

WRITING EXTENSIBLE AND MAINTAINABLE CODE IN AN ACADEMIC SETTING

Matthew Giammar

PRESENTATION OVERVIEW

1. Why make code flexible and extensible?
2. Good vs bad code, with examples!
3. Shared projects with git and GitHub.
4. When and when not to spend time improving code.

PRESENTATION OVERVIEW

- 1. Why make code flexible and extensible?**
2. Good vs bad code, with examples!
3. Shared projects with git and GitHub.
4. When and when not to spend time improving code.

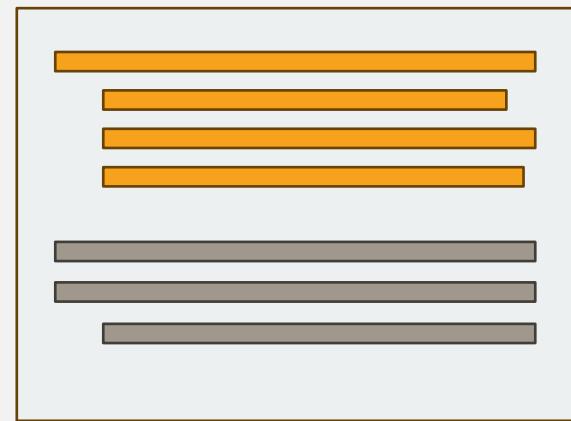
FLEXIBILITY AND EXTENSIBILITY

Flexible code can easily be swapped out with equivalent pieces in the future

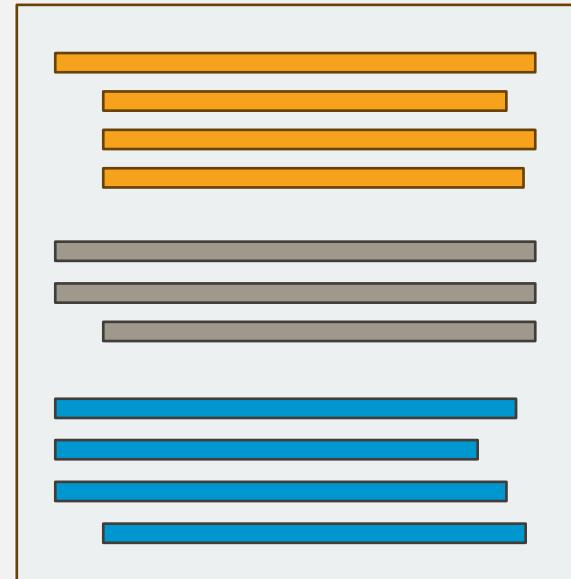
Extensibility means new functionality can be added on with minimal effort

OOP and single responsibility principals help make code flexible and extensible

Flexibility



Extensibility



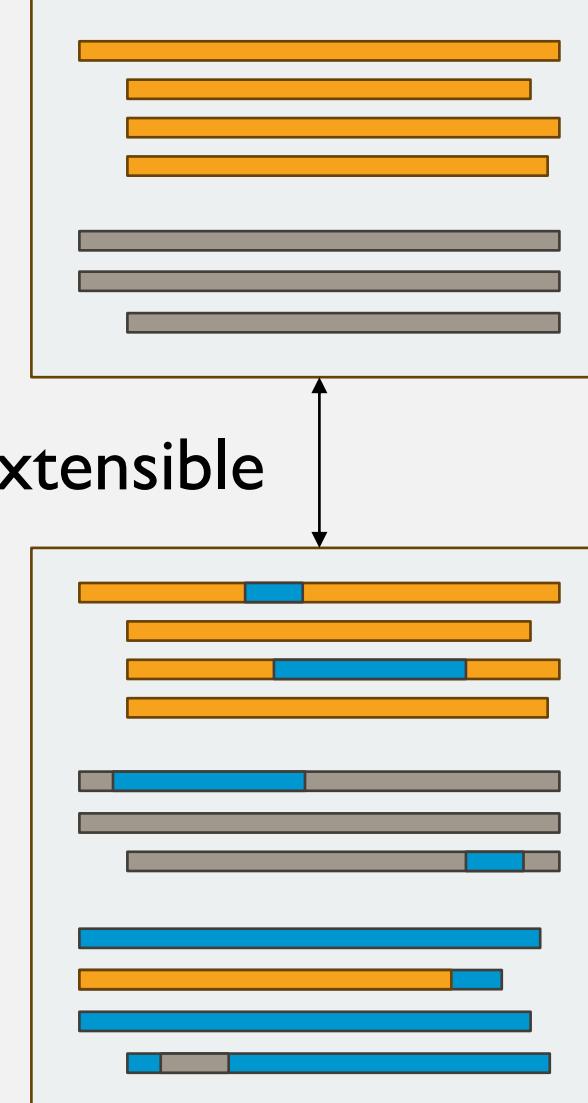
FLEXIBILITY AND EXTENSIBILITY

Flexible code can easily be swapped out with equivalent pieces in the future

Extensibility means new functionality can be added on with minimal effort

OOP and single responsibility principals help make code flexible and extensible

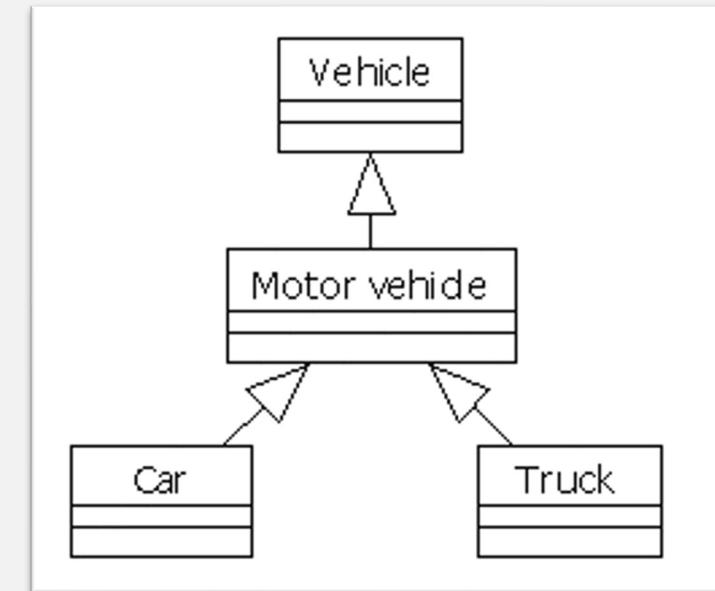
Not extensible



SINGLE RESPONSIBILITY PRINCIPLE



- Each class and method should exist for one single purpose; nothing is overly complex
- Clear information flow between objects in larger programs
- Disentanglement makes code flexible and extensible



