Living Documentation Pattern Language

The patterns of living documentation form a network, as language that can be used to describe a chosen path of implementation or, conversely, that can guide the user of the language in a generative way. To emphasize the key relationships and help visualize the pattern language, I have rendered diagrams of some of the patterns in the *Living Documentation* book, which are presented below.[1]

Note that the patterns themselves are not described in this section. For more details about each of them, please refer to the chapters of the book.

Living Documentation Essentials

As in Figure 1, rethinking documentation takes three forms, depending on the nature of the knowledge. For all the knowledge that is ephemeral, nothing beats conversations and working collectively as a way to transfer the knowledge between workmates.

At the other hand of the spectrum, the knowledge that is permanent—in other words, stable and useful in the long run—can be recorded through traditional forms of documentation, such as written documents and manually created diagrams.

In between these two cases, all the knowledge that is long-lived but evolves regularly is best considered in the system itself, where most of it already resides.

Even if the knowledge is there already, it probably is not totally complete and needs to be augmented. This knowledge augmentation is preferably done through an internal form of documentation; that is, in the artifacts of the system itself. It preferably references ready-made documentation from the literature rather than describing a solution as if it were original.

Developers can use all this knowledge, existing and augmented, directly within their programming environments as integrated documentation. It also enables them to use tools to exploit the knowledge to make it more convenient and more accessible through all the techniques of living documentation, such as curation, living diagrams, living glossaries, and many other examples of generated documentation as published snapshots.

[1] Initially rendered from plain-text descriptions using the Graphviz DOT renderer.

