Project templates and Cookiecutter

medium.com/worldsensing-techblog/project-templates-and-cookiecutter-6d8f99a06374

March 6, 2019

Héctor Canto

Jul 18, 2018

.

8 min read

Lately I've been taking several approaches to gather common code and help my colleagues follow the programming — and documenting — best practices we have set. So, I've been involved in analyzing code, making libraries, improving guides and other related tasks.

Use Cookiecutter to construct your new repositories better and faster.

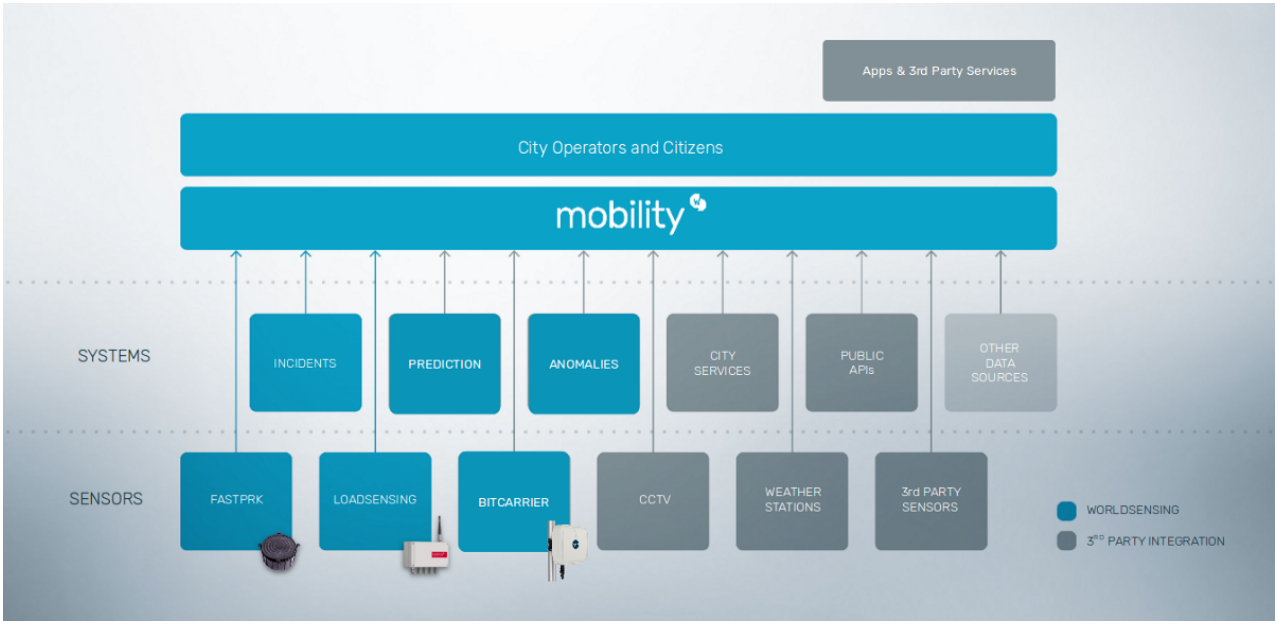
From this and previous experiences I'm writing a series of articles, starting with this one, regarding best practices and tools to help you comply with them.

One of the easier and most successful initiatives has been creating project templates — a.k.a. scaffolding, bootstrap, boilerplate or skeleton — with Cookiecutter.

Some context first

Starting a new repository in a project is always cumbersome, specially if you collaborate with others: you have to follow standards and agreements, there are some mandatory files, you have to place deploy artifacts somewhere, the requirements should be in a certain place et cetera.

One of the most recurrent activities at Worldsensing is integrating our products and platforms with partner and customer subsystems. To do it, we do not modify our core platform, we usually create middleware software that takes data from external subsystems and adapt it to our standards, or the other way around. We call this middleware *connectors*.



Worldsensing architecture overview.