CLEAR & FLEXIBLE VS MESSY

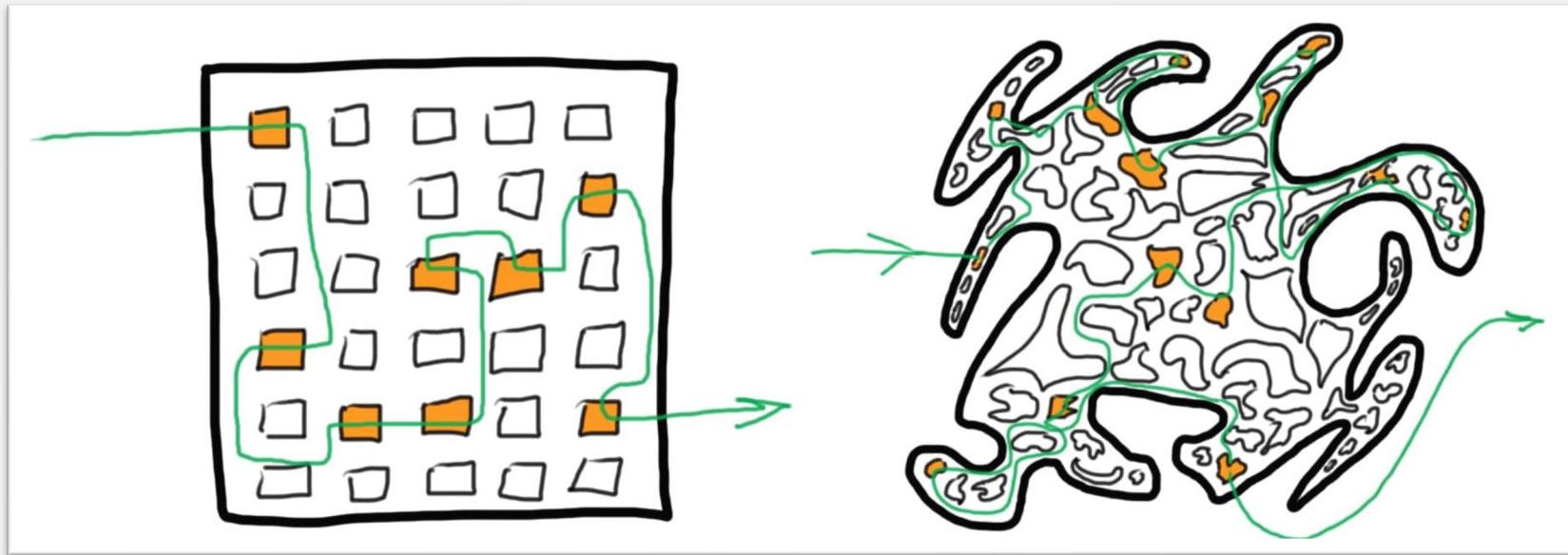
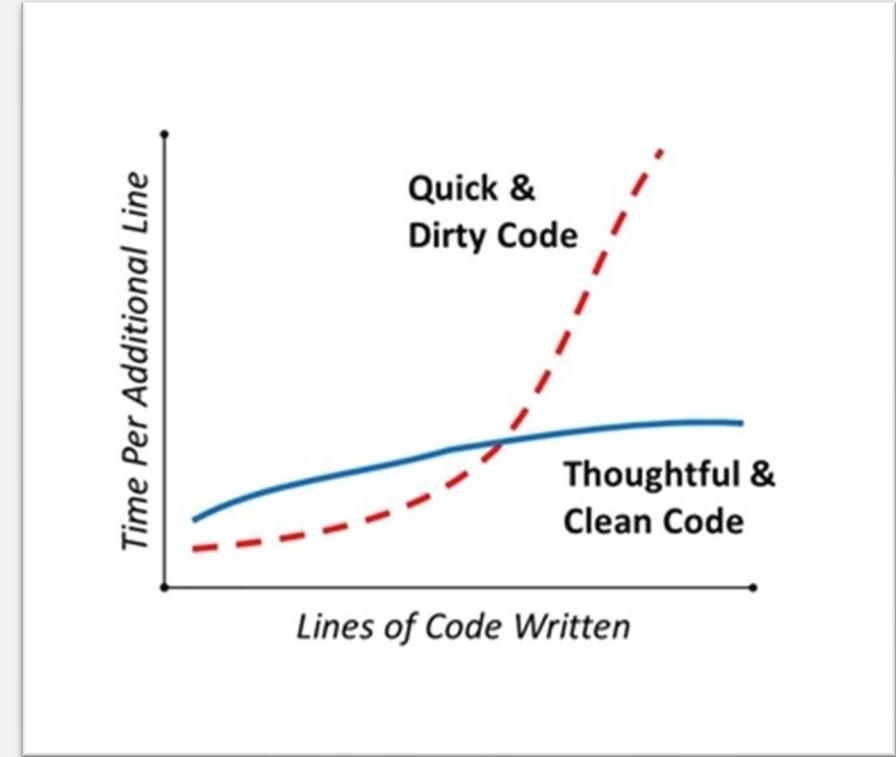


Image credit: Daniel Harper

BENEFITS OF FLEXIBLE CODE

- Update workflows as new methods arise
- Single-responsibility principle makes understanding code easier
- Increases longevity of codebase



PRESENTATION OVERVIEW

1. Why make code flexible and extensible?
2. **Good vs bad code, with examples!**
3. VS code + extensions are your friend.
4. Shared projects with git and GitHub.
5. When and when not to spend time improving code.

GOOD VS BAD CODE

Good

- Easily understandable with clear variable and function names
- Complex portions well commented
- Flexible, extensible, and maintainable
- Associated documentation and example use cases

Bad

- Overly short and unclear variable names
- Difficult to understand how pieces of code work together
- Sparse and uninformative comments
- Highly intertwined workflows and unclear documentation

```
def __molAnaloguesLibEnumeration(self, reagent, partner, numberOfRowsInSectionBBalreadyReacted,
                                  SynthonsetsUsed, reactionToUse, usedReactions = None, firstLaunch=True):
    if not usedReactions:
        usedReactions = []
    allProducts = set()
    products = self.__reactions[reactionToUse].RunReactants((reagent, partner))
    if not products:
        products = self.__reactions[reactionToUse].RunReactants((partner, reagent))
    if products:
        for prodSet in products:
            for prod in prodSet:
                prodSMILES = Chem.MolToSmiles(prod, canonical=True)
                functionality = prodSMILES.count(":")
                prod.UpdatePropertyCache()
                Chem.GetSymmSSSR(prod)
                prod.GetRingInfo().NumRings()
                newUsedReactions = usedReactions.copy()
                newUsedReactions.append(reactionToUse)
                if functionality:
                    for rid in range(len(self.__reactions)):
                        if rid not in newUsedReactions:
                            reaction = self.__reactions[rid]
                            for SynthonsetPartner in self.__Synthons:
                                if SynthonsetPartner not in SynthonsetsUsed and self.__checkBB_reactionCombination(prod,
                                                                                                         self.__Synthons[SynthonsetPartner][0], reaction):
                                    for secondBB in self.__Synthons[SynthonsetPartner]:
                                        newSynthonsetsUsed = set()
                                        for i in SynthonsetsUsed:
                                            newSynthonsetsUsed.add(i)
                                        newSynthonsetsUsed.add(SynthonsetPartner)
```

Synthl: A New Open-Source Tool for Synthon-Based Library Design

Yuliana Zabolotna, Dmitriy M. Volochnyuk, Sergey V. Ryabukhin, Kostiantyn Gavrylenko, Dragos Horvath, Olga Klimchuk, Oleksandr Oksiuta, Gilles Marcou, and Alexandre Varnek*



Cite This: *J. Chem. Inf. Model.* 2022, 62, 2151–2163



Read Online

```
Chem.GetSymmSSSR(prod)
prod.GetRingInfo().NumRings()
newUsedReactions = usedReactions.copy()
newUsedReactions.append(reactionToUse)
if functionality:
    for rid in range(len(self.__reactions)):
        if rid not in newUsedReactions:
            reaction = self.__reactions[rid]
            for SynthonsetPartner in self.__Synthons:
                if SynthonsetPartner not in SynthonsetsUsed and self.__checkBB_reactionCombination(prod,
                                                                                                     self.__Synthons[SynthonsetPartner][0], reaction):
                    for secondBB in self.__Synthons[SynthonsetPartner]:
                        newSynthonsetsUsed = set()
                        for i in SynthonsetsUsed:
                            newSynthonsetsUsed.add(i)
                        newSynthonsetsUsed.add(SynthonsetPartner)
```

Synthl: A New Open-Source Tool for Synthon-Based Library Design

Yuliana Zabolotna, Dmitriy M. Volochnyuk, Sergey V. Ryabukhin, Kostiantyn Gavrylenko, Dragos Horvath, Olga Klimchuk, Oleksandr Oksiuta, Gilles Marcou, and Alexandre Varnek*



