JAIG Brief Reference

The main features of JAIG are:

- Sending requests to the **OpenAI GPT API**
- **Including source files** into the request
- Seamless **integration** with the development environment (**IDEA**)
- **View requests during the generation process**, code highlighting
- **Parsing** of generation results, sorting of source files and packages
- **Writing** generated files **to src/main/java**
- Ability to write results to an arbitrary folder **(#save-to)**
- **Flexible** generation **settings**, including **model** selection (GPT3/GPT4 and others) and **temperature**, globally or on a per-request basis
- Creating and applying **patches** for code changes
- **Merging changes** in the code and the results of code generation using AI
- Generating a series of queries from **templates**
- Execution of a series of queries (**batches**)
- **Rollbacks**
- **Refactoring** the selected code fragment using AI in dialog mode, using the prompt library

Working with the Code Generator

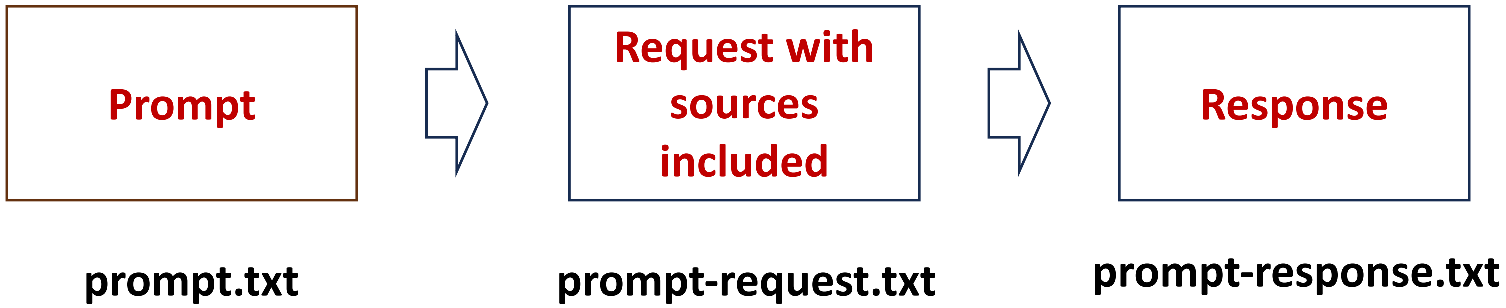
1) Create a folder
2) Place the request there in a file with a .txt extension (for example, **prompt.txt**)
3) Start generation by pressing JAIG button

Prompt

Response

Working with the Code Generator

4) Add **references** to the files or folders to the prompt
5) The contents of the files are added to the Request



Prompt

Request with sources included

Response

prompt.txt

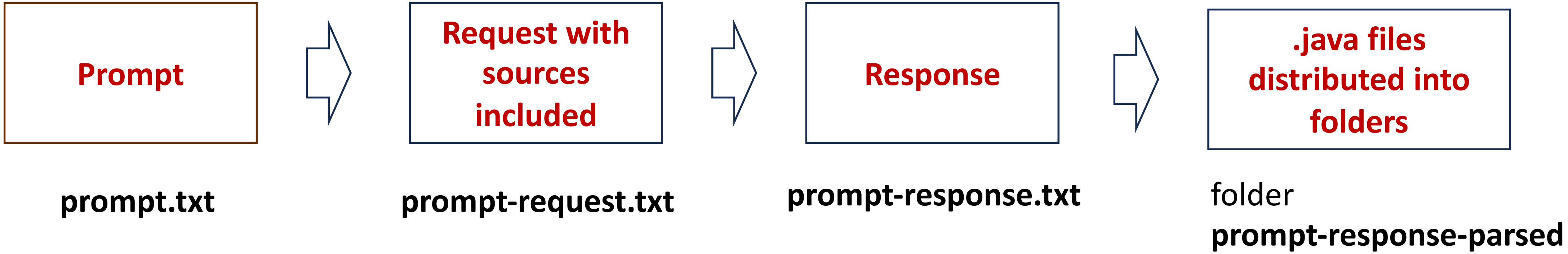
prompt-request.txt

prompt-response.txt

Working with the Code Generator + parsing

6) If the response contains code, and this code has packages specified (number of packages = number of classes and interfaces), **automatic parsing** of the response is performed: files are distributed into folders according to the package

7) The result of the parsing is written to the **prompt-response-parsed** folder



Prompt

prompt.txt

Request with sources included

prompt-request.txt

Response

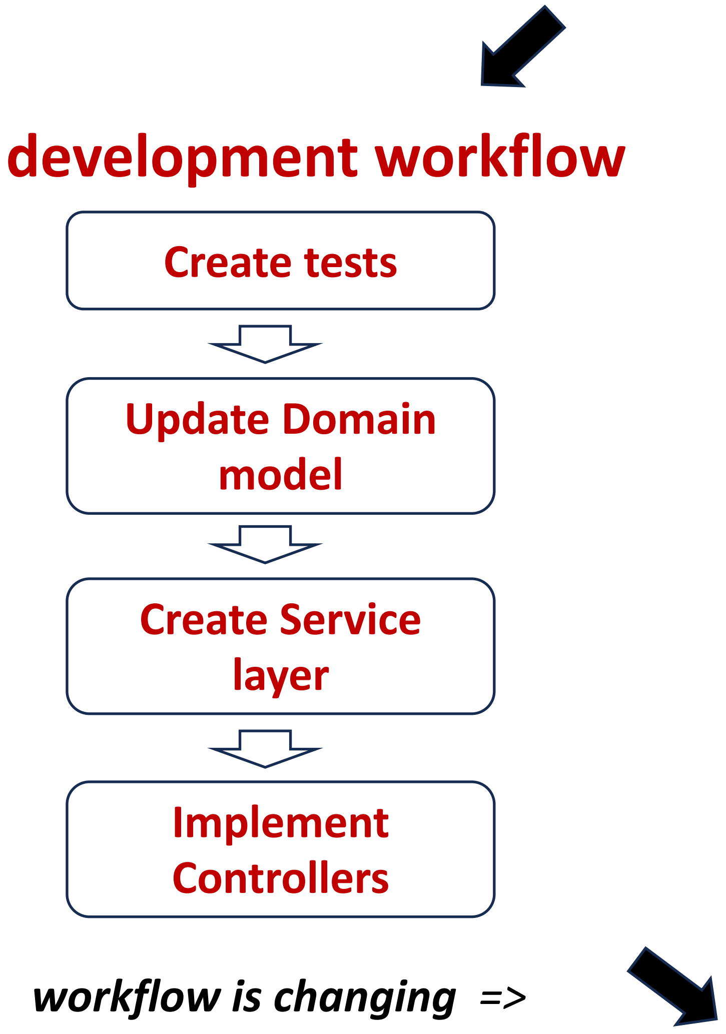
prompt-response.txt

.java files distributed into folders

folder **prompt-response-parsed**

Separation of the concerns

development workflow



workflow is changing *=>*
regenerate the code

business requirements

Requirements:

entity: course
requirements: |
- Find a course by name
- Find course by id
- Find all courses

