the repository.

Within a repository folder will be a folder for each data run named with the run-name given at the time.

Each run folder will have a folders for the tools metrics, statistics, frequency, answers and linguist.

```
.
├── Studios-private-test-data-Alamofire-swift
│   ├── RUN_Aug2021
│   │   ├── answers
│   │   │   ├── cognitive_complexity_reduced.json
│   │   ├── frequency
│   │   │   ├── git_frequency_data.json
│   │   │   ├── git_frequency_statistics.json
│   │   │   ├── plots
│   │   │   │   ├── git_frequency_full.png
│   │   │   │   ├── git_frequency_recent_fifty_percent.png
│   │   │   │   └── git_frequency_recent_twenty_five_percent.png
│   │   ├── linguist
│   │   │   ├── linguist_statistics.json
│   │   │   ├── plots
│   │   │   │   ├── 2019-02-21_linguist.png
│   │   │   │   ├── 2019-03-07_linguist.png
│   │   │   │   ├── 2019-03-29_linguist.png
│   │   │   │   └── 2019-04-06_linguist.png
│   │   ├── metrics
│   │   │   ├── javascript
│   │   │   │   ├── metrics_data.json
│   │   │   ├── swift
│   │   │   │   ├── metrics_data.json
│   │   ├── statistics
│   │   │   ├── javascript
│   │   │   │   ├── counts_over_limit.json
│   │   │   │   ├── statistics.json
│   │   │   │   ├── plots
│   │   │   │   │   ├── complexity.png
│   │   │   │   │   ├── counts_over_limit.png
│   │   │   │   │   ├── method_lines.png
│   │   │   │   └── trend.json
│   │   │   ├── swift
│   │   │   │   ├── counts_over_limit.json
│   │   │   │   ├── statistics.json
│   │   │   │   ├── plots
│   │   │   │   │   ├── balance.png
│   │   │   │   │   ├── complexity.png
│   │   │   │   │   ├── counts_over_limit.png
│   │   │   │   │   ├── method_lines.png
│   │   │   │   └── trend.json
├── Studios-private-test-data-jif-dashboard-javascript
│   ├── RUN_Aug2021
│   │   ├── answers
│   │   │   ├── cognitive_complexity_reduced.json
│   │   ├── frequency
│   │   │   ├── git_frequency_data.json
│   │   │   ├── git_frequency_statistics.json
│   │   ├── plots
│   │   │   ├── git_frequency_full.png
```

```

├── git_frequency_recent_fifty_percent.png
├── git_frequency_recent_twenty_five_percent.png
├── linguist
│   ├── linguist_statistics.json
│   ├── plots
│   │   ├── 2016-12-09_linguist.png
│   │   ├── 2016-12-19_linguist.png
│   │   ├── 2017-03-22_linguist.png
│   │   └── 2018-04-16_linguist.png
├── metrics
│   ├── javascript
│   │   └── metrics_data.json
├── output.log
├── statistics
│   ├── javascript
│   │   ├── counts_over_limit.json
│   │   ├── statistics.json
│   │   ├── plots
│   │   │   ├── complexity.png
│   │   │   ├── counts_over_limit.png
│   │   │   └── method_lines.png
│   └── trend.json
└── team_summary.json

```

## 2. Metrics Tool Data

The metrics data is where the raw static analysis data from the various collection utilities is stored.

The metrics folder under a run folder will have folders for each of the languages analysed. Each language folder will have a `metrics_data.json` file with the metrics collected for that language.

## 2.1. metrics\_data.json

The `metrics_data.json` file contains a json dictionary object comprising elements `meta` and `data`. The `meta` element follows the common structure for meta data (see below).

The data element is a list of dictionaries where each item is the data from a step in the data collection. For a gather run for 52 steps (`-t 52`) where the steps are (roughly) a week each, there will be 52 items in the list, each a dictionary containing details the data for that week. The items in the list are stored in reverse-chronological order of the steps through the git history (i.e. starting with the most recent and going to the oldest).