Cite This: *J. Chem. Inf. Model.* 2022, 6

```
Chem.GetSymmSSSR(prod)
prod.GetRingInfo()
newUsedReactions =
newUsedReactions.append(reaction)
if functionality:
    for rid in range(len(self.__reactions)):
        if rid not in newUsedReactions:
            reaction = self.__reactions[rid]
            for SynthonsetPartner in self.__Synthons:
                if SynthonsetPartner not in SynthonsetsUsed and self.__checkBB_reactionCombination(prod,
                                                                                                     self.__Synthons[SynthonsetPartner][0], reaction):
                    for secondBB in self.__Synthons[SynthonsetPartner]:
                        newSynthonsetsUsed = set()
                        for i in SynthonsetsUsed:
                            newSynthonsetsUsed.add(i)
                        newSynthonsetsUsed.add(SynthonsetPartner)
```

Bad

```
def fileConfig(fname, defaults=None, disable_existing_loggers=True, encoding=None):  
    """
```

```
    if isinstance(fname, str):  
        if not os.path.exists(fname):  
            raise FileNotFoundError(f"{fname} doesn't exist")  
        elif not os.path.getsize(fname):  
            raise RuntimeError(f'{fname} is an empty file')  
  
    if isinstance(fname, configparser.RawConfigParser):  
        cp = fname  
    else:  
        try:  
            cp = configparser.ConfigParser(defaults)  
            if hasattr(fname, 'readline'):  
                cp.read_file(fname)  
            else:  
                encoding = io.text_encoding(encoding)  
                cp.read(fname, encoding=encoding)  
        except configparser.ParsingError as e:  
            raise RuntimeError(f'{fname} is invalid: {e}')  
  
    formatters = _create_formatters(cp)  
  
    # critical section  
    with logging._lock:  
        _clearExistingHandlers()  
  
        # Handlers add themselves to logging._handlers  
        handlers = _install_handlers(cp, formatters)  
        _install_loggers(cp, handlers, disable_existing_loggers)
```

OFFICIAL PYTHON SOURCE CODE

```
developer provides a mechanism to present the choices and load the chosen
configuration).
"""
import configparser

if isinstance(fname, str):
    if not os.path.exists(fname):
        raise FileNotFoundError(f"{fname} doesn't exist")
    elif not os.path.getsize(fname):
        raise RuntimeError(f'{fname} is an empty file')

if isinstance(fname, configparser.RawConfigParser):
    cp = fname
else:
    try:
        cp = configparser.ConfigParser(defaults)
        if hasattr(fname, 'readline'):
            cp.read_file(fname)
        else:
            encoding = io.text_encoding(encoding)
            cp.read(fname, encoding=encoding)
    except configparser.ParsingError as e:
        raise RuntimeError(f'{fname} is invalid: {e}')

formatters = _create_formatters(cp)

# critical section
with logging._lock:
    _clearExistingHandlers()

    # Handlers add themselves to logging._handlers
    handlers = _install_handlers(cp, formatters)
    _install_loggers(cp, handlers, disable_existing_loggers)
```

OFFICIAL PYTHON SOURCE CODE

```
developer provides a mechanism to present the choices and load the chosen
configuration).
"""
import configparser

if isinstance(fname, str):
    if not os.path.exists(fname):
        raise FileNotFoundError(f"{fname} doesn't exist")
    elif not os.path.getsize(fname):
        ...

if is...:
    c...
else:
    t...
    else:
        encoding = io.text_encoding(encoding)
        cp.read(fname, encoding=encoding)
    except configparser.ParsingError as e:
        raise RuntimeError(f'{fname} is invalid: {e}')

formatters = _create_formatters(cp)

# critical section
with logging._lock:
    _clearExistingHandlers()

    # Handlers add themselves to logging._handlers
    handlers = _install_handlers(cp, formatters)
    _install_loggers(cp, handlers, disable_existing_loggers)
```

Good

```
def make_binary_mat(library_folder, ref_model):
    # returns 3 binary matrices containing info about whether reaction/metabolite/gene[i]
    # from parent model has been added in each "contextualized" model

    reactions_matrix = pd.DataFrame(index=[r.id for r in ref_model.reactions])
    metabolite_matrix = pd.DataFrame(index=[m.id for m in ref_model.metabolites])
    gene_matrix = pd.DataFrame(index=[g.id for g in ref_model.genes])

    for filename in sorted(os.listdir(library_folder)):
        model = cobra.io.read_sbml_model(library_folder+filename)
        rxns = []
        mets = []
        genes = []

        label = str(filename).split('.')[0]

        for r in ref_model.reactions:
            if r in model.reactions:
                rxns.append(1)
            else:
                rxns.append(0)

        for m in ref_model.metabolites:
            if m in model.metabolites:
                mets.append(1)
            else:
                mets.append(0)

        for g in ref_model.genes:
            if g in model.genes:
                genes.append(1)
            else:
                genes.append(0)

        reactions_matrix[label] = pd.Series(rxns).values
        metabolite_matrix[label] = pd.Series(mets).values
        gene_matrix[label] = pd.Series(genes).values

    return reactions_matrix, metabolite_matrix, gene_matrix
```

Article

A Distance-Based Framework for the Characterization of Metabolic Heterogeneity in Large Sets of Genome-Scale Metabolic Models

Andrea Cabbia,^{1,3,*} Peter A.J. Hilbers,¹ and Natal A.W. van Riel^{1,2}

¹Computational Biology, Eindhoven University of Technology, Groene Loper 5, 5612 AE Eindhoven, the Netherlands

²Amsterdam University Medical Centers, University of Amsterdam, Meibergdreef 9, 1105 AZ Amsterdam, the Netherlands

³Lead Contact

*Correspondence: a.cabbia@tue.nl

<https://doi.org/10.1016/j.patter.2020.100080>

```
for m in ref_model.metabolites:
    if m in model.metabolites:
        mets.append(1)
    else:
        mets.append(0)

for g in ref_model.genes:
    if g in model.genes:
        genes.append(1)
    else:
        genes.append(0)

reactions_matrix[label] = pd.Series(rxns).values
metabolite_matrix[label] = pd.Series(mets).values
gene_matrix[label] = pd.Series(genes).values

return reactions_matrix, metabolite_matrix, gene_matrix
```

Article

A Distance-Based Framework for the Characterization of Metabolic Heterogeneity in Large Sets of Genome-Scale Metabolic Models

Andrea Cabbia,^{1,3,*} Peter A.J. Hilbers,¹ and Natal A.W. van Riel^{1,2}

¹Computational Biology, Eindhoven University of Technology, Eindhoven, the Netherlands

²Amsterdam University Medical Center, Amsterdam, the Netherlands

³Lead Contact

*Correspondence: a.cabbia@tue.nl

<https://doi.org/10.1016/j.patter.2023.109302>

Good

```
for m in models:
    if m in ref_model.metabolites:
        mets.append(1)
    else:
        mets.append(0)

    for g in ref_model.genes:
        if g in model.genes:
            genes.append(1)
        else:
            genes.append(0)

    reactions_matrix[label] = pd.Series(rxns).values
    metabolite_matrix[label] = pd.Series(mets).values
    gene_matrix[label] = pd.Series(genes).values

return reactions_matrix, metabolite_matrix, gene_matrix
```

```
g = 9.81

def v(iv, t):
    v_f = iv + (g * t)
    return v_f

def s(ip, iv, t):
    s_f = ip + (iv * t) + (0.5 * g * t ** 2)
    return s_f

if __name__ == "__main__":
    p = 0
    iv = 10
    t = 2

    vf = v(iv, t)
    print("Final velocity:", vf)

    sf = s(p, iv, t)
    print("Final position:", sf)
```

Chat-GPT

Prompt: Write some poor Python code for running a physics simulation

```
def v(iv, t):
    vf = iv + (g * t)
    return vf

def s(ip, iv, t):
    sf = ip + (iv * t) + (0.5 * g * t ** 2)
    return sf

if __name__ == "__main__":
    p = 0
    iv = 10
    t = 2

    vf = v(iv, t)
    print("Final velocity:", vf)

    sf = s(p, iv, t)
    print("Final position:", sf)
```

