

02. Software Design - Requirements

- This topic discusses how we can use software design and OOP to solve problems.
- We have clients who request us to build software: the clients can be your boss, users, or even ourselves.
- We design and build high-quality software based on the request.

Clients request us to build software.

Mr. Star's Email

- A business owner, Mr. Star, emails us.

Dear Python programmer,

My name is Mr. Star and I own Star Station, a full service gasoline station in Deer Creek, a small town in the middle of the United States.

The station has 7 employees and I use an excel sheet to keep track of their names and salaries. But we want to improve our administration, and I need some fancy business software. Could you build this for us?

The first version should just replace our Excel sheet and show a list of the employees with their salaries.

My nephew (who wants to be a programmer) started with a small script that shows all the employee names. Perhaps you can use it.

Requests

- The station has seven employees.
- He needs to keep track of names and salaries.

| NAME | SALARY |
|----------|--------|
| Vera | 2000 |
| Chuck | 1800 |
| Samantha | 1800 |
| Roberto | 2100 |
| Joe | 2000 |
| Dave | 2200 |
| Tina | 2300 |

Prototype

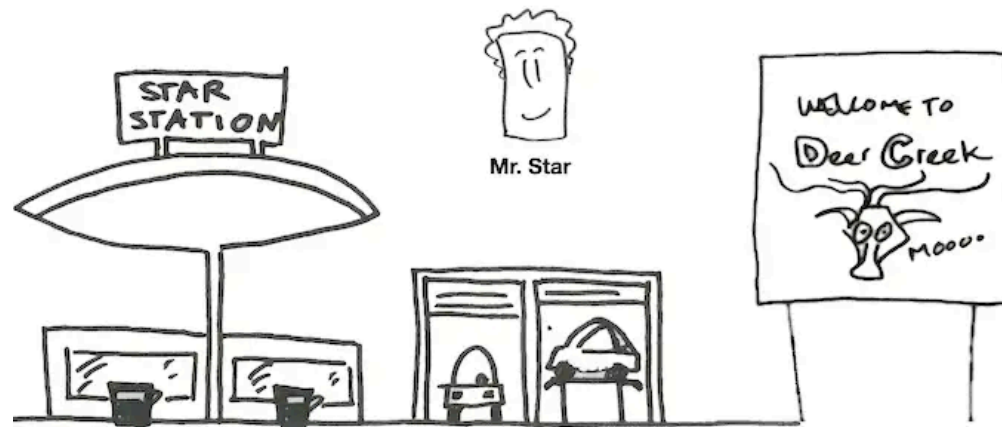
- In short, he wants to replace Excel sheets.
- His nephew (a wannabe software engineer) has already made a small script (prototype).

We have a user request, which we translate into requirements.

Business Logic

- The first step is to understand the business logic
- Without a proper understanding of business logic, we cannot design software.
- The business logic is also called domain knowledge .

- We can understand the business logic.
- We need to build software that tracks the names and salaries of employees.



The Request Details

- We have an Excel spreadsheet with all the information.

Employees



This is Vera. She is the manager at Star Station and the cashier



This is Roberto. He mans the kitchen and makes lovely sandwiches



Chuck and Samantha are the station attendants who fill up the gas tanks with gasoline.



This trio is Dave, Tina and Joe. Whatever is wrong with your car, they can fix it!

In the Real-World Situation

- We may not have well-defined requirements.
- It is because, **clients do not know what they need or want**: It is called: "unknown unknowns".
- However, we need to make requirements for designing software.

Unknown unknowns to known unknowns

- When they see the software, they begin to realize what they did not know.
- By building the software, we also begin to realize what we did not know.

Expect changes

- We should start expecting the requirements to change.
- We should build software that is designed to be changed.
- We should keep working with clients to get feedback.

Business Logic and Requirements

- It is not possible to deliver what users need without correct requirements.
- We must understand the system's business logic (problem domain) to obtain the correct requirements.

Business Logic in the Age of AI

- With vibe coding, anybody can make code.
- So, the shift is from implementation to design and architecture.
- Software engineers should also be experts in business logic.

Warning About Starting Without Requirements

- When we start a software project **without requirements** or a good understanding of business logic, we will be in deep trouble sooner or later.

- In this request for salary tracking software from Mr. Star, we can understand the business logic.
- We can translate Mr. Star's email into requirements.

Requirements

- Requirements are a set of descriptions of user requests.
- Requirements are the communication tool: clients and software engineers can agree upon the software to be delivered.

Requirements Version 1

- The software shows a list of the employees with their salaries.

- According to this requirement, if we build software that shows a list of employees with their salaries, we will solve the problem.

User Stories

- However, the description is not accurate enough.
- We have a better way of writing down requirements with **user stories**.
 - It has the format: As a [WHO], I want [WHAT], so that [WHY].

- It identifies three pieces of information: **who** (line 1), **what** (line 2), and **why** (line 3) of the requirement.
- Now we are ready to start.

Requirements Version 2

1. As a "manager,"
2. I want to "see a list of the employees with their salaries,"
3. so that "I can easily understand what is going on."

Prototype (Version 0)

- Even though we have the requirements in the form of `User stories`, we still don't know what we don't know.
- We make version 0 software (prototype) to know what we don't know.