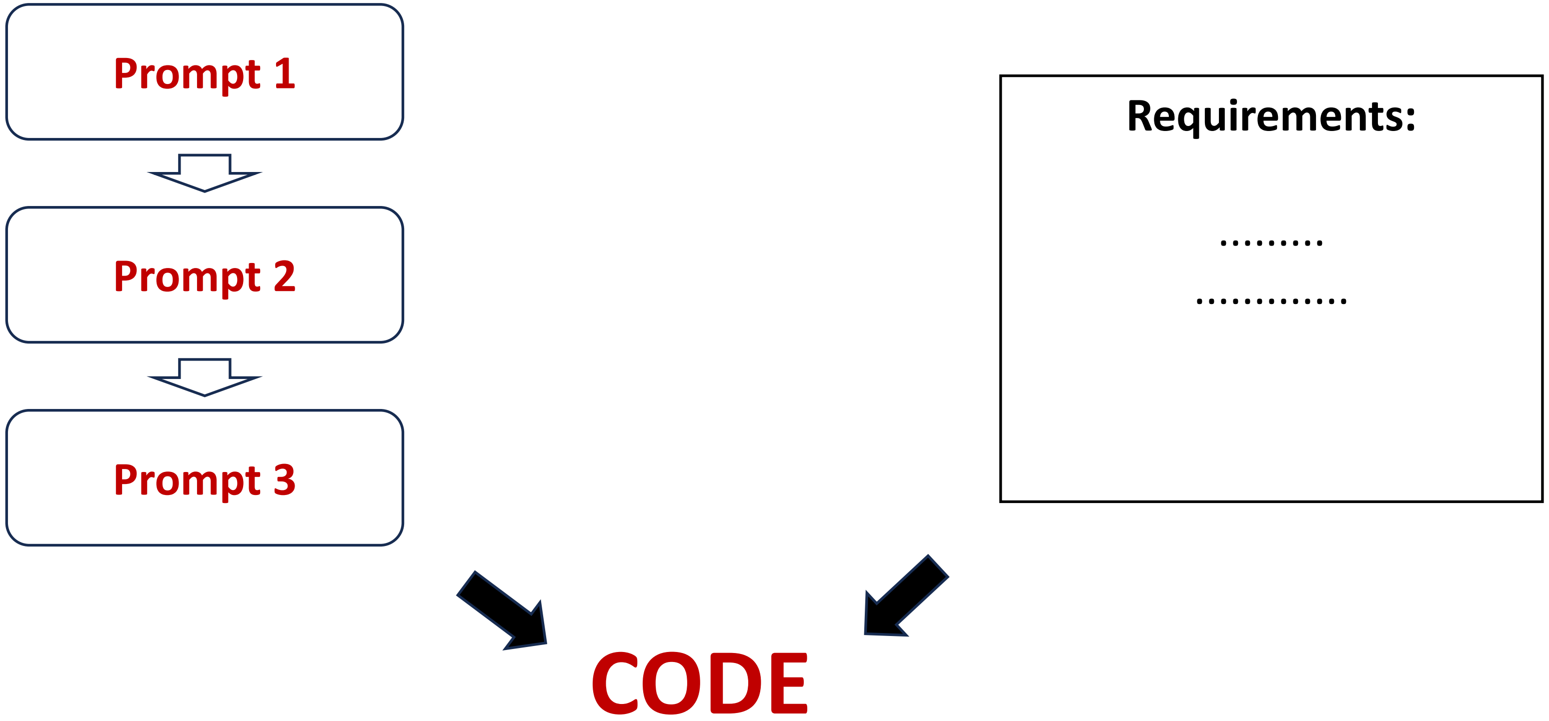
...

requirements are changing *=>*
regenerate the code

CODE

Development cycle with JAIG:

- Create a universal **prompts pipeline** that describes development workflow
- Apply any **business domain** to generate a standardized code



Prompt 1

Prompt 2

Prompt 3

Requirements:

....

....