Each item in the list is a dictionary object with elements:

item	description
revision_hash	the git commit hash (sha) for that step
hash_date	the commit date for that hash
extracted_data	A dictionary of items for each thing being examined (methods, modules, files, framework, e

The `extracted_data` item is a dictionary with items for each metric data value. The key for each item is a unique identifier for the item that the metrics are from (file, method, etc.).

The items within the `extracted_data` are:

item	description
location	A dictionary of items defining the file location for the metric
method_name	The name of the method for the metric (when available). Sometimes these
cyclomatic_complexity	The cyclomatic complexity of the item
method_length	
fan_out_complexity	
lines_of_code	
number_of_comments	
percentage_of_comments	
fan_in	
fan_out	
instability	
number_of_abstracts	
number_of_concretes	
abstractness	
distance_from_main_sequence	
number_of_methods	
number_of_tests	
number_of_imports	

`location` will always be present in some form. Each of the others optional and depend on the specifics of the underlying analysis tools used and how they apply to the particular file type (metrics vary depending on file language and options used during collection).

The `location` item will have as many of these as were available at the time of collection (also varies by analysis tool used):

item	description
file	the file name with path relative to the repository base folder
line	the line number within the file where the metric was collected

	item	description
character		the character position for the metric, when available. This will be something close to the item be

```
{
  "meta": {
  },
  "data": [
    {
      "revision_hash": "757478442696b302bb8f9ad16e93b30f9350d949",
      "hash_date": "2021-03-13T03:51:11-08:00",
      "extracted_data": {
        "a1d8c081f419a29dab6f871881a82b9e": {
          "location": {
            "file": "benchmarks/benchmark-runner.js",
            "line": 6,
            "character": 1
          },
          "method_length": 70,
          "method_name": "anonymous function",
          "cyclomatic_complexity": 9
        }
      }
    },
    {
      "revision_hash": "ad2aaa8c3c2a28b07c550aa99187cbd147a65962",
      "hash_date": "2021-03-06T00:54:40-06:00",
      "extracted_data": {
        "a1d8c081f419a29dab6f871881a82b9e": {
          "location": {
            "file": "benchmarks/benchmark-runner.js",
            "line": 6,
            "character": 1
          },
          "method_length": 70,
          "method_name": "anonymous function",
          "cyclomatic_complexity": 9
        }
      },
      "17538d28cf006865a36d4fd03c5bf262": {
        "location": {
          "file":
            "benchmarks/text-editor-large-file-construction.bench.js",
          "line": 11,
          "character": 1
        },
        "method_length": 86,
        "method_name": "anonymous function",
        "cyclomatic_complexity": 4
      }
    }
  ]
}
```

### 3. Statistics Derived from Metrics

The `statistics` folder will contain a folder for each language found during the metrics collection. Each language folder will have several data files containing statistics calculated from the metrics data. There will also be a `plots` folder containing basic plots of some of the metrics where the plotting can be performed and if that option was enabled during the Gather tool processing.

```
statistics
├── javascript
│   ├── counts_over_limit.json
│   ├── statistics.json
│   └── plots
│       ├── complexity.png
│       ├── counts_over_limit.png
│       └── method_lines.png
└── trend.json
```

#### 3.1. Statistics

The `statistics.json` file is collection of standard statistic measures calculated from the collected data (see above). As with all the data files, this be a dictionary with a meta item and a data item.

The data item is a dictionary of items with keys that are the metric names (see above). There will be one item in the list for each metric type present in the collected data (the metrics data described above).

The value for each of these metric items is a list of dictionary items containing the calculated statistics. There will be one item in the list for each step in the collected data.

	<div>item</div>	<div>description</div>
revision_hash		the git commit hash (sha) for that step
datetime		the commit date for that hash
mean		The mean calculated from the values for that step
median		The median calculated from the values for that step
mode		The mode calculated from the values for that step
min		The minimum value from the values for that step
max		The maximum value from the values for that step
stdev		The standard deviation calculated from the values for that step

In addition to the above, the `method_lines` metric will contain a `max_method` item with a list of the methods which have a length matching the `max` value from the statistics.

