In the last ten years, it has become common to use software design patterns to help design complex applications. Historically, design patterns are descriptive rather than generative. Each design pattern describes a family of solutions to a common software problem. A software architect uses a design pattern to communicate with programmers about an intended family of solutions to a software component. The programmers use the documentation associated with the design pattern to write code by adapting the solution family for the context of the particular application they are writing. A common example of a design pattern is the observer design pattern. In this pattern, several software objects, called observers, register themselves with another software object, called the subject. Every time the subject changes its state in a particular way, it informs the observer objects that it has changed so that they can perform whatever actions are appropriate. The observer pattern is used extensively in graphical user interfaces. In addition, it is used to describe the solution to the problem of updating a document that contains components from other documents. For example, a chart can observe a spreadsheet subject and update itself whenever the spreadsheet changes. Recently, descriptive design patterns are being augmented by generative design patterns, which in addition to describing a family of solutions to a problem, are also capable of generating code that implements any of the solutions.

Software design patterns do not model the knowledge domains of the applications they are designed to serve. Instead, they model the collaboration patterns of software objects that provide solutions to recurring software problems. For interactive storywriting, we are not using software design patterns. Instead, we are using patterns that correspond to common literary themes and idioms. Our patterns do model the knowledge domain of the discipline that our software is trying to serve (storywriting software). However, our patterns are generative so that they can generate the appropriate scripts so that the writer does not have to write them. it is a bit ironic that the idea of software design patterns arose from the use of design patterns to represent architectural solutions, not software solutions. We are simply returning to the roots of design patterns by drawing them back from the software domain to the application domain.