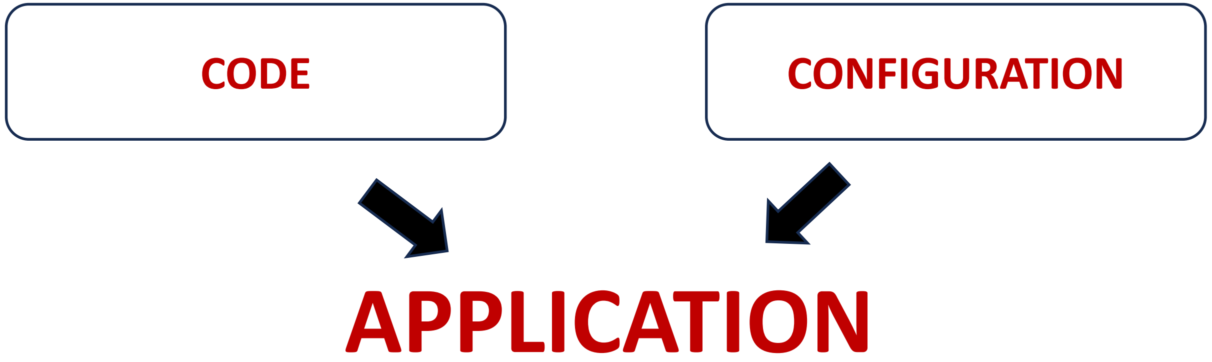
CODE

We are able to
change the pipeline
*and **regenerate***

We are able to
change requirements
*and **regenerate***

Compared: Spring vs. JAIG

Spring separates code from configuration

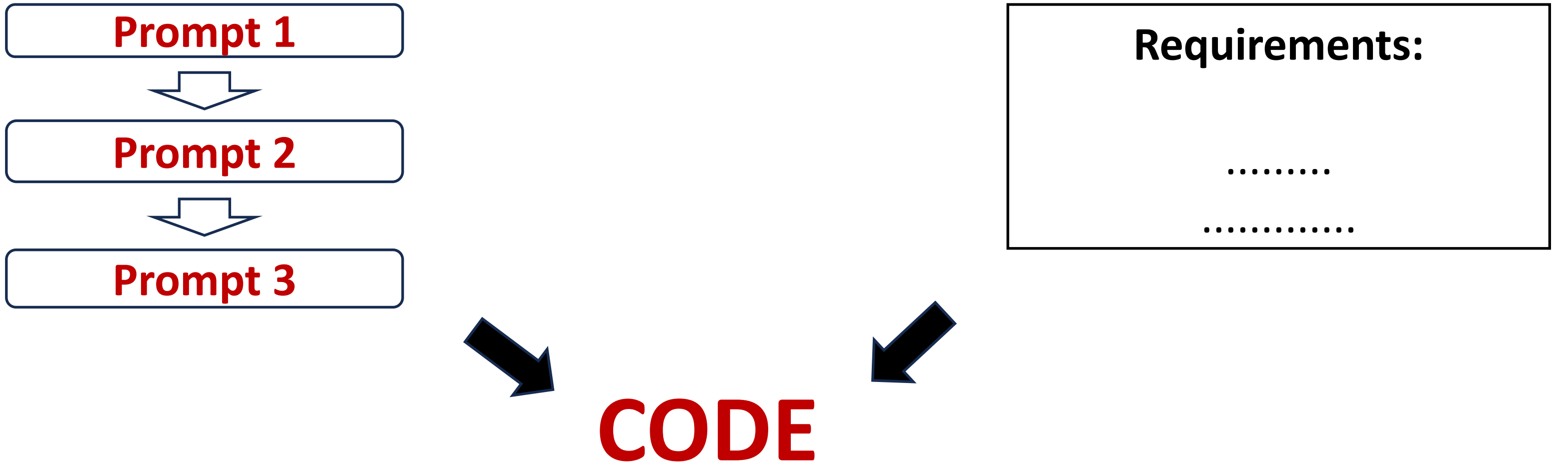


CODE

CONFIGURATION

APPLICATION

JAIG separates prompt pipeline to implement code (specific for your project) from requirements



Prompt 1

Prompt 2

Prompt 3

Requirements:

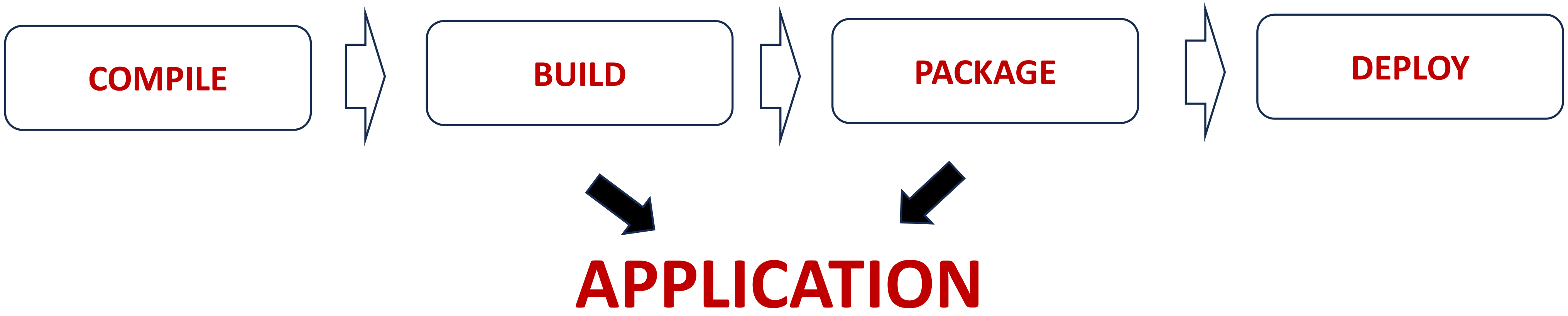
....

....