Max Method Item elements:

	<u>item</u>	<u>description</u>
name	the name of the method (from the method_name in the metrics data )	
file	the file containing the method (from the location data for the item in the metrics data )	

```
{
  "meta": {
  },
  "data": {
    "complexity": [
      {
        "revision_hash": "757478442696b302bb8f9ad16e93b30f9350d949",
        "datetime": "2021-03-13T03:51:11-08:00",
        "mean": 1.6269,
        "median": 1.0,
        "mode": 1,
        "min": 1,
        "max": 53,
        "stdev": 2.0188372793259095
      },
      {
        "revision_hash": "ad2aaa8c3c2a28b07c550aa99187cbd147a65962",
        "datetime": "2021-03-06T00:54:40-06:00",
        "mean": 1.6281935677787798,
        "median": 1,
        "mode": 1,
        "min": 1,
        "max": 53,
        "stdev": 2.0202190978726846
      }
    ],
    "method_lines": [
      {
        "revision_hash": "757478442696b302bb8f9ad16e93b30f9350d949",
        "datetime": "2021-03-13T03:51:11-08:00",
        "mean": 21.41608827193451,
        "median": 7.0,
        "mode": 3,
        "min": 1,
        "max": 8137,
        "stdev": 128.9954631950077,
        "max_method": [
          {
            "name": "anonymous function",
            "file": "spec/text-editor-spec.js"
          }
        ]
      },
      {
        "revision_hash": "ad2aaa8c3c2a28b07c550aa99187cbd147a65962",
        "datetime": "2021-03-06T00:54:40-06:00",
        "mean": 21.434258680710524,
        "median": 7,
        "mode": 3,

```

```

    "min": 1,
    "max": 8137,
    "stdev": 129.1774944053297,
    "max_method": [
      {
        "name": "anonymous function",
        "file": "spec/text-editor-spec.js"
      }
    ]
  }
}

```

### 3.2. Trends Over Time

The trend.json file is structured as all the other data files with a meta object and a data object.

```

{
  "meta": {
  },
  "data": {
  }
}

```

See above for a description of meta.

The trend data is a collection of calculated data based on the metric statistics in the associated statistics.json.

This is basically the smoothing fit values and the forecast values derived using the Statistics data.

The top-level dictionary in the data object is a collection of items with keys that are the metric names (see above). There will be one item in the list for each metric type present in the collected data. This collection will exactly match the data from the source statistics.json file.

The value for each of these metric items is a dictionary of items containing the calculated smooth-fit values for each statistic measure.

The data for each metric type will be a dictionary with a key for each of the statistic values:

item	description
maxs	
mins	
means	
modes	



item	description
stdevs	
medians	

Each of these items contains a dictionary that holds the caculated values for the smooth-fit curves associated with the metric data:

item	description
fit_data	An array of values of smooth-curve data fitting the actual values from the Statistics data file
forecast_data	An array of smooth-curve data extended a few steps into the future using the fit_data as the
forecast_slope	The slope of the forecast data as calculated from the first forecast point to the last.

```
{
  "meta": {},
  "data": {
    "complexity": {
      "maxs": {
        "fit_data": [
          67.07290995034518,
          67.03886751023525,
          67.01018393349125,
          66.98693630191768,
          66.96896418606795,
          66.955924643061,
          66.94734344139148,
          66.94266119245457,
          66.94127354564931,
          66.94256501813037,
          66.945936377486,
          66.95082577533196,
          66.956724044384,
          66.9631847255387,
          66.96982949093588
        ],
        "forecast_data": [
          50.48784488518306,
          49.00608456546201,
          47.5317330474164,
          46.06475328703747,
          44.6051084255365,
          43.152761788418715,
          41.707676884561835,
          40.269817405299165,
          38.83914722350737,
          37.41563039269871
        ],
        "forecast_slope": -1.3072214492484349
      },
      "mins": {
        "fit_data": [],
        "forecast_data": []
      }
    }
  }
}
```

```

        "forecast_slope": 0.0
    },
    "means": {},
    "modes": {},
    "stdevs": {},
    "medians": {}
},
"method_lines": {}
}

```

### 3.3. Counts Over Limit

The counts\_over\_limit.json file is data showing the number of items (file, methods, etc.) that exceed a defined limit for each metric at each step (week) in the data analysis.