Chat-GPT

Prompt: Write some poor Python code for running a physics simulation

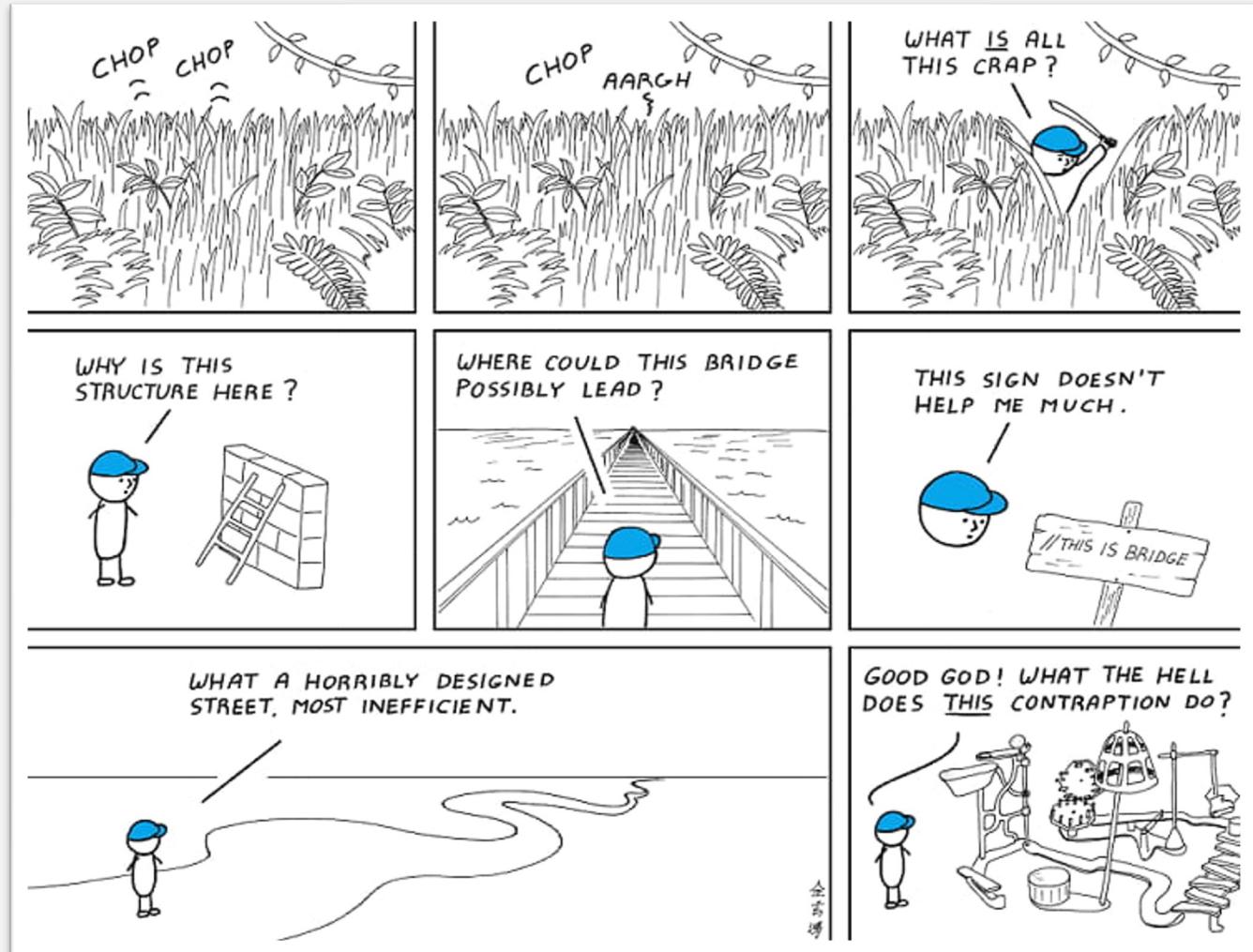
```
def v(iv, t):
    vf = iv + (g * t)
    return vf

def s(ip, iv, t):
    sf
    ret
    if __na
        p =
        iv
        t = 2

    vf = v(iv, t)
    print("Final velocity:", vf)

    sf = s(p, iv, t)
    print("Final position:", sf)
```

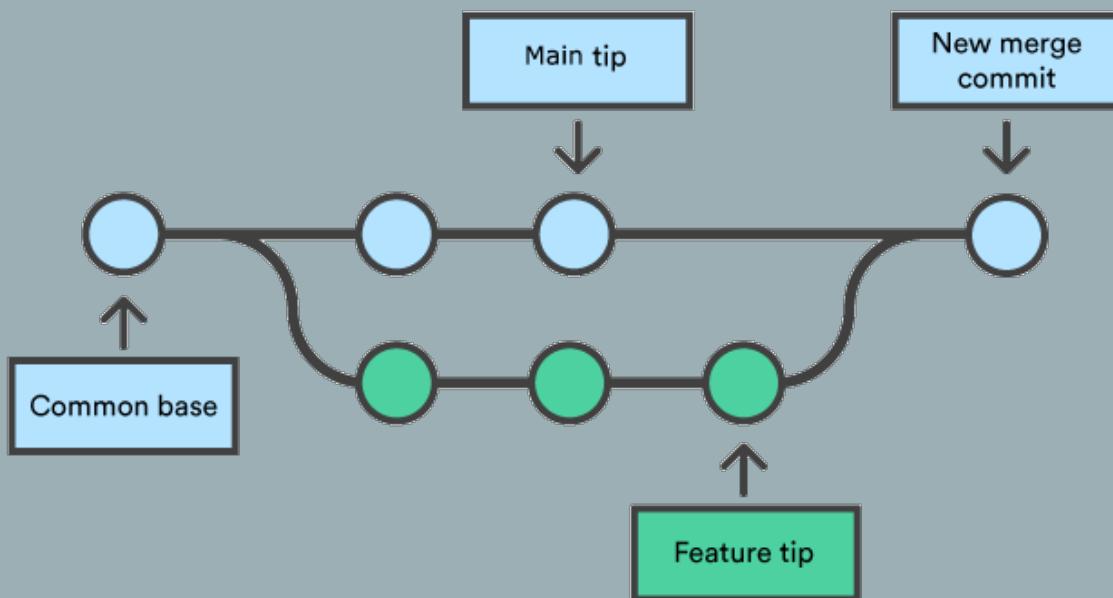
Bad?



PRESENTATION OVERVIEW

1. Why make code flexible and extensible?
2. Good vs bad code, with examples!
3. **Shared projects with git and GitHub.**
4. When and when not to spend time improving code.

GIT



- Git is a popular version control software package widely used in software development fields
- Tracks file changes between commits
- Multiple users can contribute to one main set of source code
- Versioning allows backtracking if you make a mistake or break something



HOW TO USE GIT

- Make or download a git repository (`git init / git clone <url>`)
- Change and add files to staging (`git add <file>`)
- Add new node to version history by making a commit (`git commit`)



HOW TO USE GIT

- Make or download a git repository (`git init / git clone <url>`)
- Change and add files to staging (`git add <file>`)
- Add new node to version history by making a commit (`git commit`)
- **Use the VS Code extension! Great official tutorial online**



- Online platform for storing and managing code (git repositories)
- Storing code online increases longevity and shareability
- Can create snapshots of code at certain points in time
- Easily store, share, and update code between multiple users



GitHub and Visual Studio Code

Bring your workflows closer to your code. Learn how Visual Studio Code and GitHub are better together.

[Download Visual Studio Code](#)

Free for Windows, macOS, and Linux

Get started with GitHub in Visual Studio Code

Check out our [20-minute tutorial](#) to get the best of GitHub in the editor.