Usually, they team I belong develop these connectors, but that is not always true: sometimes they are done by the partners and customers themselves, by a third party, or by another team in the company. Also, with every new project we reuse or modify existing connectors, or create new ones inspired in the old ones.

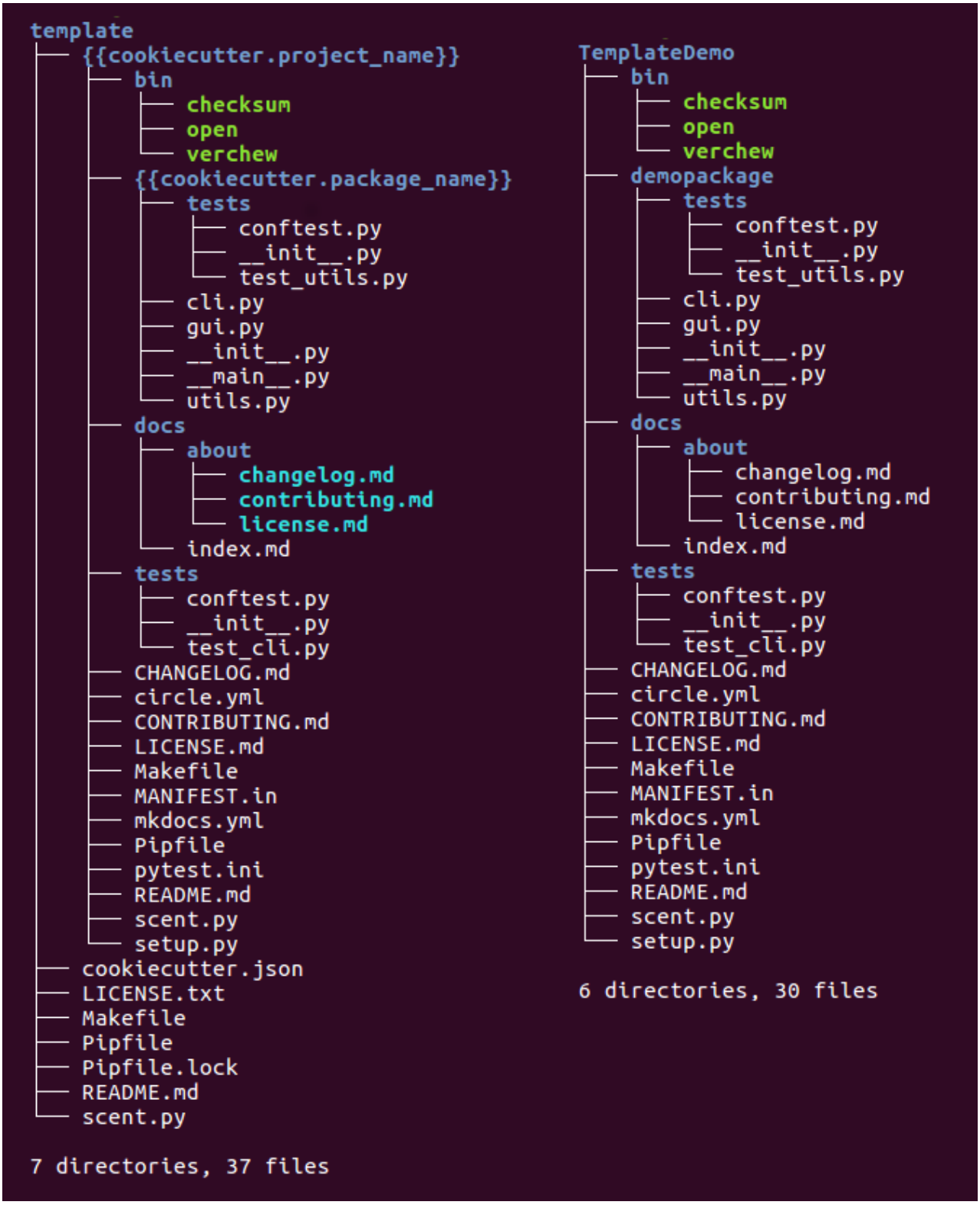
At the end we have many repositories that look alike but, while we try to follow the same standards and structures, that does not always work as it should. To help every party involved fulfill these expectations we created project templates for everyone to follow.