The limit used is currently defined as  $MEAN + (2 * STDEV)$ . Note that this will cause 'over the accepted limit' to sometimes be very much larger than what a fixed value would be. For instance, some might say that 10-20 lines per method is very large. However, in a codebase where say  $MEAN=21$ ,  $MAX=8137$ , and  $STDEV=129$ , then the 'Acceptable Limit' used will be 279. This is much larger than anyone would consider as reasonable for a method length in any language. The purpose of this data is not necessarily to identify all the items over some measure that might be reasonable, but to find the worst of the worst in the current codebase relative to the other items in the codebase. If you really need 'All Methods Over 10 lines', that data is in the metrics data and can be extracted fairly easily.

The data in the counts\_over\_limit.json has the usual meta and data objects. The data object list of objects with one element in the list for each step (week) in the gathered data from metrics. Each item in the list contains:

#### Step Object

	<u>item</u>	<u>description</u>
revision_hash		the git commit hash (sha) for that step
datetime		the commit date for that hash
"metric name"		the name of the metric is the key that has an object has Metric Limit Information

Each metric in the statistics and metric data will have in item in the above element. The key is the metric name and the value is an object containing information about the limits and counts.

#### Metric Limit Information

	item	description
	limit	the limit value used calculated as $\text{MEAN} + (2 * \text{STDEV})$
count_over_limit		the number of items exceeding the limit in the named metric

```
{
  "meta": {},
  "data": [
    {
      "revision_hash":
        "757478442696b302bb8f9ad16e93b30f9350d949",
      "datetime": "2021-03-13T03:51:11-08:00",
      "complexity": {
        "limit": 5.664574558651819,
        "count_over_limit": 344
      },
      "method_lines": {
        "limit": 279.40701466194986,
        "count_over_limit": 77
      }
    },
    {
      "revision_hash":
        "3d5c83be99e1b533127e7284d9fc5559663691fe",
      "datetime": "2021-02-23T07:00:08+03:00",
      "complexity": {
        "limit": 5.668575003601831,
        "count_over_limit": 347
      },
      "method_lines": {
        "limit": 279.7393211233443,
        "count_over_limit": 76
      }
    }
  ]
}
```

## 4. Language Distribution

The linguist data file, `linguist_statistics.json`, contains information about the distribution of the programming language files found at each step (week) of the data analysis. The file content is an object with the usual meta and data objects. The data object contains a list of items where each item is an object of information about the languages in the step.

This information is collected using Github Linguist. The possible languages that it can find is a list that can include language files that other parts of Code Analysis do not analyse (e.g. "Ruby", "Scala", "Haskell", etc.).

### Step Object

	item	description
	revision_hash	the git commit hash (sha) for that step
	hash_date	the commit date for that hash
	extracted_data	Language Information Object

The extracted\_data object contains a dictionary of language information items with a key that is a name of the programming language. There will be one item in the extracted data object for each language-file-type found in the step (week).

### Language Information Object

	item	description
	"Language Name"	The name of the language found
	location	The file location (file name, etc.)
	commits_changed_since_previous_sha	The number of commits to this file since the previous step-commit

```
{
  "meta": {
  },
  "data": [
    {
      "revision_hash": "11d518841365b85b0dca7f7a55174d12911f05a0",
      "hash_date": "2020-11-21T10:26:06-06:00",
      "extracted_data": {
        "Ruby": {
          "size": 3764,
          "percentage": "0.91"
        },
        "Swift": {
          "size": 408270,
          "percentage": "99.07"
        },
        "JavaScript": {
          "size": 77,
          "percentage": "0.02"
        }
      }
    },
    {
      "revision_hash": "ebc0f1ce8dd0e7edd491a7573ffc40f7c528967a",
      "hash_date": "2020-11-09T16:23:11-06:00",
      "extracted_data": {
        "Ruby": {
          "size": 3764,
          "percentage": "0.91"
        },
        "Swift": {
          "size": 408218,
          "percentage": "99.07"
        }
      }
    }
  ]
}
```

```

    },
    "JavaScript": {
      "size": 77,
      "percentage": "0.02"
    }
  }
}
]
}

```

## 5. Commit History Information

### 5.1. File Modification Frequency

The frequency data file, `git_frequency_data.json`, contains information about number of times each modified file was part of a commit between two steps (weeks) of the data analysis. The file content is an object with the usual meta and data objects. The data object contains a list of items where each item is an object of information about the files modified.

