# COMPETITORS

Quality code analysis is very hard to do.

* What is your definition of good code?
  + is it fast code?
  + is it low energy use?
  + is it easy to read code?
  + is maintainable code?

There are so many opposing paradigms. It depends on the business need which way code should be written.

* If it is financial trading it needs to be faster than competitors code to make the trade.
* If it changes often then it needs to be maintainable.
* If it is worked on by many developers it needs to easy to read.
* If it runs on hundreds of machines, like datacentres then the energy use is probably important.

Performant code is horrible to read, variables have single letter names, methods are long and full of loops. If it is maintainable it uses patterns which can are confusing and increases the time it takes developers to learn the system before they can make changes.

Each development team will have evolved styles that suit them, they resent being told to implement code changes for readability when they need performance.

## How is Codemology different

* Codemology looks at the punters’ code, establishes a pattern and then marks all files as to how far the drift from that pattern.
  + It makes no judgement on a developers’ choices.
  + No competitor can currently do this.
* Codemology analyses the text of the code, not the commits or reviews
* It uses hundreds of metrics not 2 as SonarCube
  + is very nuanced indeed
* It does not rely on third party tools so has no fees to pay
  + nor is it subject to exterior tool changes.
* All languages are directly comparable.
* It produces a single consistent ENTROPY value to describe a code body:
  + whether it is a single file or a solution consisting of hundreds of projects and tens of thousands of files
  + whether that code body is written in Python, C# or R
  + the same body of code will ALWAYS produce the same entropy.
  + Entropy is directly comparable between different files, projects, teams, developers, managers etc
  + Entropy can be set as a target in legal documents, eg outsourcing.
  + Entropy is easy to understand to understand for technical and non-technical teams
  + Entropy can be plotted against time, developer, file, team providing deep drillable insight into the evolution of code without development knowledge across projects, teams, developers, managers, organisations.

## Norms in this category and their relevance

* Code analysis is usually performed by code owners. They read all the code the developer is suggesting should be changed. It takes a lot of time, and subject to human error like everything else.
* In a large project there will be pockets of poor code scattered across it. How do you document where that code is? How do you prioritise and set about fixing it? How do you take on a project you are not familiar with? It is a constant career worry for all managers and senior developers.
* No product has managed to solve this, instead tools are built that chip away at the edges, they target specific security flaws eg a password hardcoded into a connection string, using http instead of https etc.
* Instead some products target git data: code reviews, commits, pull requests to show hotspots, “who owns what”, dead files, and from this assume code quality. But this is after the fact and excludes code that doesn’t have lots of commits etc.
* Developers also attack this by writing tests, to trap the obvious mistakes they make.
* But all this leaves a big block of unknown especially in a large body of work. And how does a non technical manager understand use thses tools to understand their code quality? Do they rely on their developers?
* It is this unknown that Codemology goes after. Codemology is looking at human behaviour, it learns how you write code and then points you at the code that doesn’t fit. Grading it. When you fix a piece of code it falls back and the next worst pieces of code are pushed to the front. And so on until you are happy. It does that by processing hundreds, in some cases thousands of metrics down to the statement level. It has been proven in experiment to pull out what the owners of that code regard as poor code.
* Codemology does not compete directly with any other product, indeed it complements them. It is a new avenue for analysing code that we can’t currently investigate. The theory and approach have patents pending in the EU, US and Japan.

## Learning from best in class competitors

DOs

Website:

* Build a clean simple website - https://www.scitools.com
* Use as few words as possible, make them easy to read - <https://www.hey.com/features/>
* If a phrase conveys no obvious meaning remove it.
* For the GitHub app we will be dealing with individual developers not managers
  + Copy websites that target them like
  + <https://www.devexpress.com/products/coderush/>
    - Use a walkthrough video to explain how to install and what the developer gains
  + <https://www.jetbrains.com/upsource/>
    - For instructions on creating github app
    - <https://www.jetbrains.com/help/idea/github.html#register-account>
  + Detail problems solved, features, call to action, who is using, and pricing on same page
* Keep the website looking advanced – it implies the company and product are up to date too
* Remove all impediments to sign up – it needs to be so frictionless they can do it in their sleep.