<https://vscode.github.com>



Ctrl	+	Z

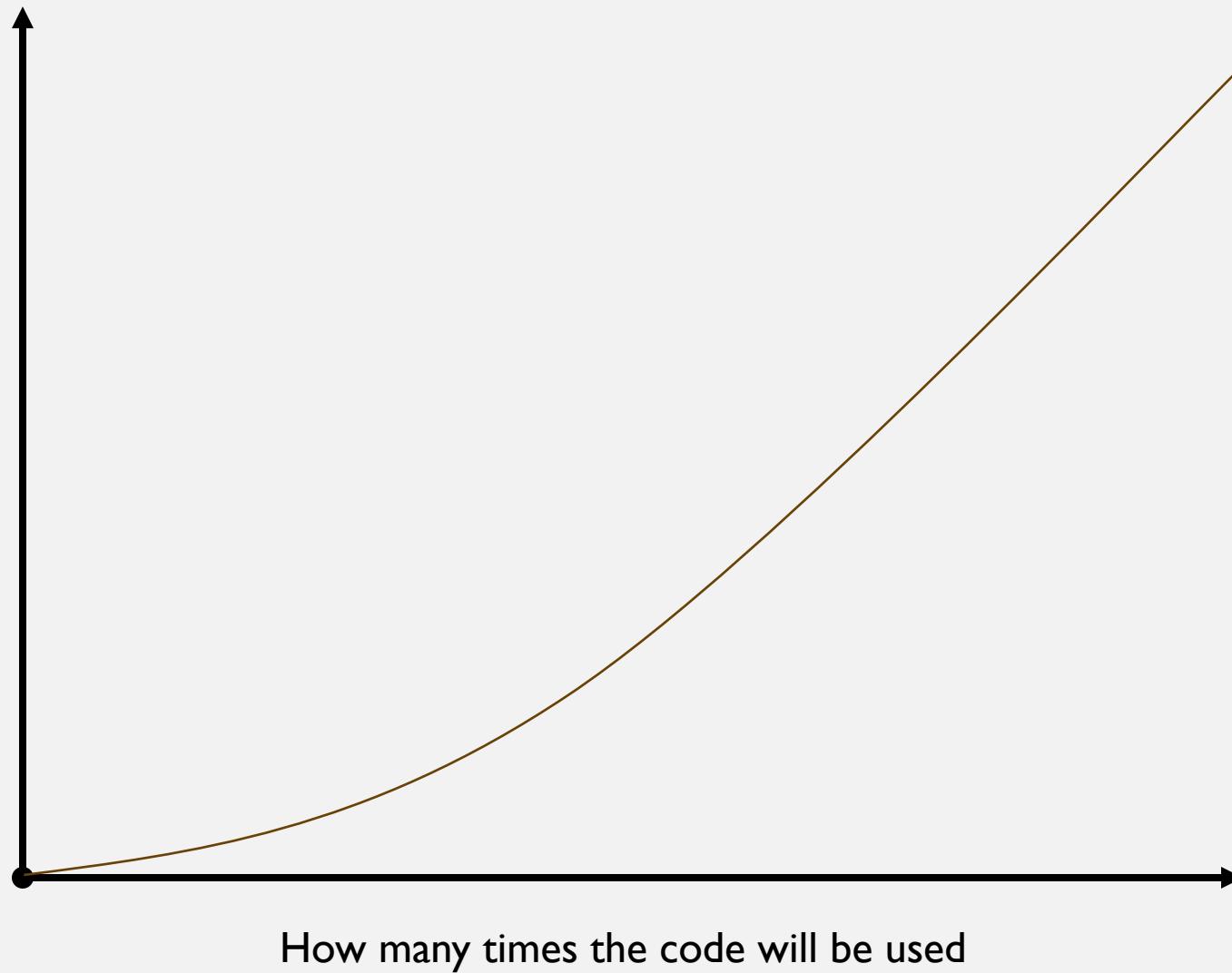
PRESENTATION OVERVIEW

1. Why make code flexible and extensible?
2. Good vs bad code, with examples!
3. Shared projects with git and GitHub.
4. **When and when not to spend time improving code.**

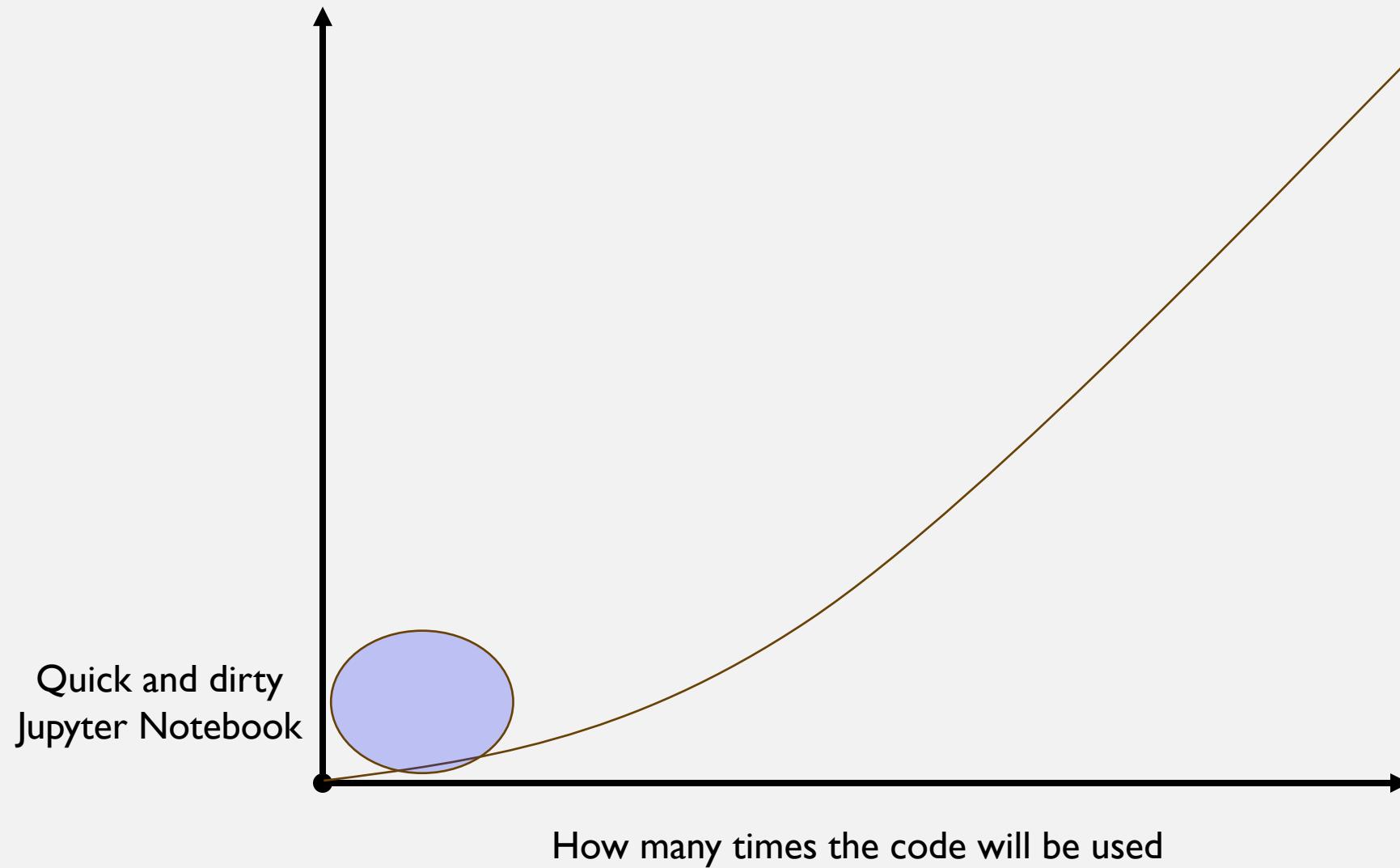
ACADEMIA ISN'T A SOFTWARE COMPANY

- Research fields evolve fast; code becomes outdated quickly.
- Want to check and show that methodology works, no time to clean up and prettify codebase.
- Disparate pieces in some pipeline may not work together easily; lack of common standards make code inflexible.