CODE

Compared: Maven vs. JAIG

Maven defines Life Cycle to build & deploy the Application



COMPILE

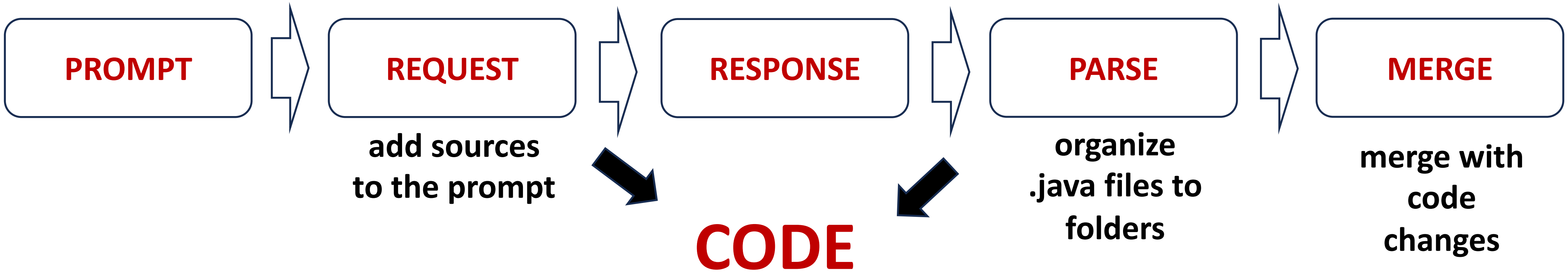
BUILD

PACKAGE

DEPLOY

APPLICATION

JAIG defines Life Cycle to create a Code for Application



PROMPT

REQUEST

add sources
to the prompt

RESPONSE

PARSE

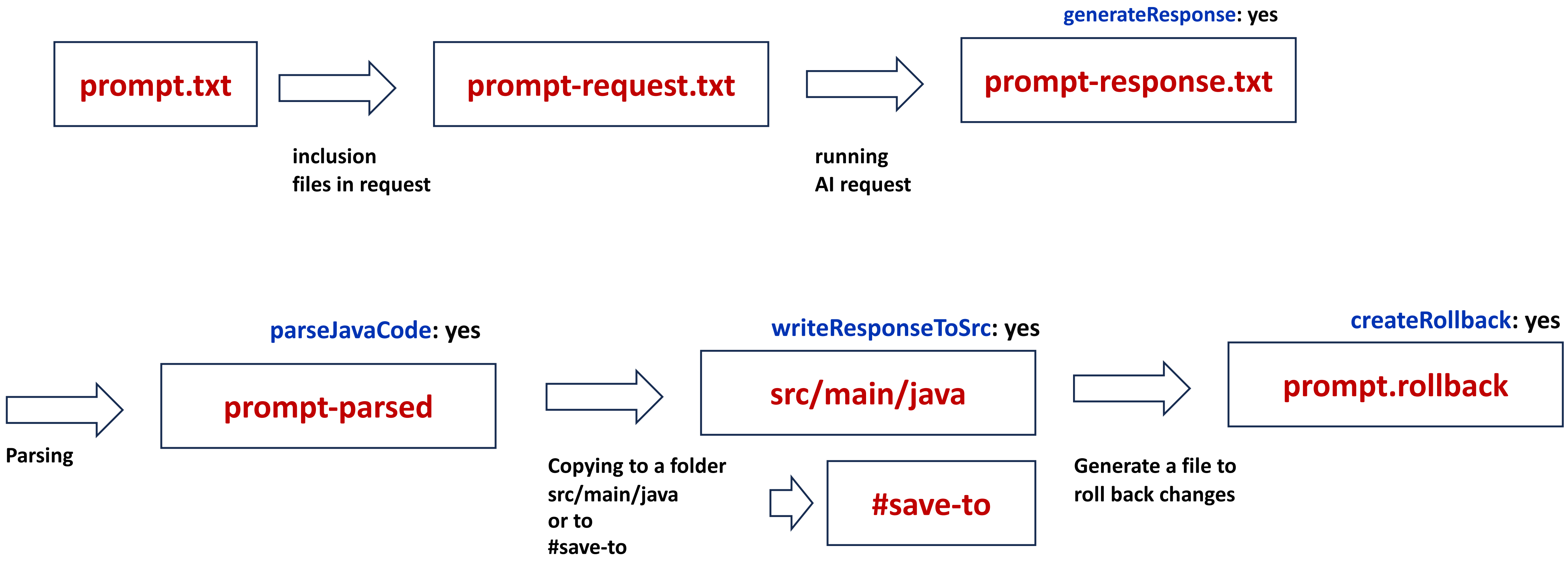
organize
.java files to
folders

MERGE

merge with
code
changes

CODE

JAIG Lifecycle: generated artifacts



prompt.txt

inclusion
files in request

prompt-request.txt

running
AI request

generateResponse: yes

prompt-response.txt

Parsing

parseJavaCode: yes

prompt-parsed

Copying to a folder
src/main/java
or to
#save-to

writeResponseToSrc: yes

src/main/java

#save-to

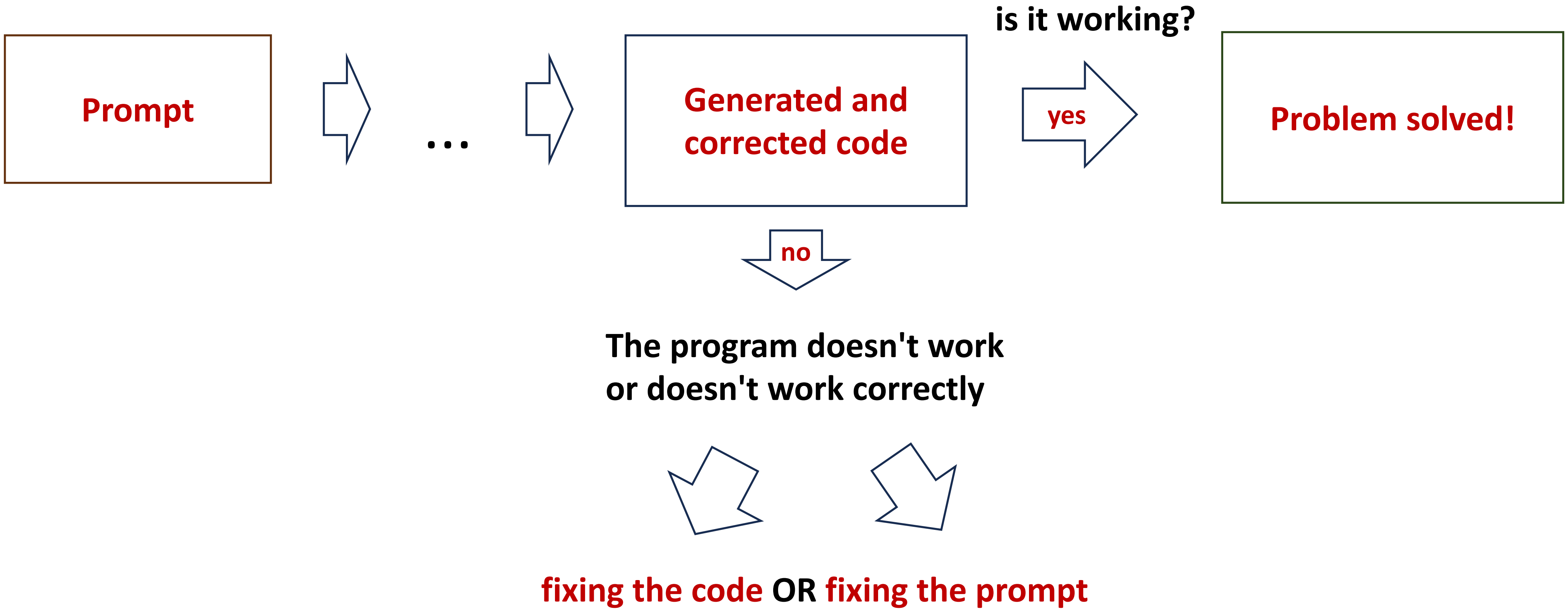
Generate a file to
roll back changes

createRollback: yes

prompt.rollback

Working with the Code Generator: Testing results

Now, the code needs to be executed and tested



Prompt

…

Generated and corrected code

is it working?

yes

Problem solved!