#### Step Object

	item	description
	<code>revision_hash</code>	the git commit hash (sha) for that step
	<code>hash_date</code>	the commit date for that hash
	<code>extracted_data</code>	File Change Information

The `extracted_data` object contains a dictionary of file information items with a key that is a unique file identifier. There will be one item in the extracted data object for each file changed in this step (week) (i.e. since the previous step-commit).

#### File Change Information

	item	description
	"File Identifier"	a unique value identifying the file. A change recorded for this file in
	location	The file location (file name, etc.)
	<code>commits_changed_since_previous_sha</code>	The number of commits to this file since the previous step-commit

```

{
  "meta": {
  },
  "data": [
    {

```

```

    "revision_hash":
      "a607fc1c8f4a32cb1fa6fdc6bc760b3d1cc3355d",
    "hash_date": "2019-03-06T17:05:43-05:00",
    "extracted_data": {
      "d18193369129c106d3efaab6103a7b74": {
        "location": {
          "file": "Documentation/CONTRIBUTOR.md"
        },
        "commits_changed_since_previous_sha": 1
      },
      "008b9615f6f1142f7260ad4a03a62bf9": {
        "location": {
          "file": "Documentation/MIGRATING.md"
        },
        "commits_changed_since_previous_sha": 1
      },
      "2bcf996b6de98d42743b4ac18b72c5f1": {
        "location": {
          "file": "Documentation/images/animatordNode.png"
        },
        "commits_changed_since_previous_sha": 1
      }
    }
  },
  {
    "revision_hash":
      "8bb6ecfe46a148bab26963f2a359736ef6ad768a",
    "hash_date": "2019-02-12T15:20:40-08:00",
    "extracted_data": {
      "a9eca5647374360f70ae80fd19d81b80": {
        "location": {
          "file":
            "lottie-ios/Classes/Private/LOTComposition.m"
        },
        "commits_changed_since_previous_sha": 1
      }
    }
  }
]
}

```

## 5.2. Files Most-Modified

The file `git_frequency_statistics.json` is an extract of information from the `git_frequency_data.json` to reduce to the counts for each file identified. It contains the usual meta and data objects. The data object is a dictionary with objects that hold subsets of the data covering the full, the most recent half, and the most recent quarter time span out of the span of time covered by the Code Analysis run.

### Data Object

	item	description
	full	a time span object covering the full range of time of the Analysis
	recent_fifty_percent	a time span object covering the most recent 50% of time of the Analysis
	recent_twenty_five_percent	a time span object covering the most recent 25% of time of the Analysis

The Time Span object which is the value of each of the above keys contains items for

