## Pluralsight Course Proposal Template

*Updated June 4, 2018*

*Proposing a developer course? Here’s an* [*example*](https://docs.google.com/a/pluralsight.com/document/d/1QhIQ79Tvip9iSJfk3cwtXCVKprpsitpkf_Lxd6tpjdw)*.*

*Proposing an IT admin course? Here’s your* [*example*](https://docs.google.com/a/pluralsight.com/document/d/1v6mBtHNSoIs1Dme13rSW78IjBvGfAvKlu6pJPvAhkfo)*.*

*Proposing a Creative | Design | Engineering course? Here’s your* [*example*](https://docs.google.com/a/pluralsight.com/document/d/17gZb5iWdW2frtuZjypOLjcyXoSe3Re27lnZ0YaQ8oyc/edit?usp=sharing)*.*

## *Visit*[*https://authors.pluralsight.com/proposing-course*](https://authors.pluralsight.com/proposing-course/) *for more details and tips.*

|  |  |
| --- | --- |
| **TITLE**  Max 65 characters  [Course naming conventions.](https://authors.pluralsight.com/proposing-course/course-types-dev/#course-types) | C# Design Patterns: Facade |
| **AUTHOR** | David Starr |
| **LEVEL**  Reflects the number of courses someone must watch, or pre-req knowledge they must have, before watching this one. [Beginner, Intermediate, or Advanced](https://authors.pluralsight.com/proposing-course/course-types-dev/#audience-levels) | 200 |
| **DURATION**  Estimate in hours | 0.5 |
| **AUDIENCE/INDUSTRY TAG**  Usually only 1; indicates primary audience/industry. Choose from:  software-development  data-professional  It-ops  security-professional  business-professional  architecture-construction  manufacturing-design  creative-professional | Use whichever of these are being applied to patterns.  software-development  architecture-construction |
| **AUDIENCE PROFILE**  Who should watch this course? What kinds of job roles do they hold in their organization? | Software developers wanting to improve their software design skills. |
| **ABSTRACT**  Should “sell” the course. Treat this as if it will appear on the website. Refer to [this page in the author kit](https://authors.pluralsight.com/course-deliverables/long-description/) for guidance on writing concise & informative "long" course descriptions. | Learn the Façade pattern. Software design patterns are repeatable known solutions to regularly occurring code design problems. They also provide a common language for use to use in speaking about our code.  The Façade pattern solves for situations no referenced value or with defined neutral ("null") behavior. |
| **PREREQUISITES**  Describe skills and knowledge (not specific courses) that someone should have before watching this course. | Static language experience like Java or C# and .NET Core. |
| **RELATIONSHIP TO EXISTING COURSES**  Provide links to related courses and how this one fits in, or describe the content gaps that this course fills | Sit alongside other courses used to create a single search for all design patterns. |
| **DESCRIPTION OF SAMPLE PROJECT / SCENARIO**  Describe the overarching scenario(s) that your examples and demonstrations will come from. Be specific. | The user will see code that is poorly implemented and would benefit from using this pattern. “Do you ever find yourself in this situation?” A pattern can help here, cleaning this code up and making things more testable. |
| **PLATFORM/TOOL VERSIONS**  Indicate which version(s) of platform/software/hardware you plan to **demonstrate or use** in your course, including whether or not any will still be in pre-release when you record. Note that in some cases it may be appropriate to specify multiple versions or even a range.  Additionally, please include your thoughts on how compatible the techniques or code you plan to demonstrate are with older versions of the relevant products/platforms. | |  |  |  | | --- | --- | --- | | **Technology** | **Version(s)** | **Pre-release? (Y/N)** | | C# and .NET Core | .NET Core 3 | N |   Due to the generically applicable solution in this course it is appropriate for multiple static languages and versions. including C# and Java, current and past version.  All demo will be in C# and .NET Core 3 |

**Outline Structure & Guidance**

Use the table below for your proposed outline. Please be as detailed as possible. Doing so can really help better organize your course up-front, make it easier to accurately estimate module times, and help better develop a meaningful storyline for your course. Be sure to include a time estimate for each module, which should be between 25 and 40 minutes long unless you've already discussed variations from this guideline with your editor. The estimates you provide give us insight into the breadth and depth of coverage you have in mind, so please give this some consideration.

|  |  |  |
| --- | --- | --- |
| **Module Outline** | | |
| **1** | **Course Overview**  *Do not edit or remove this module from the course outline.*  You will need to narrate a very brief informational [course overview](https://authors.pluralsight.com/course-deliverables/course-trailer/) toward the end of course production in preparation for publication.  Course overviews are the first module of your course and are accessible outside the Pluralsight paywall without a subscription. Overviews are used to market courses and provide an easy way for learners to quickly determine whether a course is right for them.  It’s a good idea to read over the course overview script now. That way you know what will and will not be covered, and can avoid accidentally repeating the same information elsewhere in the course.  No action is required during the proposal process. Your Production Editor will coordinate this with you when the time comes, and your Author Success Manager can answer any questions you may have at this time. | **60 - 90 seconds** |
|  | **Introduction**  **Overview**  **Slide**  Show a large class in UML.  “Have you ever come across a very large class? One that was so big you had to read the list of methods several times looking for what you needed?”  “The façade pattern abstracts away what we don’t need and gives us access only to those methods appropriate to our need.  **Demo – A large class and its internal dependencies**  Show a very large class in code (in an IDE). Discuss how hard it is to use. Describe the internal side effects that may be present in such a class. We don’t know what’s happening on the inside of this class when it is a black box. Even with a white box it is hard to test without side effects.  **Introduce the Façade Pattern**  Share the definition and intended use of the pattern as stated by its creators, if possible. Otherwise use my own words. Simplify the definition of needed.  Introduce the Façade pattern conceptually. First with a UML model of the pattern, then a less formal view of the pattern without UML. Show its shape and structure using a large class represented by several smaller interfaces.  Explain that smaller classes tend to adhere to separation of concerns and single responsibility better than large classes.  **Demo part 1 Applying the Façade Pattern**  Create the interface that implements only the methods you need by applying a purpose-built interface. This means using the IDE to extract a purposeful interface and have the class implement it. While we might be starting with a class of 30 methods, we only need 3 or so, therefore that’s what the extracted interface will include.  **Demo part 2**  **Use tests to refactor in the demo**  Now create a unit tests that instantiate the class but do so using the new interface. Only 3 methods to test!  Create a test class and only 3 tests for the 3 methods of the interface. Instantiate the offending class using the interface. Write incrementally passing tests against this simpler set of methods defined in the interface.  Now write a failing test by passing in a null to a method we’ve already shown the happy path on. Go into the class and add a null check. Now we’ve improved the giant class, if even a little bit.  **Slides**  Show the visual concept of having solved the problem with the pattern.  Show the model of segregating parts of the class by using special built interfaces until the code can be split apart and new smaller classes can be factored out. UML or conceptual model (haven’t decided, maybe both).  Note the interface is not strictly needed as we could have used inheritance, but interfaces provide no additional functionality, which we would likely have in the concrete child class.  **Wrapping Up**  Reminder of the pattern’s intended usage  Reminder of refactoring techniques to apply it  Reminder you may end up refactoring the original class due to internal dependencies or design choices like the null check | **20m** |