no

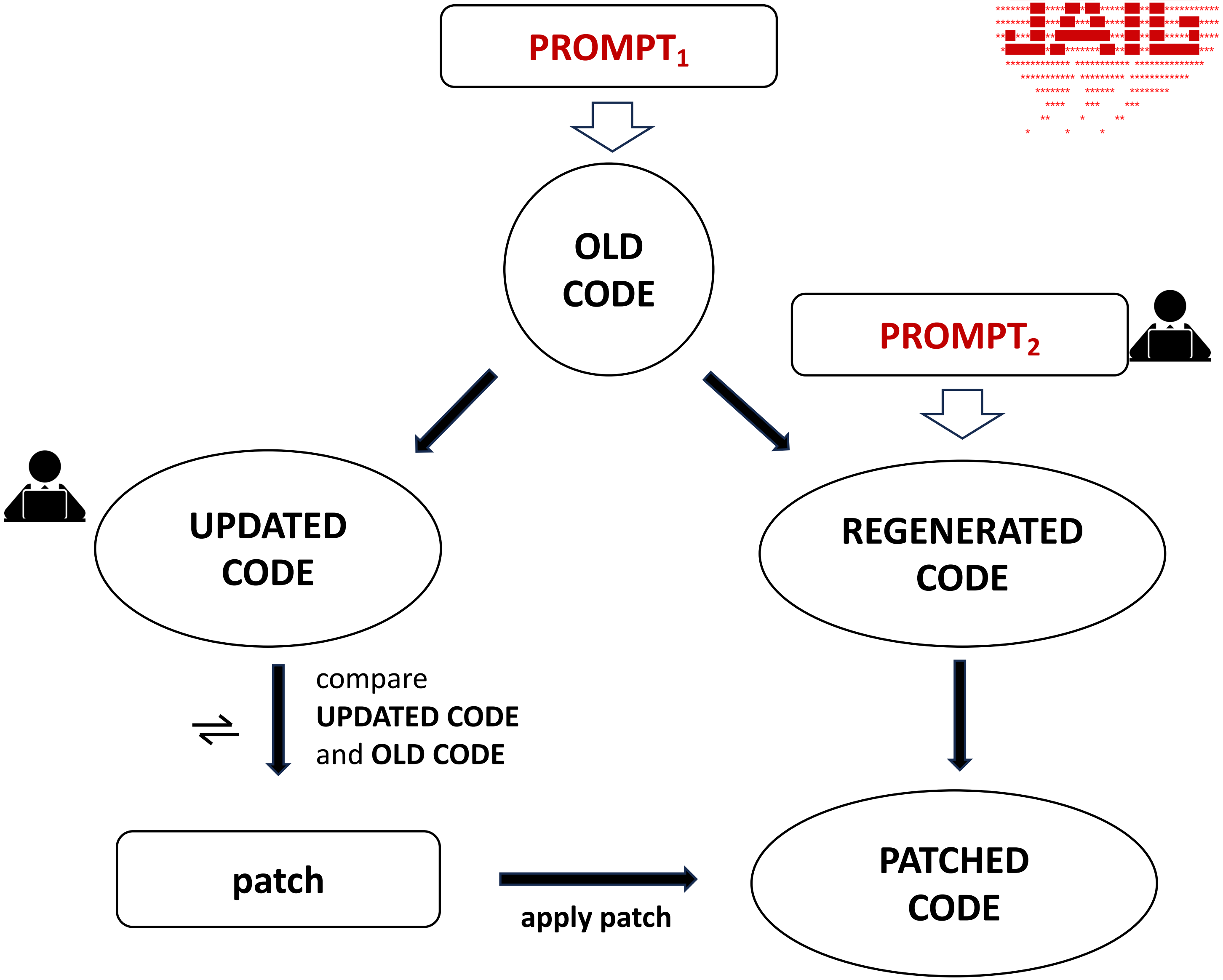
The program doesn't work or doesn't work correctly