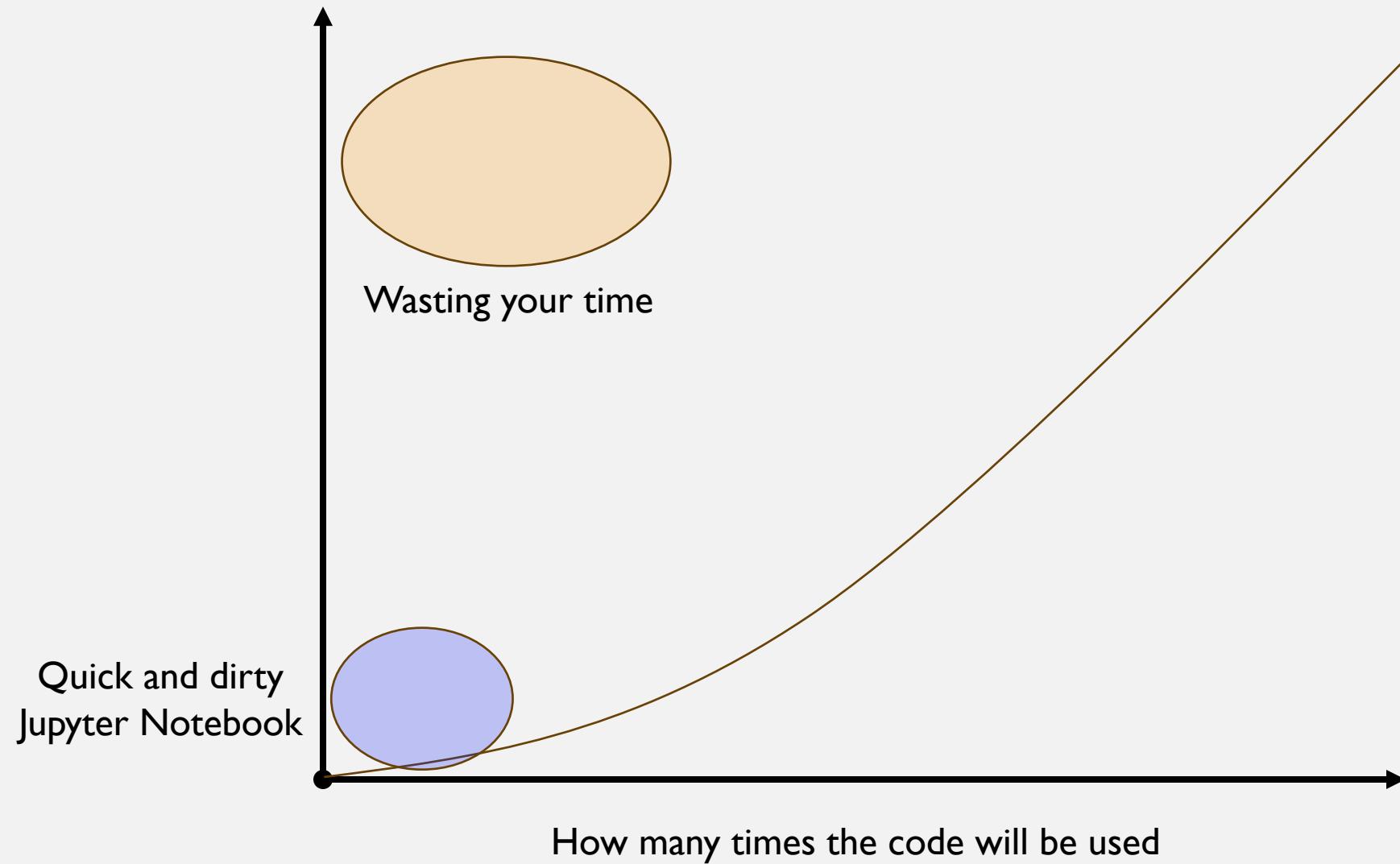
How “good” the code should be



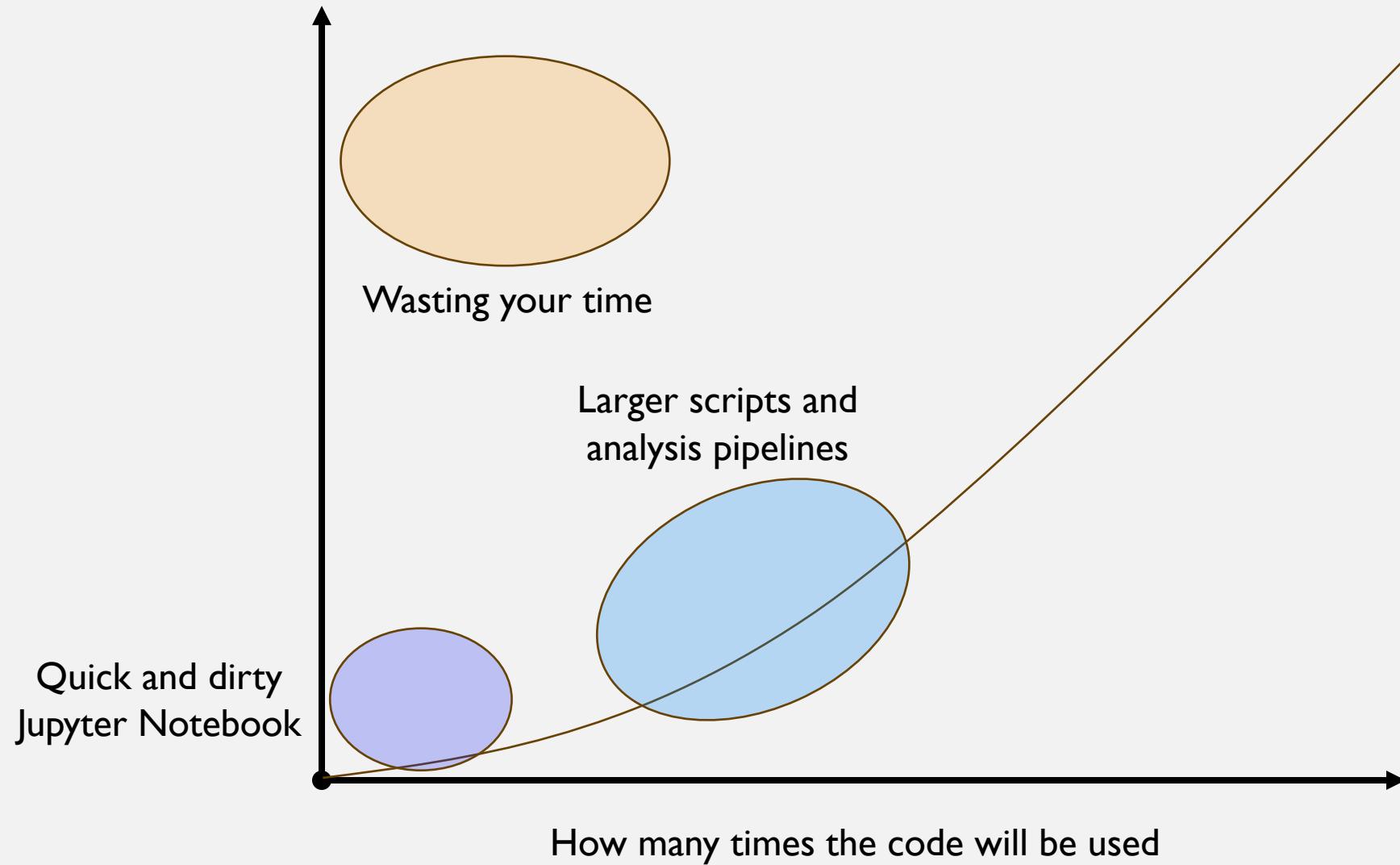
How “good” the code should be



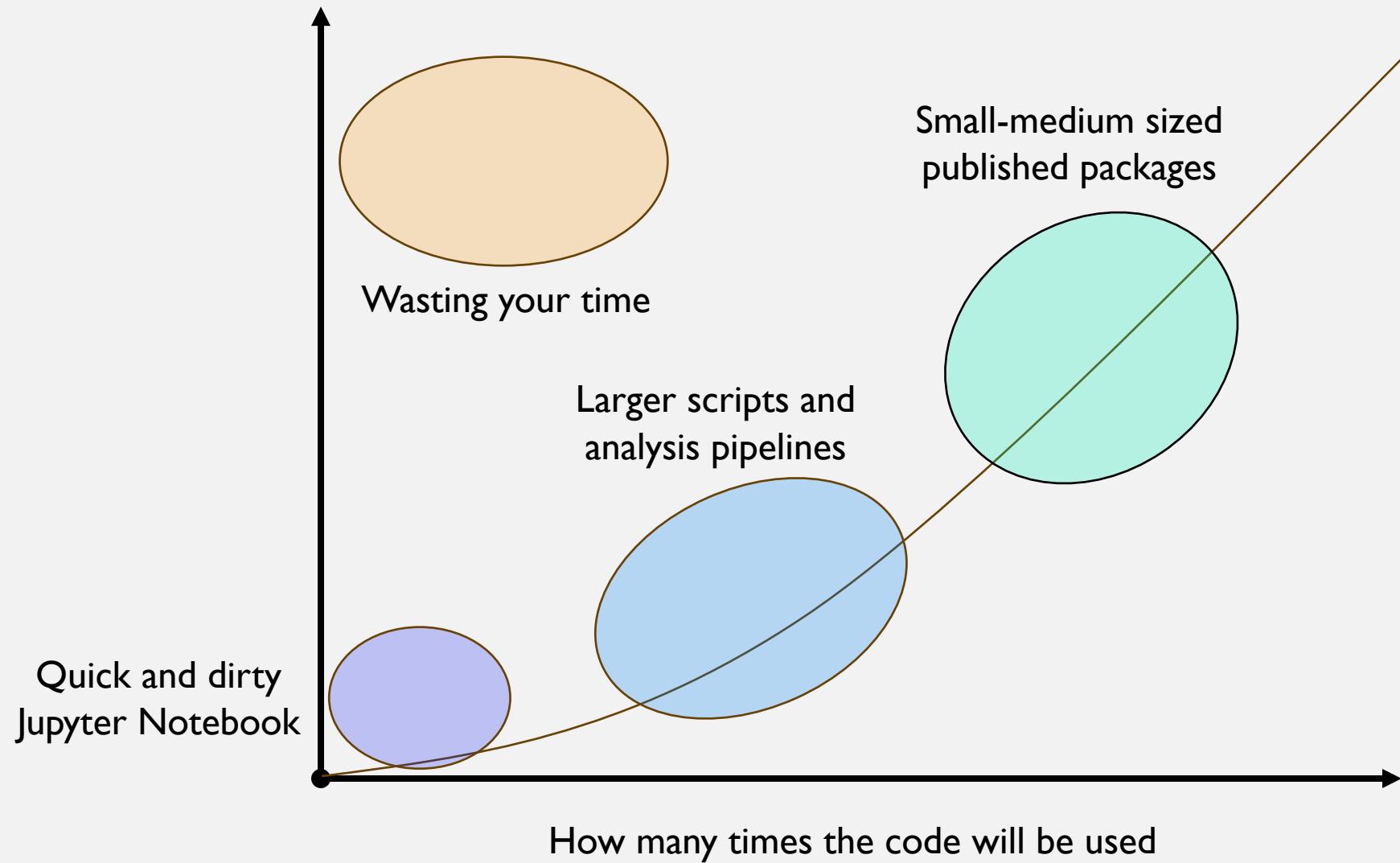
How “good” the code should be



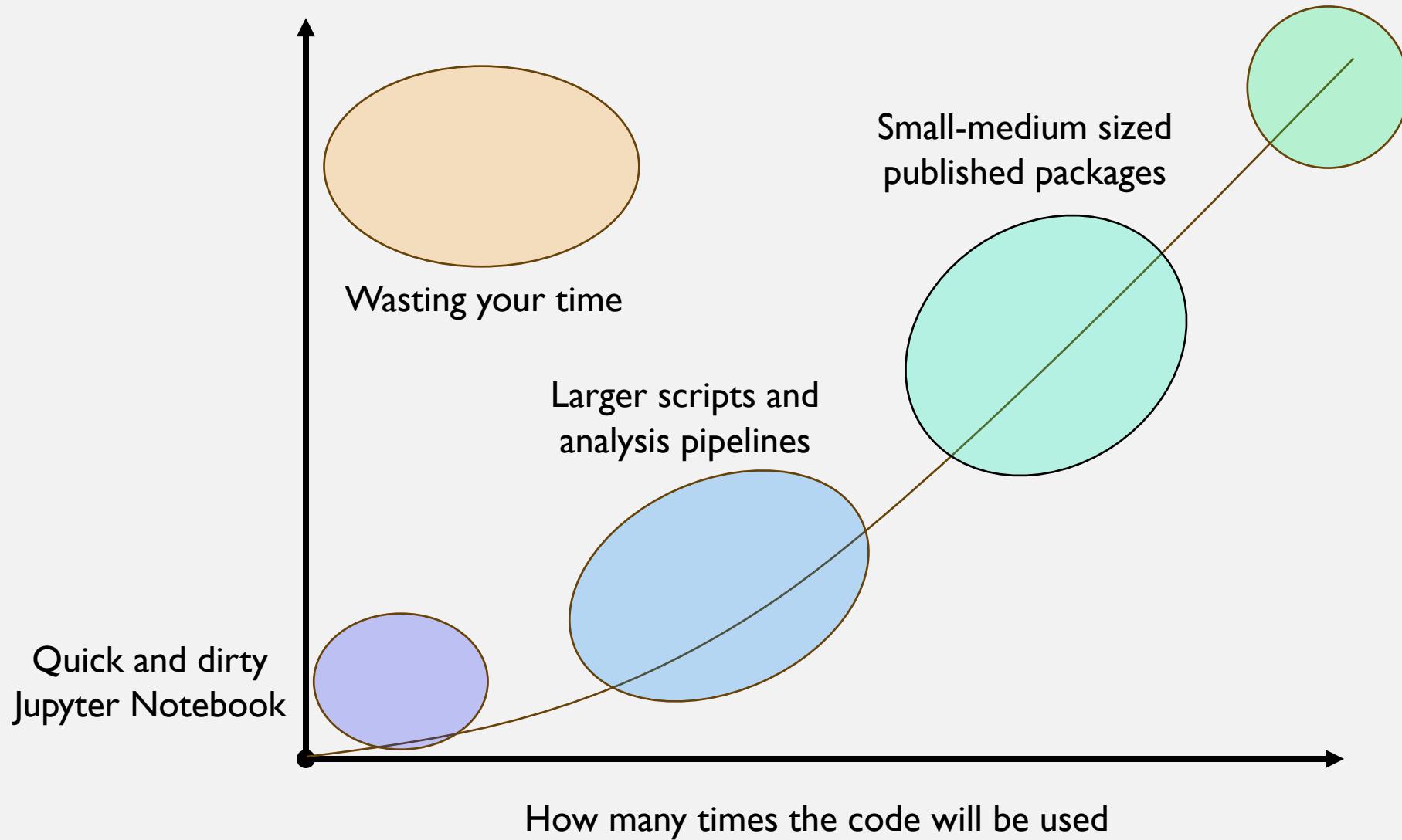
How “good” the code should be



How “good” the code should be

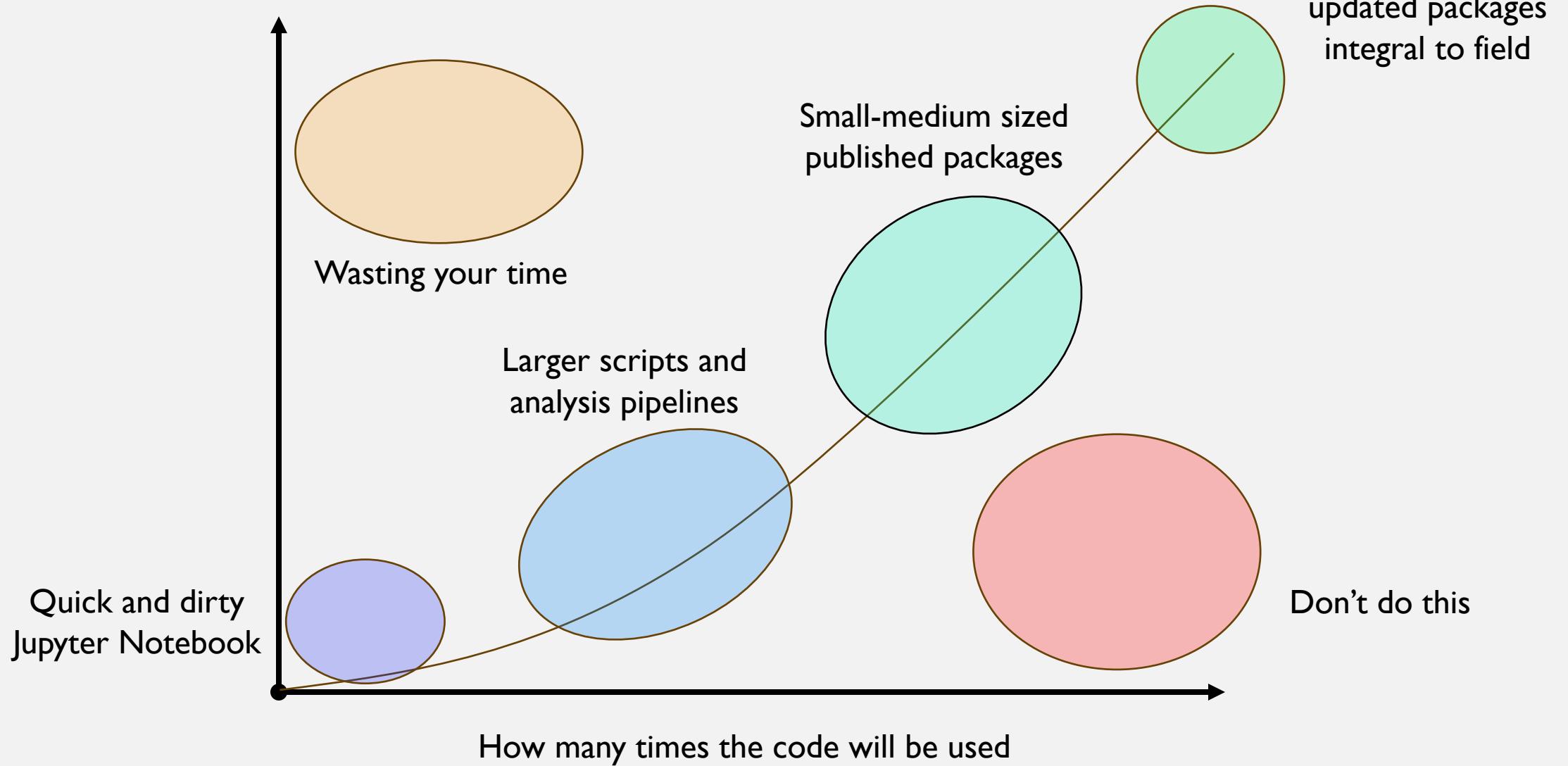


How “good” the code should be