### Time Span Object (file set)

	item	description
"file paths"		the path of the file relative to the repository base folder (there will one item for each file). The

```
{
  "meta": {
  },
  "data": {
    "full": {
      "lottie-ios/Classes/PublicHeaders/LOTAnimationView.h": 3,
      "lottie-ios/Classes/Private/LOTComposition.m": 3,
      "Documentation/CONTRIBUTOR.md": 4,
      "Documentation/MIGRATING.md": 4,
      "Documentation/images/animatorNode.png": 2,
      "Example/Podfile.lock": 26,
      "Example/Pods/Local Podspecs/lottie-ios.podspec.json": 22,
      "Example/Pods/Local Podspecs/lottie-swift.podspec.json": 2,
      "Example/Pods/Manifest.lock": 26,
      "Example/Pods/Pods.xcodeproj/project.pbxproj": 21,
      "Example/Pods/Target Support Files/Pods-Lottie
        Viewer/Pods-Lottie Viewer-dummy.m": 1,
      "Example/Pods/Target Support Files/Pods-Lottie
        Viewer/Pods-Lottie Viewer.modulemap": 1,
      "Example/Pods/Target Support
        Files/Pods-lottie-swift_macOS/Pods-lottie-swift_macOS-frameworks.sh":
        10,
      "Example/Pods/Target Support
        Files/Pods-lottie-swift_macOS/Pods-lottie-swift_macOS-resources.sh":
        7,
      "Example/lottie-swift/ViewController.swift": 16,
      "Example/lottie-swift_macOS/AppDelegate.swift": 1,
      "README.md": 20,
      "_Gifs/HeartButton.gif": 1,
      "lottie-ios.podspec": 24,
      "lottie-ios/Classes/.gitkeep": 1,
      "lottie-swift/src/Private/LayerContainers/CompLayers/TextCompositionLayer.swift":
        26,
      "lottie-swift/src/Private/LayerContainers/Utility/CompositionLayersInitializer.swift":
        4,
      "lottie-swift/src/Private/LayerContainers/Utility/InvertedMatteLayer.swift":
        4,
      "lottie-swift/src/Private/LayerContainers/Utility/LayerImageProvider.swift":
        2
    }
  }
}
```

```

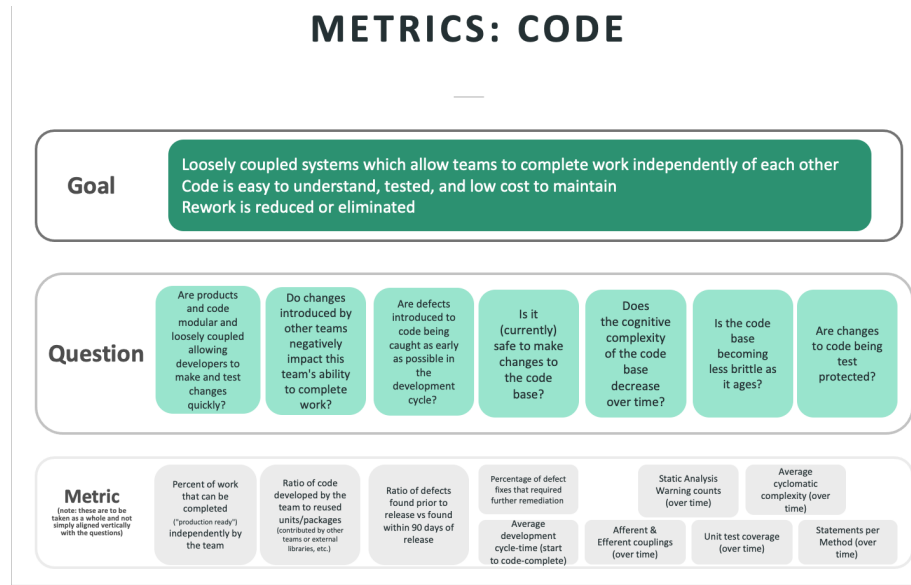
    },
    "recent_fifty_percent": {
        "lottie-swift/src/Private/Utility/Extensions/MathKit.swift": 2,
        "lottie-swift/src/Private/NodeRenderSystem/Nodes/OutputNodes/Renderables/GradientFillRenderable.swift": 2,
        "Lottie.xcodeproj/project.pbxproj": 3,
        "lottie-swift/src/Public/DynamicProperties/ValueProviders/ColorsValueProvider.swift": 1,
        "lottie-swift/src/Public/iOS/AnimatedControl.swift": 4,
        "lottie-swift-testing.podspec": 2,
        "_Gifs/spm-branch.png": 2,
        "README.md": 5,
        "Package.swift": 2,
        "lottie-swift/src/Public/Animation/AnimationView.swift": 20,
        "lottie-swift/src/Public/AnimationCache/LRUAnimationCache.swift": 2
    },
    "recent_twenty_five_percent": {
        "lottie-swift/src/Public/iOS/AnimatedControl.swift": 2,
        "lottie-swift/src/Public/Animation/AnimationView.swift": 5,
        "lottie-swift/src/Private/Utility/Extensions/AnimationKeypathExtension.swift": 1,
        "lottie-swift/src/Public/MacOS/BundleImageProvider.swift": 2,
        "lottie-swift/src/Public/iOS/BundleImageProvider.swift": 1,
        "Example/Pods/Pods.xcodeproj/project.pbxproj": 2,
        "Example/lottie-swift/ViewController.swift": 2,
        "Lottie.xcodeproj/project.pbxproj": 1,
        "lottie-swift/src/Private/LayerContainers/AnimationContainer.swift": 1,
        "lottie-swift/src/Private/LayerContainers/CompLayers/PreCompositionLayer.swift": 1,
        "lottie-swift/src/Private/LayerContainers/CompLayers/TextCompositionLayer.swift": 3,
        "lottie-swift/src/Private/LayerContainers/Utility/CompositionLayersInitializer.swift": 1,
        "lottie-swift/src/Private/LayerContainers/Utility/LayerFontProvider.swift": 1,
        "lottie-swift/src/Private/LayerContainers/Utility/TextLayer.swift": 5,
        "lottie-swift/src/Public/FontProvider/AnimationFontProvider.swift": 1,
        "lottie-swift/src/Public/iOS/Compatibility/CompatibleAnimationView.swift": 1,
        "README.md": 1
    }
}
}