fixing the code OR fixing the prompt

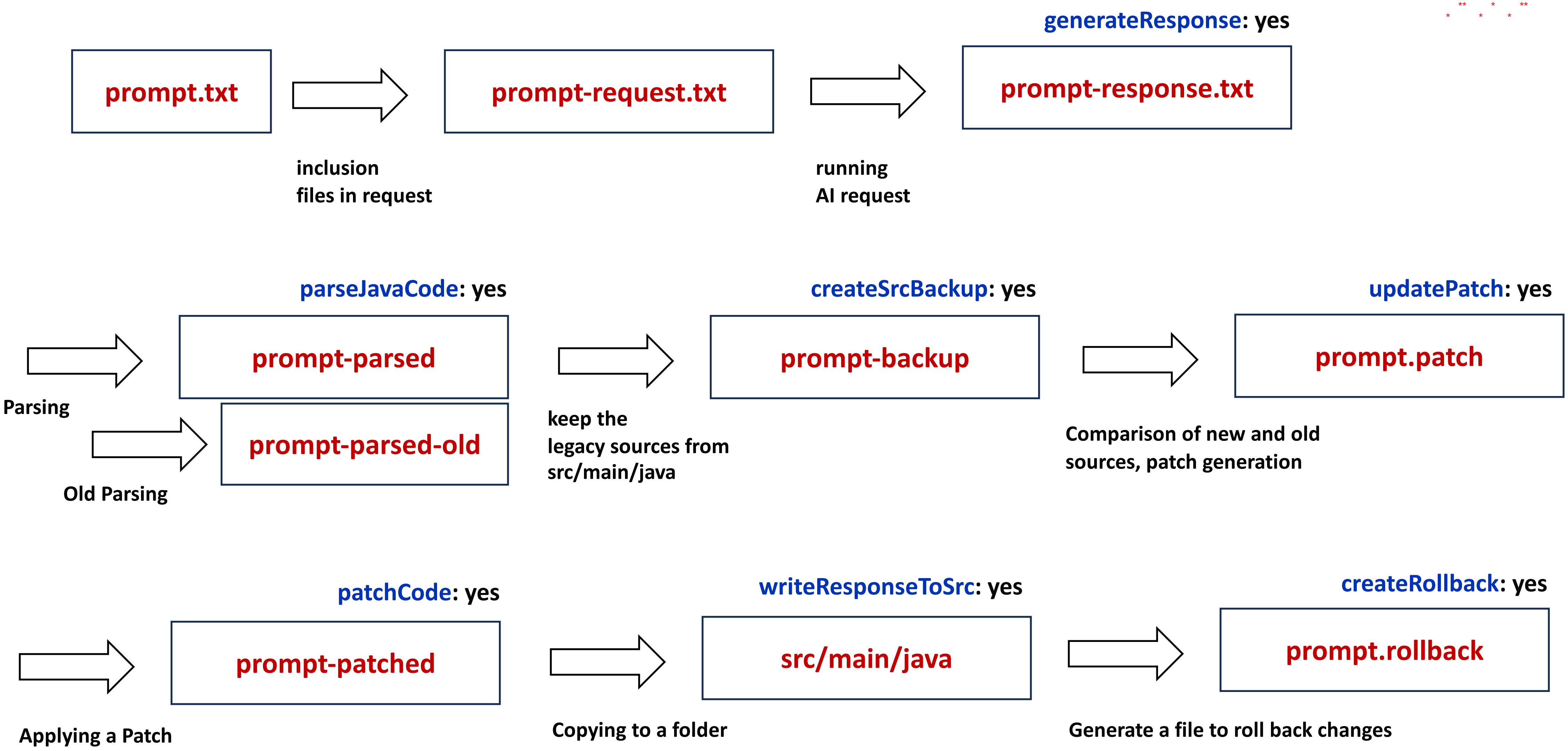
Patch GPT response

We can have
2 types of changes:
- In the **code**
- In the **prompt**

If you have **both**, you need to **merge** them



JAIG Lifecycle with Patching: generated artifacts



prompt.txt

inclusion
files in request

prompt-request.txt

running
AI request

generateResponse: yes

prompt-response.txt

Parsing

parseJavaCode: yes

prompt-parsed

Old Parsing

prompt-parsed-old

keep the
legacy sources from
src/main/java

createSrcBackup: yes

prompt-backup

Comparison of new and old
sources, patch generation

updatePatch: yes

prompt.patch

Applying a Patch

patchCode: yes

prompt-patched

Copying to a folder

writeResponseToSrc: yes

src/main/java

Generate a file to roll back changes

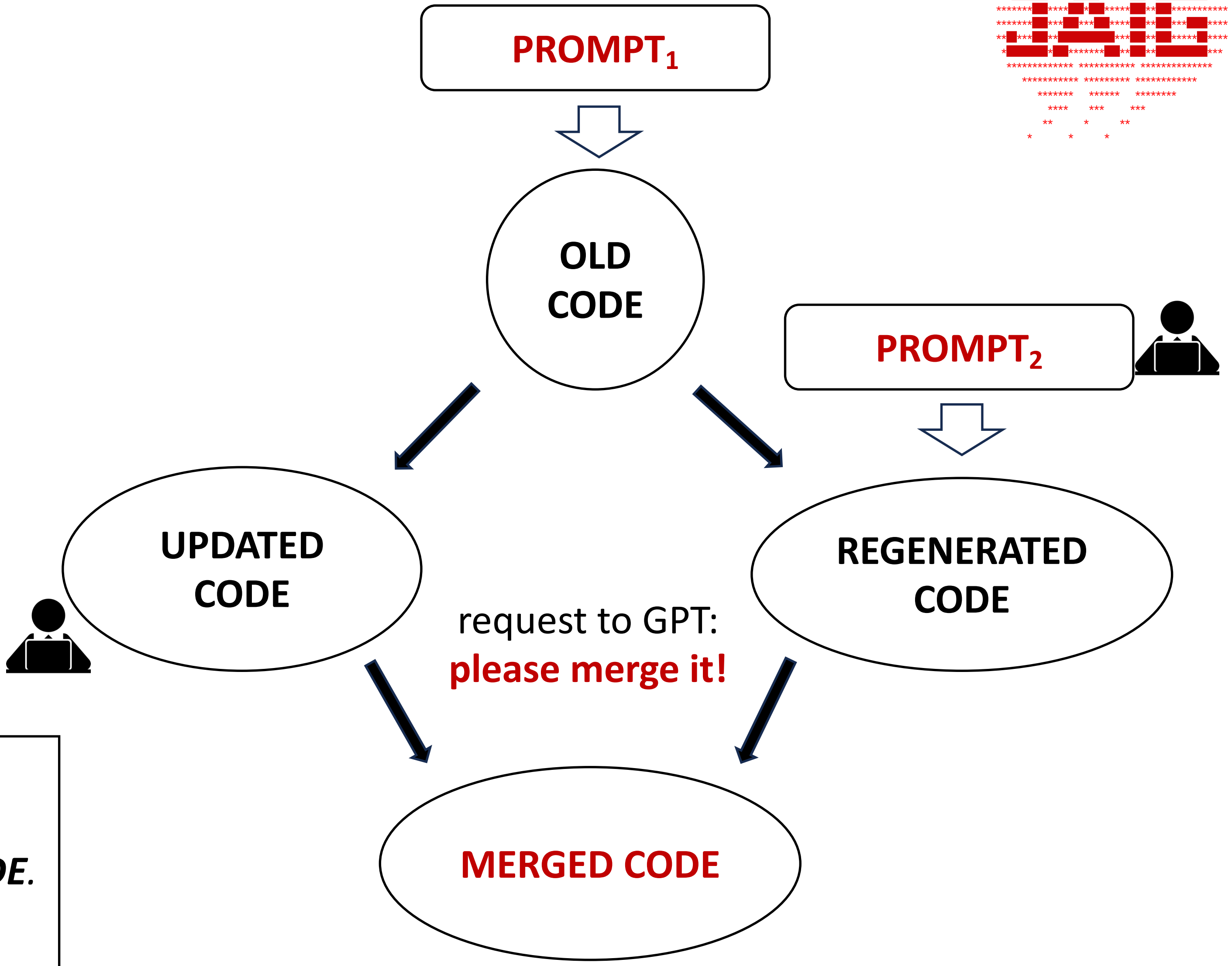
createRollback: yes

prompt.rollback

Merge code and GPT response

We can have
2 types of changes:
- In the **code**
- In the **prompt**

If you have **both**, you need to **merge** them



PROMPT$_1$

OLD CODE

PROMPT$_2$

UPDATED CODE

REGENERATED CODE

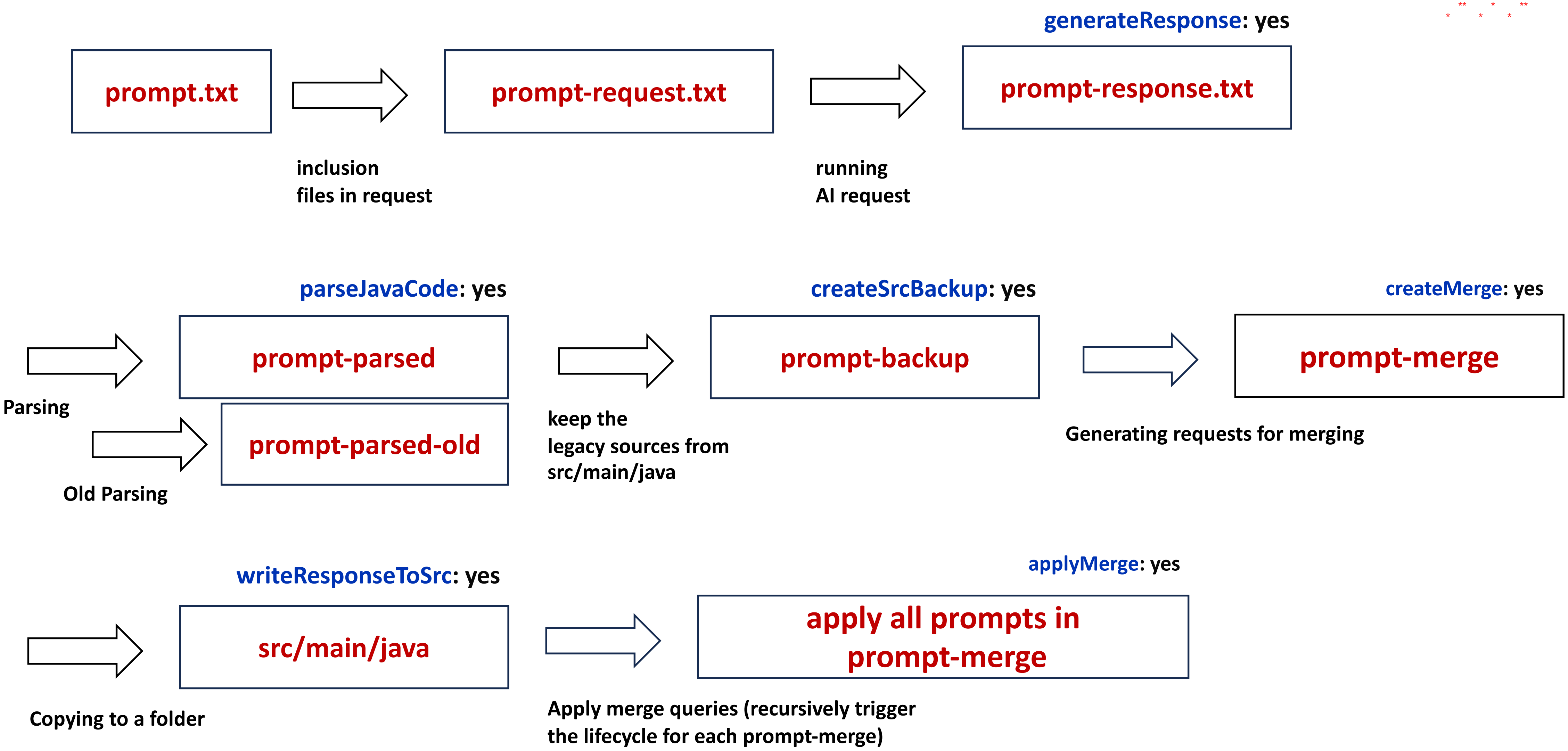
request to GPT:
please merge it!