Email:

* If a punter replies to an email acknowledge it with a reply and thank/wish them well be the one whose email is the last in the chain.
* Reply quickly to email enquiries – 15 minutes.

If Codemology is wrong we need to find out today – then we can improve it for tomorrow.

Listen to punters – they know far more about what they need than we do

DON’Ts

Website:

* Be corporate or vague
* Use filler text - a punter’s time is VERY valuable to them – don't waste half a second of it
  + don't string punters along
  + don't show happy-clappy marketing videos
  + Don't have animated intros
* Don’t make them contact you to get information or to get access
* Don’t have any impediments to sign up

Email:

* Don’t send an email with a doc without a personal note
* Don’t expect them to sign up
* Don’t get offended if they just want info or offer an opinion – listen

Don’t lecture the punter

## Who is currently in this market?

*Strengths, Weaknesses, Opportunities, Threats*

**SONARCUBE**

https://www.sonarqube.org/

Strengths

* Handles 27 languages
* Addresses both code quality AND code security
* Fits in existing tools and pipelines
* Feedback during code review

Weaknesses

* Relies on a host of third party tools for pulling out metrics
  + Does not compare your code to your coding style instead making arbitrary judgements about your code that you will find simplistic and annoying.
  + Fees are payable to third parties
  + Not possible to directly compare different languages
  + Reliant on those services not changing their systems and affecting the metrics
  + No single number can be used to describe a file, project or solution
  + It is an absolute sod to implement all the stuff to analyse projects – worse if several lprogramming anguages involved
  + In C# the are many versions of the compiler, different versions of the code need different compilers, those compilers need to be installed and maintained to run metrics
  + If the code doesn’t build you don’t get any metrics.
  + Codemology assembles its own metrics from the code directly
* Only has 2 metrics for C# and likely similar numbers for other languages – this is shocking
  + (codemology relies on hundreds of metrics)

Opportunities

* They produce a 1950s Russian tractor. It has 2 metrics, produces inconsistent results that change depending on language, compiler version. Very useful to compare against.

Threats

* Large established, well known company, with financial clout

**SEMA**

<https://semasoftware.com>

Strengths

* Seems to make noises as if it can do what Codemology does. The following comes from https://semasoftware.com/codebase-solutions

“We’ve built the only solution that can:

* + Comprehensively assess code, process and team.
  + Analyze developer quality, not just quantity.
  + Express the results in ways that technologists and non-technologists can understand together.”

Weaknesses

* Website is confused with no details of how it works or how it has to be implemented
* Seems too hyped up

Opportunities

* Just launched startup – none yet

Threats

* Just launched startup – none yet

**CODACY**

<https://www.codacy.com/product>

Strengths

* Seems to be a possible competitor. The following comes from https://www.codacy.com/product

##### “Track your quality evolution

Drill down on the evolution of your project's quality with alerts on the most problematic areas.”

Weaknesses

* In spite of the above link it feels like just a code review tool
* Finds issues after they happen – ie there is a commit clash
  + Codemology is one step ahead - it analyses the code and pulls out problems before commit clashes or where commit clashes might never even happen.
  + It can pull out the issues equally well in a project that has a single commit which the above cannot even analyse.

Opportunities

Threats

**CODEGRIP**

<https://www.codegrip.tech>

Strengths

Weaknesses

* Like the other relies on “code smells” - that they define for you ( a pain) or you have to define (a lot of work that constantly needs updating and you can only look for what you know to be a problem).
* Each language has to be defined separately by the client unless you accept off the peg “code smells”

Opportunities

Threats

**PERFORCE**

<https://www.perforce.com>

Strengths

Weaknesses

* Corporate offering
* Again based on defined “code smells”. If you don’t define it you won’t find it.
* From https://www.perforce.com/blog/sca/what-static-analysis

“Static code analysis is a method of debugging by examining source code before a program is run. It’s done by analyzing a set of code against a set (or multiple sets) of coding rules.”