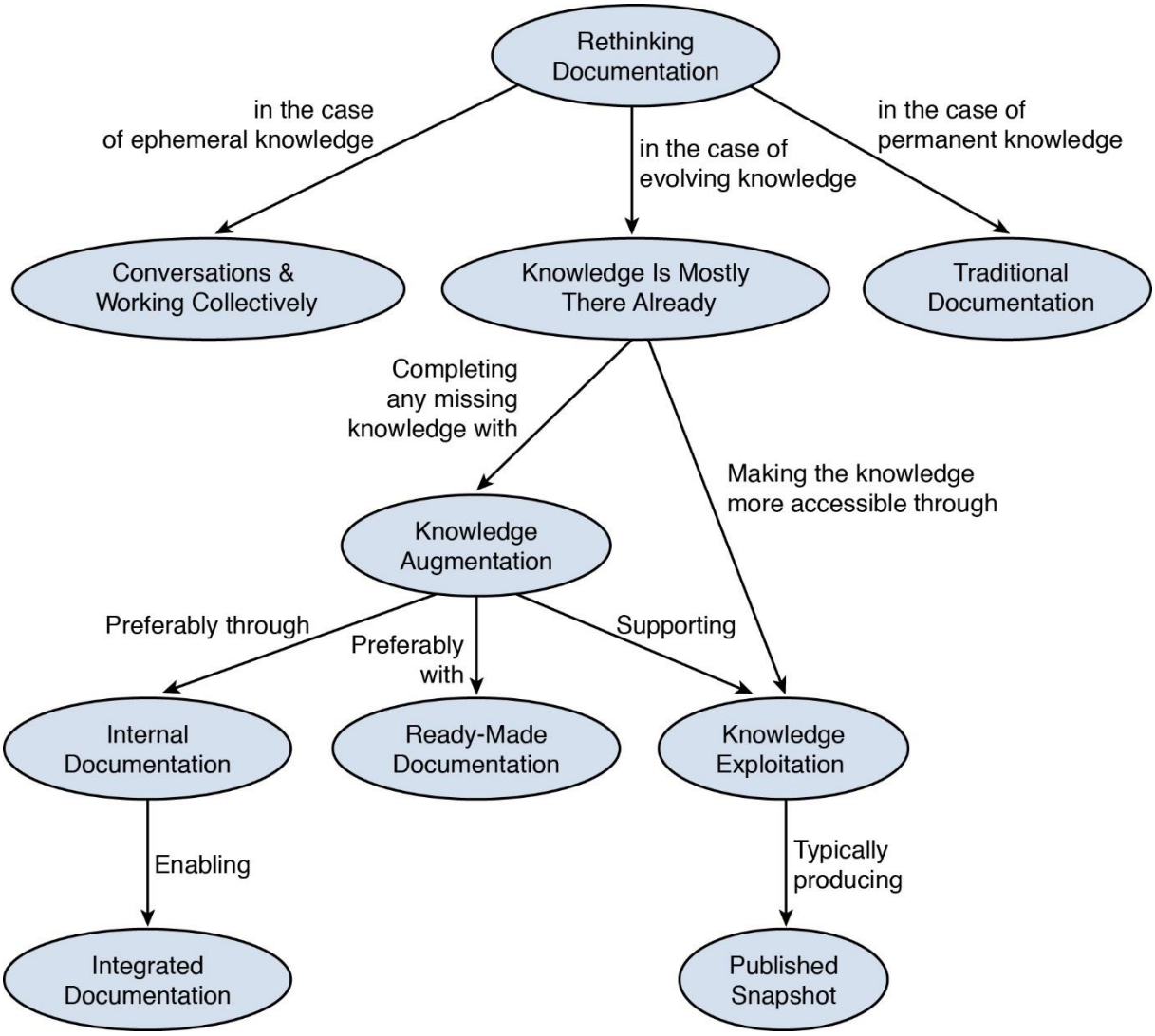

Figure 1: Essentials of living documentation

Knowledge Augmentation

System artifacts, such as code and configuration files, typically do not describe the rationale behind design decisions, the influences behind the reasoning of the team members. They tend to only describe the consequences of these decisions and reasoning. *Knowledge augmentation* is about putting that missing knowledge into the system, preferably in its artifacts directly, leading to an *internal documentation*, as illustrated in the diagram in Figure 2.

Knowledge augmentation for source code, which we call *augmented code*, can be achieved through *documentation by annotations* or through *documentation by convention*, both enabling *machine-accessible documentation*. Knowledge augmentation can also be done outside the system artifacts, as *external documentation* using *sidecar files* or a *metadata database*, though this is not the preferred option.

In any case, the knowledge augmentation is best done by putting the *intrinsic knowledge* onto the thing itself, while the *module-wide knowledge* should be put at the module level. And the augmented knowledge will reference *ready-made knowledge* from the literature whenever possible for best results.