MERGED CODE

merge prompt be like**:**

Dear Chat GPT,
*I had **OLD CODE**.*
*Then programmer changed it to **UPDATED CODE**.*
*And you changed it to **REGENERATED CODE**.*
*Can you please **merge all** these changes?*

JAIG Lifecycle with Merging: generated artifacts



generateResponse: yes

prompt.txt → prompt-request.txt → prompt-response.txt

inclusion
files in request

running
AI request

parseJavaCode: yes

prompt-parsed

Parsing

prompt-parsed-old

Old Parsing

createSrcBackup: yes

prompt-backup

keep the
legacy sources from
src/main/java

createMerge: yes

prompt-merge

Generating requests for merging

writeResponseToSrc: yes

src/main/java

Copying to a folder

applyMerge: yes

apply all prompts in
prompt-merge

Apply merge queries (recursively trigger
the lifecycle for each prompt-merge)

Patching or merging?

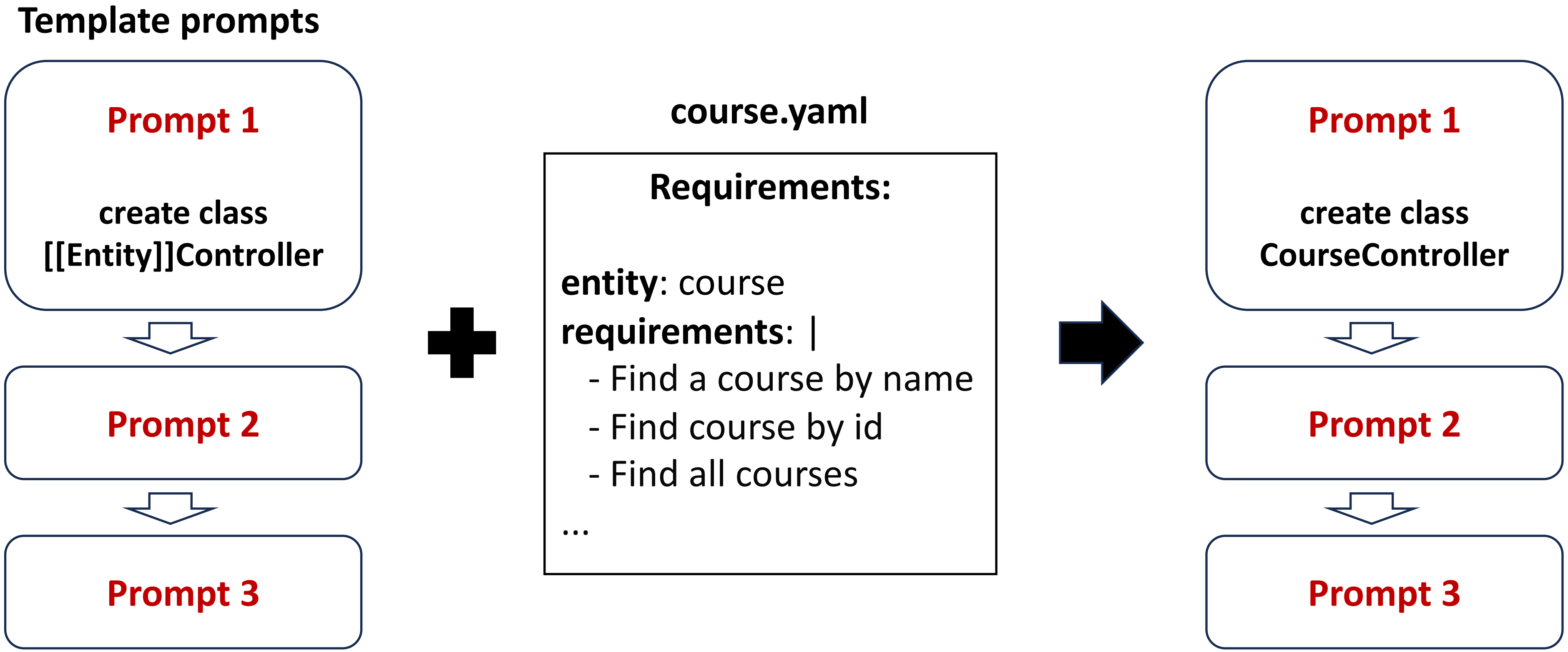
Patching

- **Patching is faster** and doesn't require an AI generation
- Patching can be **precisely controlled**
- For patching, we can open the **<prompt>.patch** file and see exactly what has been changed
- Works well for simple cases, but **fails on more complicated**