Using Cookiecutter and project templates has been key to ease the adoption of best practices and saved us a lot of time constructing new repositories.

What is Cookiecutter

Cookiecutter is a CLI tool (Command Line Interface) to create an application boilerplate from a template. It uses a templating system — Jinja2 — to replace or customize folder and file names, as well as file content.

We will see some examples later but the result is more or less what shows in the picture bellow:



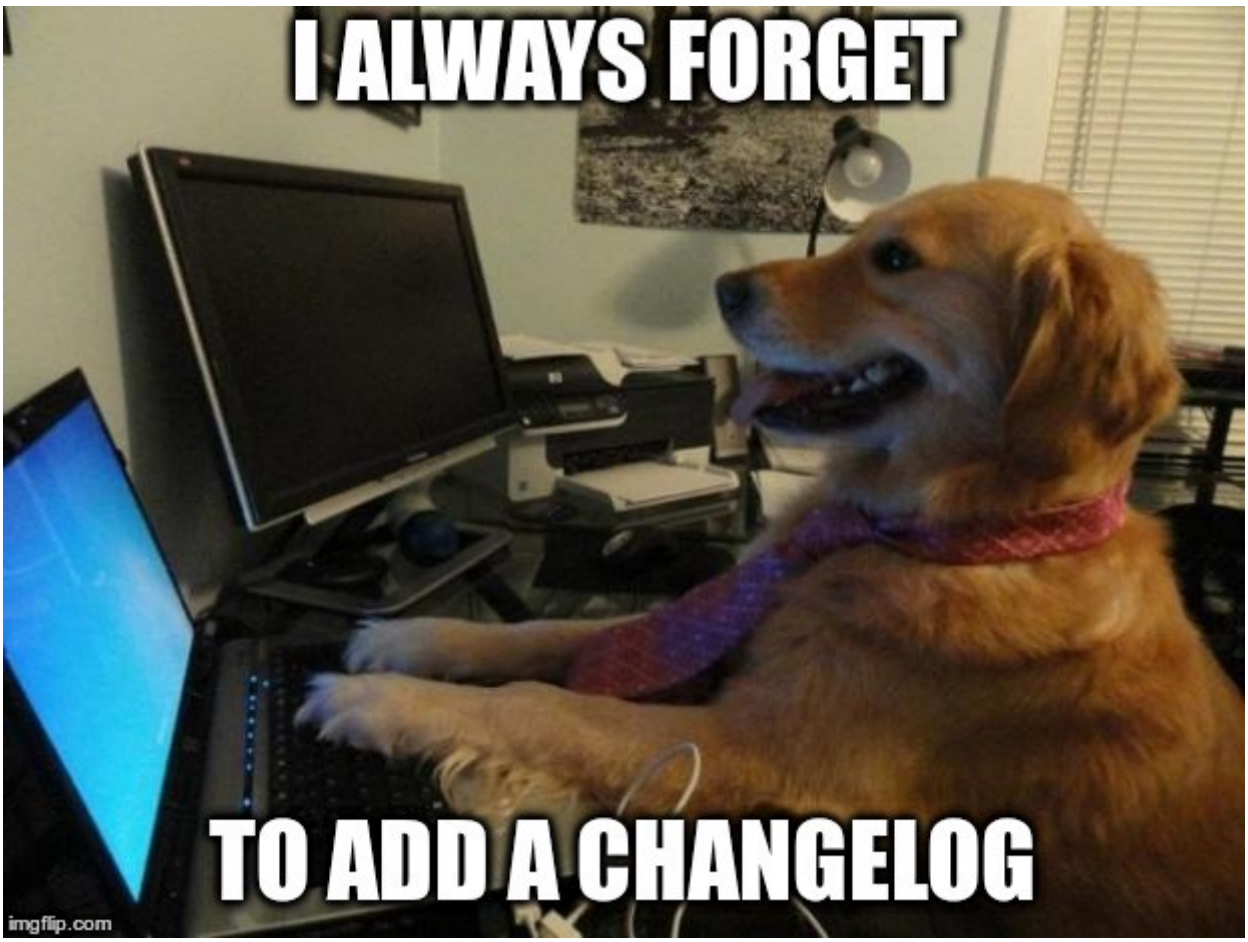
Left is the template, Right is the result after running Cookiecutter

