В качестве модели жизненного цикла проекта были рассмотрены следующие варианты:

1) **Build-and-fix.**

Данная модель подходит лишь для простых проектов, размер которых обычно не превышает тысячи строк.

2) **Модель быстрого прототипирования.**

**«**Основное преимущество такого подхода — в скорости: в ответ на свои требования заказчик почти сразу получает прототип интерфейса, и сразу может уточнить требования, до того как начато написание рабочего кода системы. Стоимость изменения требований на этом этапе очень низкая, поскольку нет кода, который нужно было бы переписывать» (источник: <https://ru.wikipedia.org/wiki/Прототипирование_программного_обеспечения> )

Предполагает тесное взаимодействие с пользователем/заказчиком на ранних стадиях разработки с целью уточнения спецификаций, что позволяет снизить общую стоимость проекта, а также избавиться от возможных расхождений в представлении о конечном продукте между заказчиками и разработчиками.

В нашем случае изначально есть достаточно полное представление о том, как должен выглядеть конечный продукт и какой функциональностью он должен обладать, при этом нет возможности тесно взаимодействовать с заказчиком, поэтому многократное уточнение требований только затянет разработку.

3) **Каскадная (водопадная модель).**

Преимущества:

* Последовательное выполнение этапов проекта в строгом фиксированном порядке
* Позволяет оценивать качество продукта на каждом этапе

Недостатки:

* Отсутствие обратных связей между этапами
* Не соответствует реальным условиям разработки программного продукта

(источник: <http://habrahabr.ru/post/245171/> )

Модель требует хорошего знания CASE-средств и стандартов оформления документации, организованности и дисциплинированности среди разработчиков и т.д., то есть данная модель больше подходит для команды опытных разработчиков.

4) **Инкрементальная модель.**

Особенностью модели является разбивка проекта на несколько релизов, каждый из которых дает работоспособный продукт, имеющий ограниченную функциональность (в нашем случае, например, можно разделить на три релиза: стандартное судоку с квадратными секциями, судоку с секциями произвольного вида, а в третьем релизе будет добавлена возможность играть вдвоем).

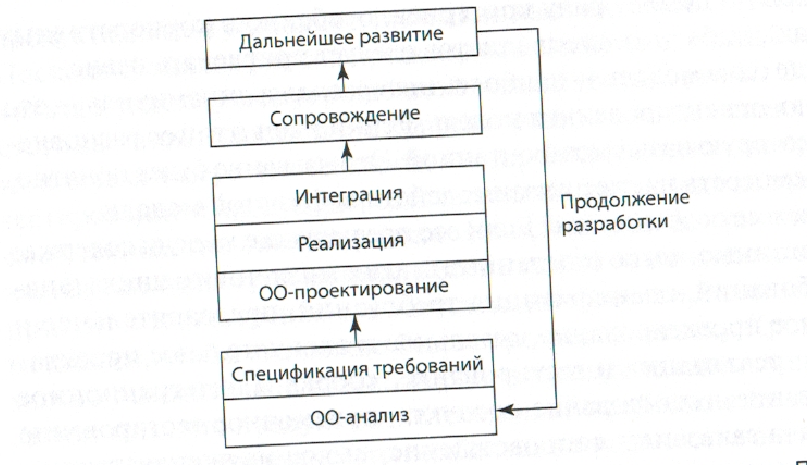
В нашем случае нет большой необходимости разделять проект на несколько релизов (что еще и требует более тесного взаимодействия с заказчиком), поэтому данная модель также нецелесообразна.

5) **Спиральная модель**

Спиральная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции.

Данная модель целесообразна, когда процесс сопровождается различными рисками (нереалистичные сроки, реализация несоответствующей функциональности, разработка неправильного пользовательского интерфейса, непрекращающийся поток изменений и т.д.) Большая часть этих рисков связана с организационными и процессными аспектами взаимодействия специалистов в проектной команде. В нашем случае проект достаточно прост для того, чтобы не иметь явных рисков, поэтому основные преимущества данной модели нивелируются.

6) **Объектно-ориентированная модель**



Данная модель довольно проста и хорошо соотносится с объектно-ориентированным подходом к программированию. Как видно из рисунка, имеет место перекрытие фаз проектирования (например, ОО-анализ перекрывает спецификацию требований).

При применении данной модели могут возникнуть проблемы в связи с большой сложностью предметной области и в связи с использованием концепции полиморфизма. Однако конкретно в нашем случае предметная область достаточно проста и можно обойтись без сложных иерархий классов (или, в предельном случае, вообще без использования наследования).

**Вердикт:** исходя из вышеизложенного, наиболее подходящей в рамках решаемой задачи является объектно-ориентированная модель жизненного цикла.