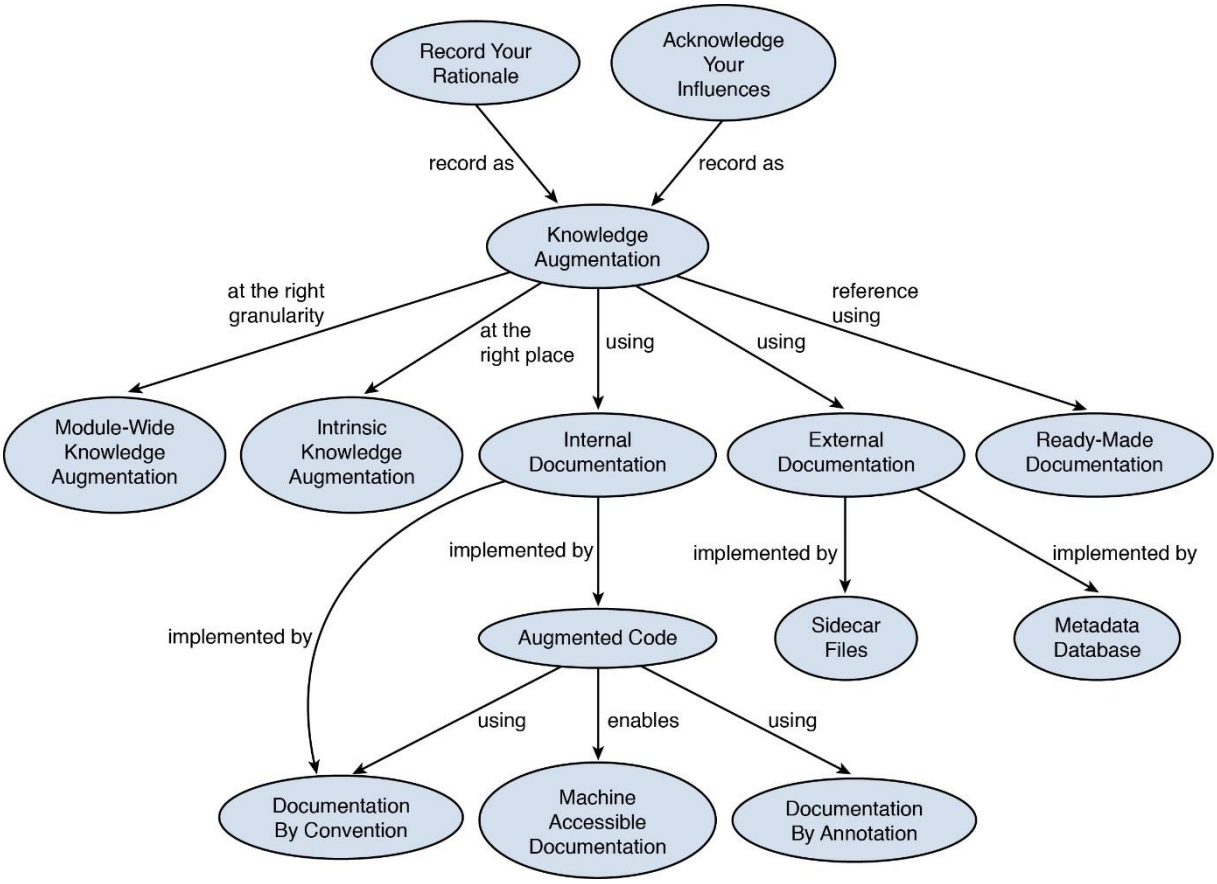

Figure 2: The pattern language of knowledge augmentation

Avoiding Traditional Documentation

Given our love-hate relationship with *traditional* forms of documentation, it's somehow satisfying to explore alternatives. Acknowledging that *conversations* are indeed a form of knowledge transfer is a good start, along with *working collectively* and having *informal communications* like the ones you have at the coffee machine, as shown at the top of the diagram in Figure 3.

From that, some ideas deserve to be recorded for the longer run, through a process of *idea sedimentation*. The ideas that deserve to last for a while can then evolve into *on-demand documentation*, can be thrown away later, or can be turned into some form of *living document* or other techniques described in the book, like *declarative automation* (automation code or configuration that reads well enough to be its own documentation) or *enforced guidelines* (knowledge enforced as automated checks).