While Cookiecutter is built in Python and the examples shown in this article are in Python too, it can be used with any other programming language.

Mind that, to make your own Cookiecutter template you need to learn some Jinja, and to implement hooks you do need Python.

Why do we use Cookiecutter

We use Cookiecutter to save time constructing a new repository, to avoid forgetting mandatory files like Readme or Changelog; and to lower the entry level to new collaborators — new team members, freelancers, partners.



Our new intern is a bit forgetful.

We also use it as a way to enforce standards: you can see it as an invitation to the developer to follow the established rules: make tests and check coverage , write documentation, follow a particular naming syntax... By giving them the structure and the boilerplate code, it makes it easier for developers to do things right.

In the case of connectors, which are rather simple programs, the code reuse is very high and most files are left almost unchanged from the generated ones, making it also a very powerful tool to save time increasing reusability.

How it is used

Cookiecutter can be used either with a Git repository, with a folder or even a Zip file. Also, if working with repositories, you can start your template from any branch. Some dummy example calls:

```
cookiecutter my_local_template/cookiecutter
https://github.com/some_account/template_py.gitcookiecutter
```

To try it out for real you can do the following with a repository available at Audrey R, Greenfeld's account, creator of Cookiecutter:

```
pip install – user cookiecuttercookiecutter
```

This example uses a template meant for Python packages to be published in Pypi (the Python Package Index) . If you execute it, you will be asked for a few values, for example:

```
full_name [Audrey Roy Greenfeld]: version [0.1.0]:use_pytest [n]:Select
command_line_interface:1 – Click2 – No command-line interfaceChoose from 1, 2 [1]:
```

You will either press enter for accepting the default value — within square brackets [] — or make your own choice — in bold in the example above. Options might be text, booleans (*y* or *n*) or a set of options. The prompt is created depending on the fields defined in the cookiecutter.json — we will see some details about it later.

Those two features, execute from a branch or folder and provide a prompt to resolve variable, make it easier to develop a new template. This way, you may start with a bare template repository , creating your desired structure and content using template variables instead of definitive names where you want to provide options. The final names will be resolved when executing Cookiecutter against the template.

You can see a live example in the following link:

Cookiecutter demo

A Cookiecutter demo using a public template from https://github.com/jacebrowning/template-python.git

asciinema.org

How a Cookiecutter template works

A Cookiecutter repository template looks something like this:

```
- cookiecutter.json – hooks | – pre_get_project.py | – post_get_project.py – [
README.md ] – [ CHANGELOG.md ] – {{cookiecutter.project_slug}} | –
```

Let's see each item in detail:

Cookiecutter.json is the key file in your template. It is a JSON with variables and choices, we will talk about it in the next section.

Hooks are python scripts with rules applicable before and after the repository generation. Personally I use the pre-hook to validate the choices you make and the post-hook to remove undesired files depending on those choices.

{{cookiecutter.project_slug}}/Here you will put your repository template with all the structure and content you want to replicate, placing template variables — between double curly brackets {{ }} — and Jinja2 ifs and loops.

README.md and **CHANGELOG.md** are not mandatory, but I really recommend you to have them to explain in detail the choices available in the Json file and to to keep track of the template evolution.