GitHub repository
`ase420/code/00P/v0-prototype`

prototype-v1.py

- Marty already made a prototype.
- Marty does not know software design, so he made code with some smell.


```
1 names = ["Vera", "Chuck", "Samantha",  
2          "Roberto", "Joe", "Dave", "Tina"]  
3 salaries = [2000, 1800, 1800, 2100,  
4             2000, 2200, 2300]  
5  
6 for n in names: print(n)  
7 for s in salaries: print(s)
```

```
Vera  
Chuck  
Samantha  
Roberto  
Joe  
Dave  
Tina  
2000  
1800  
1800  
2100  
2000  
2200  
2300
```

- Marty already identified the problem in the code.

Dear Uncle,

this shows a list of your employees. And then it shows a list of all the salaries. Unfortunately I don't know how to connect the names and salaries and I'm late for school.

Your nephew,
Marty

Code smell

- Names and salaries are printed separately. There is no connection between the employee and their salary!

prototype-v2.py - Fixing the Connection

- We can address Marty's problem using the index.

```
1  count = len(names)
2  for c in range(count):
3      print(f"{names[c]}, ${salaries[c]}")
```

```
Vera, $2000
Chuck, $1800
Samantha, $1800
Roberto, $2100
```

We Hire Ringo, then What?

- We hire a new employee, Ringo, so we need to **change** our software.
- But we forgot to add Ringo's salary to the salaries list.

```
1  # Added Ringo
2  names = ["Vera", "Chuck", "Samantha",
3           "Roberto", "Joe", "Dave", "Tina", "Ringo"]
4  # OOPS! Forgot to update salaries
5  salaries = [2000, 1800, 1800, 2100, 2000, 2200, 2300]
6
7  count = len(names)
8  for c in range(count):
9      print(f"{names[c]}, ${salaries[c]}")
```



prototype-v3-bug.py - The Error

- And we get an error message.
- Maybe we can easily fix the problem from the error message.

```
Traceback (most recent call last):  
  File "/code/prototype/v3-bug.py", line 6, in <module>  
    print (f" {names [c]}, ${salaries[c]}")  
IndexError: list index out of range
```

Python - The pitfall

- Some static language compilers could find this error when we compile the code.
- Unfortunately, Python is a dynamic language; it does not execute that checking.
- We can find the error only when we run the code.

prototype-v3-bugfix.py

We add Ringo's salary to the list and solve this issue.

```
1  names = ["Vera", "Chuck", "Samantha",  
2          "Roberto", "Joe", "Dave", "Tina", "Ringo"]  
3  salaries = [2000, 1800, 1800, 2100, 2000, 2200, 2300, 1900]  
4  
5  count = len(names)  
6  for c in range(count):  
7      print(f"{names[c]}, ${salaries[c]}")
```

```
Vera, $2000  
Chuck, $1800
```


Joe Quits, then What?

- Joe quits.
- So, we need to **change** our software again.
- But again, we forgot to delete Joe's salary.



Joe

```
names = ["Vera", "Chuck", "Samantha", "Roberto", "Dave", "Tina", "Ringo"]
salaries = [2000, 1800, 1800, 2100, 2000, 2200, 2300, 1900]

count = len(names)
for c in range(count):
    print(f"{names[c]}, ${salaries[c]}")
```

Code smell and the rule of three

- When one change impacts multiple parts of the code, it is a code smell.
- When we do the same thing three times, it is time to act.

prototype-v4-bug.py - Silent Bug

- When we check the output, we find something wrong.
- Problem: Dave shows Joe's salary (\$2000), Tina shows Dave's salary (\$2200), etc.

- We forgot to delete Joe from the salary list.
- This bug is **harder to find** than before because Python did not print out any error messages.

| | |
|------------------|------------------|
| Vera, \$2000 | Vera, \$2000 |
| Chuck, \$1800 | Chuck, \$1800 |
| Samantha, \$1800 | Samantha, \$1800 |
| Roberto, \$2100 | Roberto, \$2100 |
| Joe, \$2000 | Dave, \$2000 |
| Dave, \$2200 | Tina, \$2200 |
| Tina, \$2300 | Ringo, \$2300 |
| Ringo, \$1900 | |

prototype-v4-bugfix.py

We deleted Joe's salary, and everything worked fine.

```
names = ["Vera", "Chuck", "Samantha", "Roberto", "Dave", "Tina", "Ringo"]  
salaries = [2000, 1800, 1800, 2100, 2200, 2300, 1900] # fix
```

```
Vera, $2000  
Chuck, $1800  
Samantha, $1800  
Roberto, $2100  
Dave, $2200  
Tina, $2300  
Ringo, $1900
```

Change - the source of complexity

- Even when we finish our implementation, the problem domain keeps changing.
- We should make software in a way that can **meet changes effectively**.

Software Design - the tool for changes

- The current approach with separate lists is error-prone and hard to maintain.
- We need to design software to meet the changes.

More Changes are coming.

- Mr. Star likes the program but wants to add a **job title** to the report.

| NAME | SALARY | JOB TITLE |
|----------|--------|------------|
| Vera | 2000 | Manager |
| Chuck | 1800 | Attendant |
| Samantha | 1800 | Attendant |
| Roberto | 2100 | Cook |
| Dave | 2200 | Car Repair |
| Tina | 2300 | Car Repair |
| Ringo | 1900 | Car Repair |



Vera, \$2000, Manager
Chuck, \$1800, Attendant
Samantha, \$1800, Attendant
Roberto, \$2100, Cook
Dave, \$2200, Car repair
Tina, \$2300, Car repair
Ringo, \$1900, Car repair