Opportunities

Threats

**SYNOPSYS**

<https://www.synopsys.com>

Strengths

Weaknesses

* Weaknesses Corporate offering
* Again based on defined “code smells”.

Opportunities

Threats

**CRUCIBLE**

<https://www.atlassian.com/software/crucible>

Strengths

* Part of Atlassian which includes BitBucket a small competitor to GitHub

Weaknesses

* Based on defined “code smells” and code reviews

Opportunities

Threats

**UPSOURCE**

<https://www.jetbrains.com/upsource/>

Strengths

* Part of JetBrains a famous Visual Studio plugin

Weaknesses

* Based on defined “code smells” and code reviews

Opportunities

Threats

**PHACILITY**

<https://www.phacility.com/phabricator/>

Strengths

Weaknesses

* “code smells” and file watchers

Opportunities

Threats

* Feels dated even compared to the rest of the competition

**ESLINT**

<https://eslint.org/>

Strengths

* Open source real-time error checker as you type
* Highlights compile time errors: checking types, parameters, spelling mistakes etc
* Ubiquitous
* Indispensable

Weaknesses

Opportunities

Threats

**CODE CLIMATE**

<https://codeclimate.com/velocity>

Strengths

Weaknesses

* Analyses commits, commit comments, pull requests not the code

Opportunities

Threats

**UNDERSTAND**

<https://www.scitools.com/>

Strengths

* Clean website

Weaknesses

* Still not clear what it does looks like a EIDE that has deeper insights to linkages in code

Opportunities

Threats

**SIDER**

<https://sider.review/>

Strengths

* Run many lints in one place

Weaknesses

* Based on rules

Opportunities

Threats

**MOOSE**

<https://modularmoose.org/>

Strengths

* Open source
* A little bit like a compiler or the Codemology Abacus
* Can query software rather like Semmle – but need to know what to look for first...

Weaknesses

* English is weird - impacts confidence

Opportunities

Threats

**LDRA**

<https://www.ldra.com/>

Strengths

* 40 years experience
* Aeronautical, automotive, rail, medical and power generation

Weaknesses

* ISO standards – not right for fast paced development
* Relies on code rules

Opportunities

Threats

**CODESCENE**

<https://codescene.com>

Strengths

* All major languages
* GitHub, GitLab, BitBucket, Azure DevOps

Weaknesses

* Analyses commits, pull requests etc

Opportunities

Threats

**CODERUSH**

<https://www.devexpress.com/products/coderush/>

Strengths

Weaknesses

* Only Visual Studio plugin
* Like a Lint
* Relies on code rules

Opportunities

Threats

**SEMMLE**

<https://semmle.com/>

Strengths

* Bought by GitHub (Microsoft) soon after Dependabot
* Will be ubiquitous soon as GitHub will use it on all its repos to alert the owners of possible trouble.
* GitHub intends to write the queries itself saving the user work
* Allows queries to be written against code to look for security vulnerabilities
* Will eventually become a feature that makes companies move to GitHub

Weaknesses

* Have to write those queries if not using the GitHub ones

Opportunities

* Codemology complements Semmle and Dependabot
* We are in contact with Grey Baker – Senior Director of security at GitHub, founder of Dependabot

Threats

* None really – it seems to be under Grey’s remit

**CODESONAR**

<https://www.grammatech.com/products/source-code-analysis>

Strengths

Weaknesses

* Uses the built in compiler
* Not a SAAS
* Pattern matching for finding problems

Opportunities

Threats

**EMBOLD**

<https://embold.io/product/>

Strengths

Weaknesses

* “code smells”

Opportunities

Threats