This Readme and Changelog will not be present in the generated repository, there should be another *Readme* and *Changelog* within <*your repository template*>.

You can begin your own either from a bare repository adding all things you want, or copy an existing repository and start replacing names and content for variables, or removing undesired elements.

Variables and choices in cookiecutter.json

To have names and content generated you define different variables in the *cookiecutter.json, so* you will get prompted for the values you want.

For each variable you set a default text value, a boolean option or a list of options. Some usual examples:

```
{  project_name: Project Sample  project_slug :{{
cookiecutter.project_name.lower().replace(‘ ‘,  ‘_’).replace(‘-’, ‘_’) }}”
python_version: [“2.7.15”, “3.6.5”]  some_lib_version: 1.2.1  initial_version:
[“0.1.0rc1”, “1.0.0rc1”]  use_pytest: true}
```

Input validity can be checked using pre-generation hooks. Decisions like removing or adding files can be made with post-generation hooks — for advanced usage beyond Jinja2 capabilities.

As you can see, you can either provide simple default values or use Jinja or Python commands to create dynamic default values.

How Cookiecutter templating system

As we said before Cookiecutter uses Jinja2, but in general you have to know two things about it:

Variable substitution in names and content:

```
{{cookiecutter.variable_name}}
```

Section selection with if statements:

```
{%- if cookiecutter.variable_name== ‘y’ %}{%- endif %}
```

There are other artifacts like for loops in Jinja2, but so far I did not go beyond variables and ifs. They should be more useful in other code bases like HTML.

Some templates you can already use

While researching this topic I've compiled a few Cookiecutter templates ready to use or fork. Here's the list:

How to develop your own template

To develop a project template, so far I've used two strategies:

- Copy an existing repository and convert it to a template.
- For a new project, start with the template instead of the specific project.

Starting from an existing repository is the easy way to go, but it is good to start from scratch sometimes, to avoid taking old assumptions based on the existing code.

With any of the two, take these tips into account to make a good template:

- Replace all the specific names for Jinja variables in folder and file names, and content.
- Keep reusable sections or wrap them with templates
- Write a README explaining the meaning of the options in cookiecutter.json
- Add some comments to help you and your colleagues understand what was the intention of some obscure sections. They can remove them later on.
- Use some code word to help them to identify comments meant for deletion like NOTE or TEMPLATE — in the same fashion as TODO.
- On empty variables, functions or methods; add some exception mechanism to prevent unfilled sections. You can use Python's Java's NotImplementedException, JavaScript new Error("Not implemented") and so on.

Template and repository maintenance

And for maintaining the template — and the projects based on it — I recommend:

- Have the template open in an IDE (integrated development code editor) window as you advance in the project to add new common code you haven't considered at the beginning.
- When coming back to an old repository, refactor it to match the current state of the template; thus reducing the difference between old projects and new ones and matching them to your current (team) standards.
- You do not necessarily have to refactor every old repository based in a given template when a template changes. Once a project or repository is started the template has already made its function.

One important warning, do no put all your common code into the template. There is a point when a big chunk of reusable code should become a library

Conclusions

We are very happy with the results of using Cookiecutter and repository templates with connectors and we are extending its use to other types of component. Those types are more complex and they have less code and structure in common between them, but the effort pays off anyway.

Now our repositories start in a good shape with all the elements it should and we reduced the time it took to start a new repository.

We also have started using the templates with third parties as from time to time they ignored some of the instructions we shared with them. By stating them in the template, the deviation was lower.

This freelance works well but does not like to make tests.

There is also an interesting point that it was easy for us to remind external collaborators they forgot to do something as expected. If they deleted some elements of the template that they should have not — tests folder, Changelog … — , it was easier for us to complain and for them to recognize their (naughty) intentions.

I hope this article helps you and that you find Cookiecutter as useful as it was for us.