Large continuously updated packages integral to field

How “good” the code should be



WHY MAKE CODE NICE



StackOverflow

- Spending time on good base code means easier to implement new code for interesting research directions
- Swap out or add in new code chunks as methods are developed
- Sharing and collaborating on code much easier if written in a readable manner
- Revisiting old code is much easier after weeks/months away

KEY TAKEAWAYS

1. Write readable code with comments!
2. Single-responsibility and OOP makes code flexible
3. Code editors (e.g. VS Code) come with helpful extensions which can make your programming life easier
4. Git and GitHub are invaluable tools for tracking and sharing code
 - Talk to your lab about making an organization account

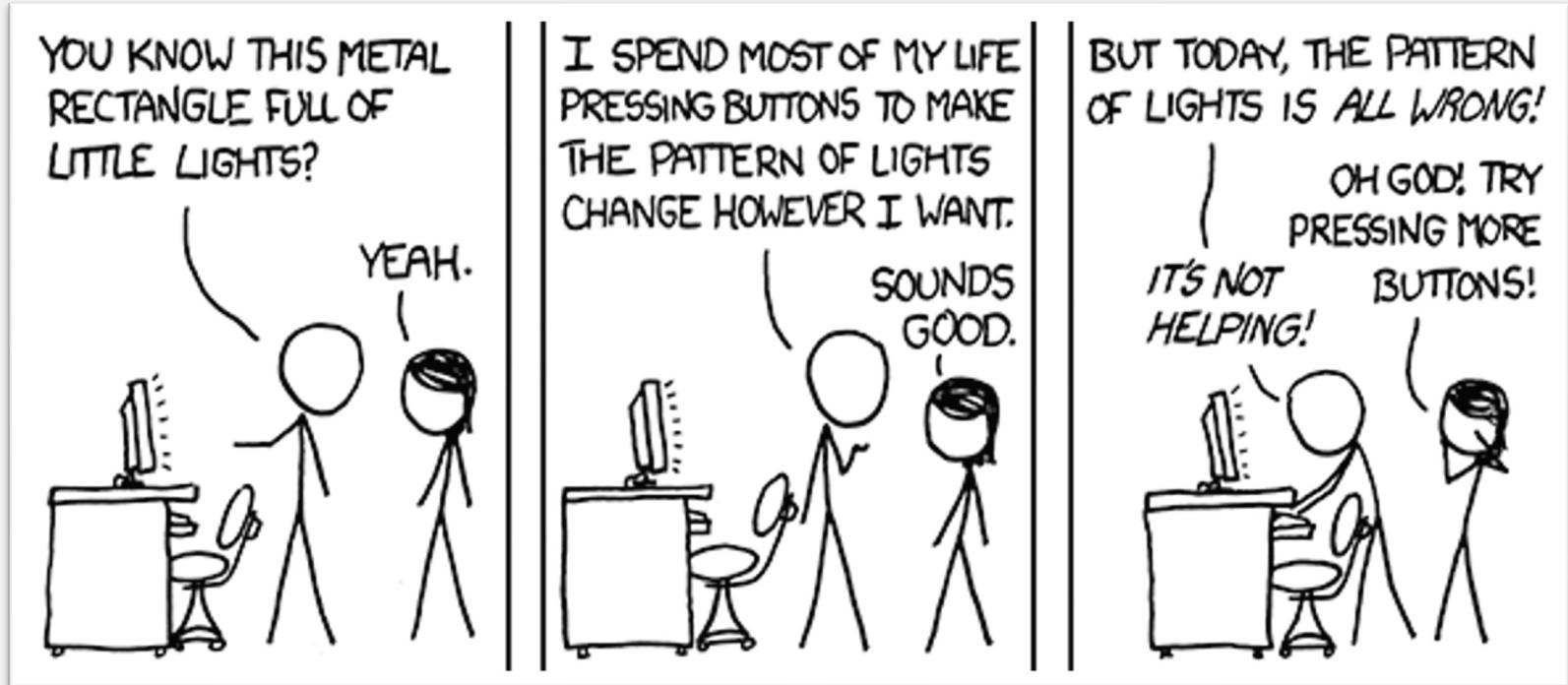


Image credit: xkcd

THANKS!