```

## 6. GQM Answers

The Code Analysis data is used to provide an answer, or at least some guidance, for the set of Goal-Question-Metrics questions we are asking.





In this version of Code Analysis we generate an 'answer' for the question: **Does the cognitive complexity of the code base decrease over time?**

We calculate this estimate of the answer using the trends of the metrics collected. We record an answer to the question for each language that is analysed. We also record and summary answer using the results for each language weighted by the volume of each in the repostitory (see the Linguist data above).

The Answer is determined by comparing Evaluations of the metrics used (currently: Cyclomatic Complexity and Method Length). A Metric Evaluation is calculated by comparing the Average Score and Trend Score of the metrics. Scores for a metric are calculated by comparing the average (mean) of the metric result and the slope of the trend-over-time of the metric data over the time period of the full Code Analysis collection.

A metric average (mean) is 'Scored' by considering its value to some absolutes chosen based on our extensive experience in many languages. The Score for a metric level will be High, Elavated, or Low.

A metric trend is 'Scored' by considering its slope over the time of the analysis. The Score for a metric trend will be Increading, Flat, or Decreasing.

### Metric Scores

	Metric Average Level	Metric Trend			
	High	Elevated	Low	Increasing	Flat
Cyclomatic complexity	>6	Between 3 & 6	<3	Slope >1	Slope Between 1 & 2
Method Length	>10	Between 6 & 10	<6	Slope >1	Slope Between 1 & 2

## Metric Evaluation Based on Scores

		Metric Trend Level		
Metric Average	If High	If Increasing	If Flat	If Decreasing
	If Elevated	Real Bad	Bad	Warning
	If Low	Bad	Warning	Good
		Warning	Good	Great

Table 18: Cognitive Complexity from Evaluations

		Method Length				
Cyclomatic complexity	Real Bad	Real Bad	Bad	Warning	Good	Great
	Bad	No	No	No	No	Inconsistent
	Warning	No	No	No	Inconsistent	Inconsistent
	Good	No	No	Watch Listed	Watch Listed	Watch Listed
	Great	Inconsistent	Inconsistent	Watch Listed	Yes	Yes
		Inconsistent	Inconsistent	Watch Listed	Yes	Yes

While we only answer the one question in this version of Code Analysis, we intend to expand this to other Questions and to expand the set of questions as we discover what is most useful to our clients.

## 6.1. Answers to Questions

The `cognitive_complexity_reduced.json` data file holds the 'answer' data for the question: **Does the cognitive complexity of the code base decrease over time?**

This file contains an top-level object which has the usual meta and data items. The data item Repository Answer Object containing the summary and language-specific data and answers for the question for the current repository.

### Repository Answer Object

	item	description
summary		The Summary Answer object
"language name"		A language Answer Data object. There will be one for each language analysed.

