During development, I face some important things that must be kept somewhere. some delicate details about classes, function parameters, etc. Moreover, some assumptions that I used for development. some of these information are important for developers, some for users. How developers, or teams manage these information keep them, store and maintain?

**ChatGPT said:**

This is a very important topic in sustainable software development: **how to capture, store, and maintain critical project knowledge** — both for developers and users.

Here’s a guide to best practices, tools, and structure for managing this information:

**🧠 1. Classify the Information**

Separate the types of knowledge into categories:

| **Type** | **Examples** | **Audience** |
| --- | --- | --- |
| **Design decisions** | "Why we use fr\_model\_params via kwargs", architectural assumptions | Developers |
| **Internal conventions** | "T-norms must be lowercase", "Don't use \_\_init\_\_ in set\_params" | Developers |
| **Usage assumptions** | "X must be pre-normalized", "Models require positive-definite matrices" | Both |
| **Function/class details** | Parameter types, valid options, edge cases | Developers |
| **User-facing behavior** | Config options, install instructions, output format | Users |

**📁 2. Where to Store This Information**

| **Format** | **Use Case** | **Tool Examples** |
| --- | --- | --- |
| **Code comments** | Local hints in functions and classes | Docstrings (Doxygen, reST) |
| **Markdown docs** | Rich, navigable, readable project documentation | /docs/, GitHub Wiki, MkDocs |
| **README files** | Intro + usage guide per module/subsystem | README.md in each subfolder |
| **Developer guide** | Internal assumptions, conventions, code style | CONTRIBUTING.md, /dev\_notes/ |
| **Config schemas** | Centralized param types, ranges, options | Pydantic models, JSONSchema |
| **Changelog / history** | What changed and why | CHANGELOG.md, Git tags |
| **Issues & discussions** | Ephemeral decision-making and Q&A | GitHub Issues, Notion, Jira |

**📦 3. Folder Structure Example**

bash

CopyEdit

project/

├── README.md # Intro for users

├── CONTRIBUTING.md # Dev setup, testing, coding conventions

├── docs/ # User and API documentation

│ ├── usage.md

│ ├── api\_reference.md

│ └── assumptions.md # Design/usage assumptions

├── dev\_notes/ # Internal developer knowledge

│ ├── fuzzy\_rough\_models.md # Model-specific quirks

│ ├── smote\_behavior.md

│ └── naming\_conventions.md

├── tests/

│ └── ...

├── src/

│ └── your\_modules/

**✅ 4. Best Practices**

* ✅ **Keep dev notes in version control**
* ✅ **Write documentation close to code** (e.g., docstrings, per-class README)
* ✅ **Use structured formats** for config (YAML/JSON)
* ✅ **Link related issues/decisions** in the code via comments
* ✅ **Use templates** for adding new components (docstring format, test naming)

**✍️ 5. Suggested Tools**

| **Tool** | **Role** |
| --- | --- |
| **MkDocs** | Fast static site from Markdown |
| **Sphinx** | Auto API docs from Python |
| **Jupyter Notebooks** | For documenting experiments |
| **Notion/Confluence** | Team-wide structured notes |
| **GitHub Projects/Wiki** | Lightweight but integrated |

Would you like me to scaffold a developer notes folder or a MkDocs setup for your fuzzy-rough project?