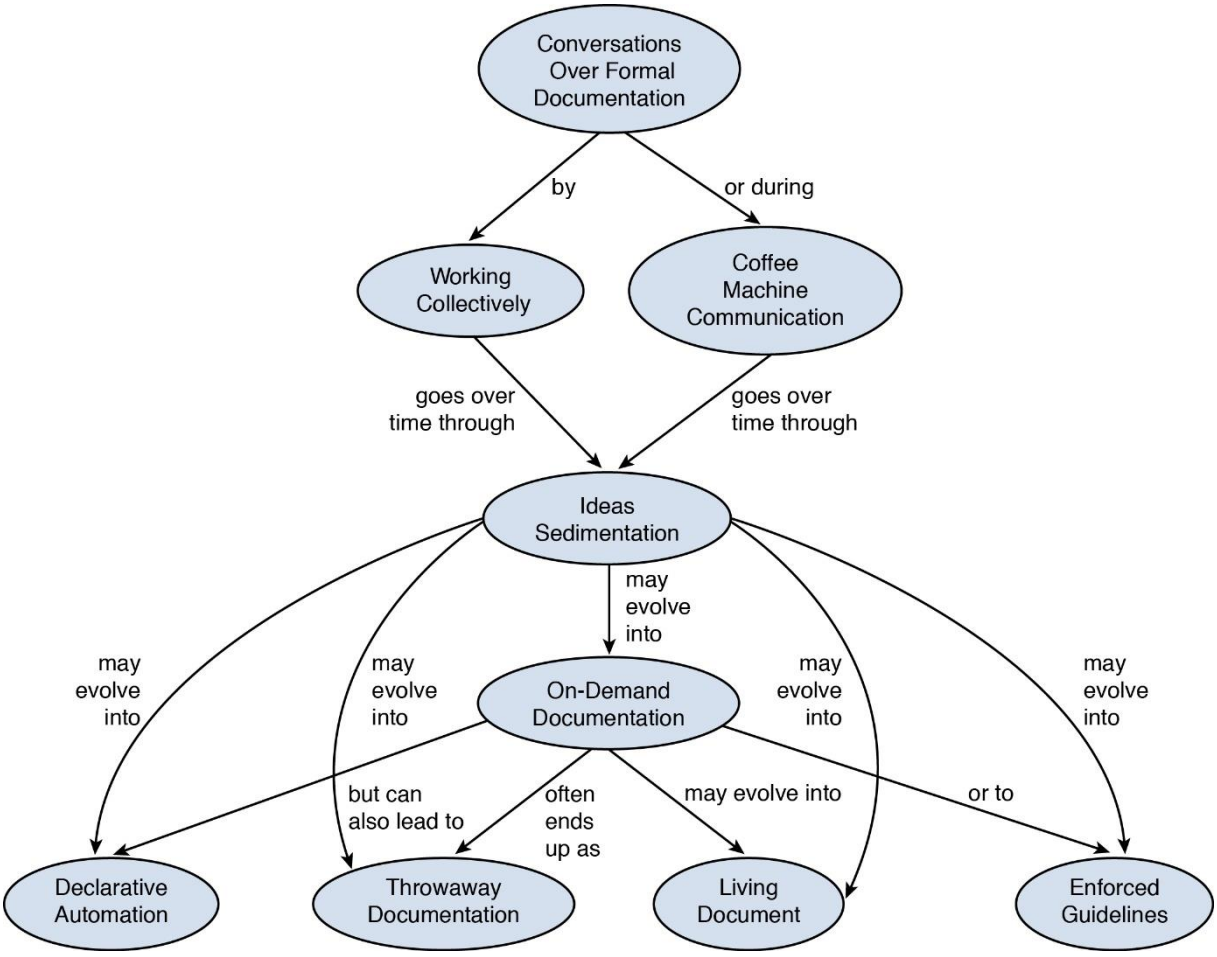

Figure 3: Patterns for avoiding traditional documentation

Beyond Documentation

Documentation has more to offer than just knowledge transfer and preservation. The advice "*Listen to the documentation*" suggests that you consider the frustrations of creating it as a signal to encourage more *deliberate decision-making*, along with the approach of being *documentation-driven* (writing the documentation before starting the construction), as represented in the diagram in Figure 4.

Skilled developers making more deliberate decisions enable a process of *embedded learning* for their workmates, who grow their skills as a result. This in turn helps them make more deliberate decisions, and this loop improves the quality of the *living design*. Using living documentation techniques and approaches also enables *hygienic transparency*, whereby everyone can see how clean the system is inside, which also gives developers feedback to improve the living design.

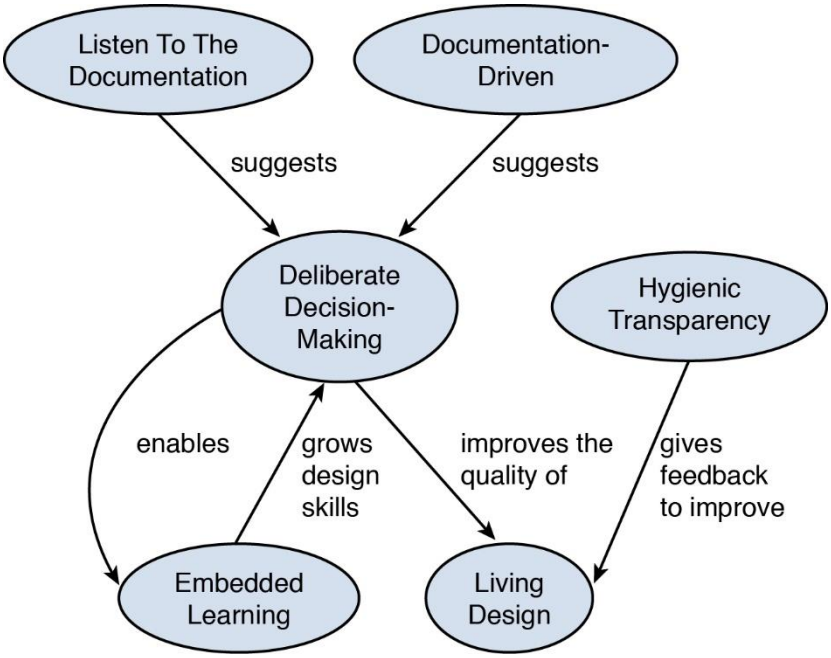

Figure 4: A pattern language of how to improve the system beyond documentation