

Tabla 1. Líneas de código efectivas por proyecto y lenguaje de programación.

Mi idea principal para realizar esta tabla era utilizar la API de GitHub y obtener los lenguajes con los que trabaja el proyecto como tal. Pero después encontré un herramienta llamada cloc. Cloc cuenta líneas en blanco, líneas de comentarios y líneas de código en diferentes lenguajes de programación.

La instalación es muy fácil <https://github.com/AlDanial/cloc> y haciendo una prueba eso fue un resultado que me arrojó con el proyecto de provisioning

```
[taniamedina@192 Downloads % cd provisioning
[taniamedina@192 provisioning % cloc . --exclude-dir=deps
  254 text files.
  104 unique files.
  5186 files ignored.

github.com/AlDanial/cloc v 1.92  T=0.72 s (143.6 files/s, 86579.7 lines/s)
-----
Language             files      blank      comment      code
-----
HTML                  1         19212          0         32826
Elixir                59         1312          630         7358
JSON                  28           0           0         1096
Erlang                12           0           0          110
Markdown              2           44           0           82
Bourne Shell          1           5           6           23
XML                   1           0           0           20
-----
SUM:                  104        20573         636        41515
-----
taniamedina@192 provisioning %
```

Figura 1. Grafo de dependencias entre los distintos repositorios de MachineCorpz.

Para obtener un diagrama con las dependencias del proyecto, creo que es buena idea utilizar mix deps.tree. Que nos arrojará algo parecido a la imagen de abajo. Obviamente esto se tendría que realizar con todo el proyecto como tal. Ya que en este caso solo se está realizando con provisioning.

```

[taniamedina@192 provisioning % mix deps.tree
provisioning
├── adm >= 0.2.3 (Hex package)
│   ├── bcrypt_elixir ~> 2.0 (Hex package)
│   │   ├── comeonin ~> 5.3 (Hex package)
│   │   └── elixir_make ~> 0.6 (Hex package)
│   └── ecto_sql ~> 3.2 (Hex package)
│       ├── db_connection ~> 2.2 (Hex package)
│       │   └── connection ~> 1.0 (Hex package)
│       ├── ecto ~> 3.5.0 (Hex package)
│       │   ├── decimal ~> 1.6 or ~> 2.0 (Hex package)
│       │   ├── jason ~> 1.0 (Hex package)
│       │   │   └── decimal ~> 1.0 or ~> 2.0 (Hex package)
│       │   └── telemetry ~> 0.4 (Hex package)
│       ├── postgrex ~> 0.15.0 or ~> 1.0 (Hex package)
│       └── telemetry ~> 0.4.0 (Hex package)
├── guardian ~> 2.1.0 (Hex package)
│   ├── jose ~> 1.8 (Hex package)
│   └── plug ~> 1.3.3 or ~> 1.4 (Hex package)
│       ├── mime ~> 1.0 (Hex package)
│       ├── plug_crypto ~> 1.1.1 or ~> 1.2 (Hex package)
│       └── telemetry ~> 0.4 (Hex package)
├── luhn ~> 0.3.3 (Hex package)
├── money ~> 1.4 (Hex package)
├── pbkdf2_elixir ~> 1.0 (Hex package)
│   └── comeonin ~> 5.3 (Hex package)
├── plug_cowboy ~> 2.0 (Hex package)
├── postgrex ~> 0.15.0 (Hex package)
│   ├── connection ~> 1.0 (Hex package)
│   ├── db_connection ~> 2.1 (Hex package)
│   ├── decimal ~> 1.5 or ~> 2.0 (Hex package)
│   └── jason ~> 1.0 (Hex package)
├── scrivener_ecto ~> 2.0 (Hex package)
│   ├── ecto ~> 3.3 (Hex package)
│   └── scrivener ~> 2.4 (Hex package)
├── timex ~> 3.5 (Hex package)
└── altan ~> 0.1 (Hex package)
    ├── csv ~> 2.3 (Hex package)
    ├── parallel_stream ~> 1.0.4 (Hex package)
    └── flow ~> 1.0 (Hex package)

```

Tabla 2. Métricas de pruebas por proyecto.

Con el comando `mix test` se pueden obtener el total de pruebas y cuantas de esas pruebas son exitosas o cuantas fallan. Para obtener el % de cobertura se puede hacer uso de Excoveralls y tan solo con hacer `mix coveralls` se puede obtener esa información.

Tabla 3. Warnings de compilación por proyecto.

Con el comando `mix compile` se puede apreciar si tu código no tiene algún error o si presenta warnings en el código. Es muy bueno ya que te indica el porqué del error o del warning y es más fácil de corregir.

Tabla 4. Conto de warnings clasificados por tipo. Los tipos que están en rojo son potencialmente bugs

Yo creo que existe algún comando donde le puedes indicar que te muestre la cantidad de warnings dependiendo el tipo de warning que quieras consultar. De momento no lo pude

encontrar. Solo se me ocurre categorizar a mano cada warning y contarlos para llenar la tabla

Tabla 5. Hallazgos de calidad encontrados con herramientas de análisis.

Esta tabla se puede llenar con el uso del comando mix credo -a que muestra cuantos warnings encontró, así como los refactoring opportunities. Esto se tendría que hacer para cada proyecto.

Tabla 6. Líneas de acción propuestas y sus respectivos estimados.

Esta tabla en particular creo que se llena a partir de los hallazgos que se encontraron del proyecto y de las tablas anteriores así como de la capacidad que tienen los desarrolladores de Bunsan para estimar un tiempo específico para completar cada tarea y asignarle dicha prioridad a cada tarea.