Merging

- For merging, we're **asking the AI** to merge the changes => it is **slower** and more **expensive**
- The work of merging is **less reliable** and cannot be controlled
- Merging is better suited in **complex cases** when you need to merge non-obvious changes in the code and in the generation results

JAIG Templates



Template prompts

Prompt 1

create class [[Entity]]Controller

Prompt 2

Prompt 3

course.yaml

Requirements:

```
entity: course
requirements: |
  - Find a course by name
  - Find course by id
  - Find all courses
...
```

Prompt 1

create class CourseController

Prompt 2

Prompt 3

entity: course

Entity: Course

entities: courses

Entities: Courses

JAIG can be applied to:

To apply JAIG, open the file or focus on file/folder in Project Tree, then press JAIG button

.txt file -> send **prompt** to GPT and start the **JAIG lifecycle**

FOLDER -> find all .txt files, **create a .batch** with all prompts found in subfolders

OR **cleanup folder** (remove all generated files/folders)

FOLDER like NN_smth (e.g. 05_smth) -> insert the folder and shift all the following folders down

.yaml file with the specified template (e.g. template: templates/rest)

-> read the **requirements** and **create prompts** from the **template**

.batch file -> treat each line as a prompt and **execute sequentially**

.patch file -> **apply the patch** to the -parsed folder, save the result to -patched

-parsed.rollback file -> apply rollback by removing generated files or restoring files to the original state

-full.rollback file -> roll back all changes made by this prompt

-response.txt file -> start the JAIG lifecycle **without sending it to the AI**

.java + selected code -> JAIG **refactoring mode**

JAIG Prompt Directives

All directives should be placed at the beginning of the line in the prompt file.

#temperature: 0.0

#model: gpt-4 (you can find the available models in JAIG/JAIG.yaml)

#src – write to the **src/main/java** folder (more precisely, to the **srcFolder** folder configured in JAIG.yaml)

#test – write to the **src/test/java** folder (more precisely, to the **testFolder** folder configured in JAIG.yaml)

#save-to: path – write the AI's response to the specified file on the path path

#nomerge – prevent merging

#merge – allow merging (same as createMerge: yes in JAIG.yaml)

#patch – create a patch (same as createPatch: yes in JAIG.yaml)

#apply-merge – apply merge automatically

#norollback – prevent automatic rollback before re-running (if there is a .rollback file)

#merge-incomplete: <comma-separated list of classes> - merge existing code with GPT output(if response is incomplete)

All other lines starting with # are considered comments

Insert a folder in the middle of a prompt sequence

- Often, in the list of folders to generate, you need to **insert the folder in the middle** of the sequence
- If you apply JAIG to a folder like NN_something, such as 05_prompt, it will prompt you to enter a folder name
- When a name is entered, the folder will be named 06_folder_name, and all old folders 06_prompt, 07_prompt, etc., will receive an index one more.

1) click on the folder, press JAIG

- 01_rest_test
- 02_rest
- 03_controller
- 04_service
- 05_repository
- 06_unit_test

2) Enter the name of the folder: **dto**

3) A new folder **05_dto** inserted after 04_service:

01_rest_test
02_rest
03_controller
04_service
05_dto
06_repository
07_unit_test

Rollbacks

- When the code is generated, a **.rollback** file is created
- We can apply JAIG to **.rollback** file and all the changes will rollback **1 step back**.
- We have 2 possible .rolback files:
  - **<prompt>-parsed.rollback** allows to go **1 step back**
  - **<prompt>-full.rollback** allows to revert to the state before prompt was applied first time
- Rollbacks are **applied automatically** (if **applyRollback: yes** is set in **JAIG.yaml**) to be able to improve the request – this way we **start over every time**
- This means that **before the prompt is launched**, all **previous changes will be rolled back**, regardless of the changes made to the source code
- This is useful for **initial polishing of the prompt**, but is not suitable if you are going to change the source code (in this case, use $patch or #merge)
- Rollbacks are **never applied automatically** if one of these directives used: **#patch**, **#merge**, **#merge-incomplete**, **#norollback**)
  - **Patching** and **merging** need the folders like **backup** and **parsed-old not to be removed**, that's why it is incompatible with automatic rollbacks)

Here's an example of rollback file: **create-enum-parsed.rollback**: we restore the updated sources from the backup, cancelling all changes done by JAIG.

```
Restore src/main/java/test/Semaphore.java
from    JAIG/test-parse/create-enum-backup/test/Semaphore.java
Restore src/main/java/test/SemaphoreManager.java
from    JAIG/test-parse/create-enum-backup/test/SemaphoreManager.java
```

- If **applyRollback: no** is set in **JAIG.yaml,** you can add a **#rollback** directive to a prompt to apply the rollback automatically

Parsing <prompt>-response.txt

- After receiving the results of AI generation, a <prompt>-response.txt file is created with a response from the AI
- We can **apply JAIG to <prompt>-response.txt** instead of prompt
- In this case, the file content is parsed and patched, **without** having to **call the AI** again
- **This is useful for:**
    - **Demonstrations**: In this case, you can demonstrate how JAIG works (parsing, patching, etc.) without having to rely on the GPT generation (if you already have **<prompt>-response.txt**)
    - **Problems in the response parsing** (for example, the package before each class is not specified or incorrectly specified) which prevents response from parsing and we need to **make changes manually before parsing**
        - Parsing is possible ONLY if number of classes/interfaces (or other aritifacts defined in javaFileNameRegexp) is the same as number of packages
        - Otherwise, we can change prompt and **ask to generate package** for every class OR change **<prompt>-response.txt** manually

If we are not satisfied with the results of parsing (for example, it wasn't able to find Java classes), we can:

- Roll back changes (apply JAIG to .rollback)
- Edit **<prompt>-response.txt** file
- Apply JAIG to **<prompt>-response.txt**