### Summary Answer Object

	item	description
question		The text of the question being answered
answer		The answer generated for the question in an Answer Object as calculated from the weighted evaluation
evaluation		A dictionary of Evaluation Object items with the results of combining the weighted evaluation
scaled_weights		A dictionary with keys of the languages used in answering the question with values of the weights

### Answer Object

	item	description
name		The name for the answer
value		A human-language (sort of) result for the answer

### Evaluation Object

	item	description
"metric name"		A Metric Evaluation Object. There will be one MetricName:Evaluation pair for each metric used

### Metric Evaluation Object

	item	description
name		The name of the evaluation result
value		The numeric value of the evaluation result

### Language Answer Object

	item	description
question		The text of the question being answered
answer		The answer generated for the question in an Answer Object (see above) for the current language
score		A dictionary of Score Object items with the score data for each metric. The keys for the metrics are the metric names
evaluation		A dictionary of Evaluation Object items (see above) for the current language. The keys for the metrics are the metric names
percentage_of_repo		The percentage weight the current language has in the overall repository

```
{
  "meta": {
    "name": "WARNING",
    "description": "Warning: This repository is not a standard repository."
  },
  "data": {
    "summary": {
      "evaluation": {
        "complexity": {
          "name": "WARNING",
          "value": 0.5
        }
      }
    }
  }
}
```

```

        "value": 3
      },
      "method_lines": {
        "name": "WARNING",
        "value": 3
      }
    },
    "question": "Does the cognitive complexity of the code base
      decrease over time?",
    "answer": {
      "name": "WATCH_LISTED",
      "value": "Better not tell you now"
    },
    "scaled_weights": {
      "swift": 99.97981632859018,
      "javascript": 0.02018367140982945
    }
  },
  "swift": {
    "score": {
      "complexity": {
        "average": {
          "name": "ELEVATED",
          "value": 2
        },
        "trend": {
          "name": "FLAT",
          "value": 2
        }
      },
      "method_lines": {
        "average": {
          "name": "ELEVATED",
          "value": 2
        },
        "trend": {
          "name": "FLAT",
          "value": 2
        }
      }
    },
    "evaluation": {
      "complexity": {
        "name": "WARNING",
        "value": 3
      },
      "method_lines": {
        "name": "WARNING",
        "value": 3
      }
    },
    "question": "Does the cognitive complexity of the code base
      decrease over time?",
    "answer": {
      "name": "WATCH_LISTED",
      "value": "Better not tell you now"
    },
  },

```

```
        "percentage_of_repo": 99.97981632859018
      },
      "javascript": {
        }
      }
    }
```

6.2. Team Summary

The team\_summary.json holds data for each **Run** which combines all of the data from the cognitive\_complexity\_reduced.json for each each **Repository** to generate a Team Summary Answer for the run. The Team Summary Answers for all runs are stored in this file. Note that this file is stored at the base folder of the results collection (identified as the --output-folder when the Code Analysis tool is run.)

This file contains an top-level object which has the usual meta and data items. The data item is a dictionary whose keys are **Run Names** for objects containing the Team Summary Object data items.

Team Summary Object

	<u>item</u>	<u>description</u>
repo_answers	A dictionary of Repository Answer Object items (see above). The dictionary keys are the name of the repository.	
summary	A Team Summary Answer object	

Team Summary Answer Object

	<u>item</u>	<u>description</u>
question	The text of the question being answered	
answer	The answer generated for the question in an Answer Object as calculated from the weighted answers	

```
{
  "meta": {
  },
  "data": {
    "2022_Q1": {
      "repo_answers": {
        "product_client": {
          "evaluation": {
            "complexity": {
              "name": "GOOD",
              "value": 4
            },
            "method_lines": {
              "name": "GOOD",
              "value": 4
            }
          }
        }
      }
    }
  }
}
```

```

        "value": 4
      },
      "question": "Does the cognitive complexity of the code
        base decrease over time?",
      "answer": {
        "name": "YES",
        "value": "It is decidedly so"
      },
      "scaled_weights": {
        "swift": 100.00000000000001
      }
    },
    "product_library": {
    },
    "product_server": {
    }
  },
  "summary": {
    "question": "Does the cognitive complexity of the code base
      decrease over time?",
    "answer": {
      "name": "INCONSISTENT",
      "value": "Outlook not so good"
    }
  }
},
"2021_Annual": {
}
}

```