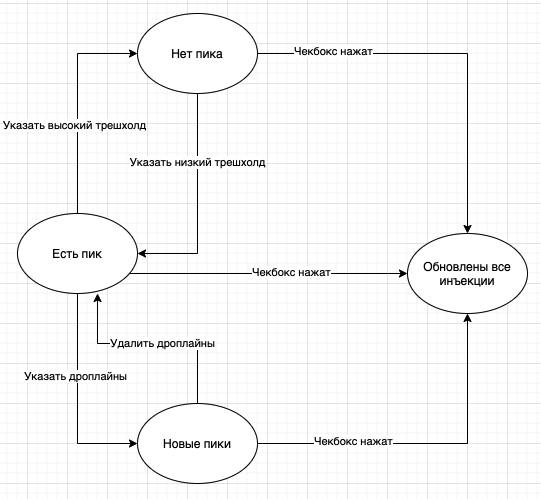
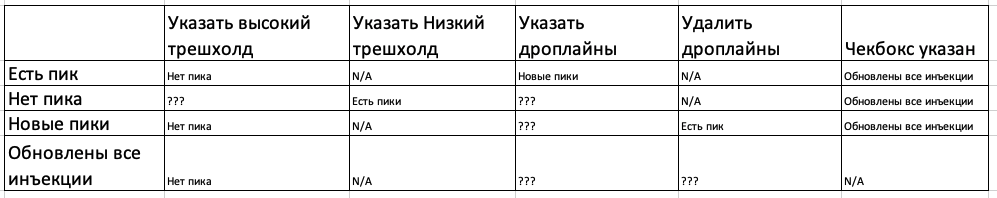
Имея на руках диаграмму состояний и переходов данной функциональности – составить таблицу переходов и протестировать все пересечения, применяя эквивалентные классы различных сущностей и элементов в каждой проверке

Здесь более верхнеуровневые переходы, нежели при тестировании ранее. Там были переходы функциональности (конкретно дроплайнов), здесь же состояния и переходы модуля.

Пики находятся в перспективе, так как Droplines от них зависят. Переходы же самих Droplines мы протестировали ранее

**Диаграмма для проверки функциональности – Droplines**

****

****

**Диаграмма состояния и переходов для приложения**

*В диаграмме состояний и переходов мы отображаем логику поведения системы*

***States Are Behavioral Modes***

* Are there things I could do now that I could not do before?
* Are there things I cannot do now that I could do before?-
* Do my actions have different results now than before?

*Whenever the questions reveal differences in behavior, new state is found*

You can also detect states by attending to the language you use to describe what’s happening. Watch for situations where you use the word *while* in describing the software, such as in these examples:

* “While the system is importing data…”
* “While the system is running the report...” • “While the account is suspended...”
* “While the call is on hold...”
* “While the server collects usage data...”

*Any time you can use the word while in describing behavior, you’ve identified a state.*

***Events Trigger State Transitions***

Remember that every state is triggered by an event. Thus, if you identify a state, back up to identify the event that triggered the state. As you are doing this analysis, you’re likely to find that you switch back and forth between looking for events and looking for states since the two things are so inextricably intertwined.

* *User actions are the most obvious kind of event*

However, user actions are just one kind of event. Other kinds of events are harder to control and thus may be harder to spot. As you’re looking for events, consider the following:

* *Externally generated events*: any event coming from outside the software. For example, if your software monitors the contents of a directory on the file system, changing the contents of that directory is an example of an externally generated event.
* *System-generated events*: any event triggered by the software itself. Often these events are the result of the system completing some background activity: loading or exporting data, connecting to a remote server, authenticating a user, or performing some background calculation. Any time you notice that there is a delay between your action and the system’s response, it is likely that there is an interstitial state and associated sys- tem-generated event hidden behind the scenes.
* *Passage of time*: some events, such as timeouts, are simply the result of the passage of time. Look for conditions that occur after a given duration, such as alerts. Also watch for events that occur at a specific time, such as a backup scheduled for midnight every night.

***Diagramming a State Model***

Creating a state model takes more than just noticing states and events. So many states and events exist even in a simple system that the task can quickly become overwhelming. Here are some strategies to keep you on track.

***Narrow Your Focus***

You can narrow the scope of your model by identifying a single target, such as a **feature** or **workflow**. Take uploading a file: the workflow involved might involve states like Selecting a File, Uploading, and Confirming Upload.

You might even choose to focus on the life cycle of an entity within the system.

**Identify a Perspective**

Consider a phone call. You could map states from the perspective of the caller. In that case, an event might include taking the phone off the hook to get a dial tone. Another event might include putting the phone back on the hook. If the caller pressed the speakerphone button before replacing the handset on the phone (another event), then the call will be in a different state than if the caller simply hung up the handset.

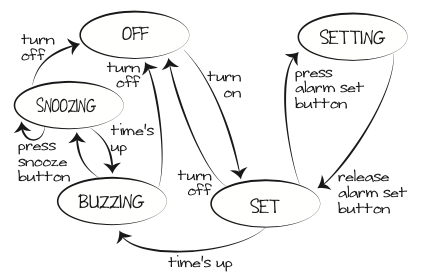
Alternatively you could map the states from the perspective of the call. The call would not come into existence until two parties are connected. Once connected, the call might be placed on hold, forwarded, or connected with a third party. The events might still involve user actions on a handset (e.g., pressing the Hold button), but the perspective dictates which states and events belong in the diagram and thus narrows the scope.

**Dial Up or Down the Level of Abstraction**

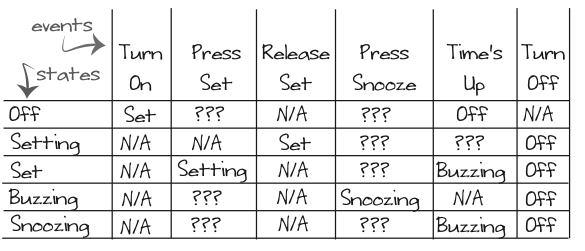
If you are modeling a simple small interaction, you can do so at a low level of detail. You can map out all the teeny transitory states between user-generated events. This will give you tremendous insight into those pernicious, narrow windows of vulnerability.

However, you’ll need to work at a higher level of abstraction if your target is something bigger. Instead of listing out numerous separate transitory states, you’ll lump them together into one big state with a less specific name.

***State Tables***

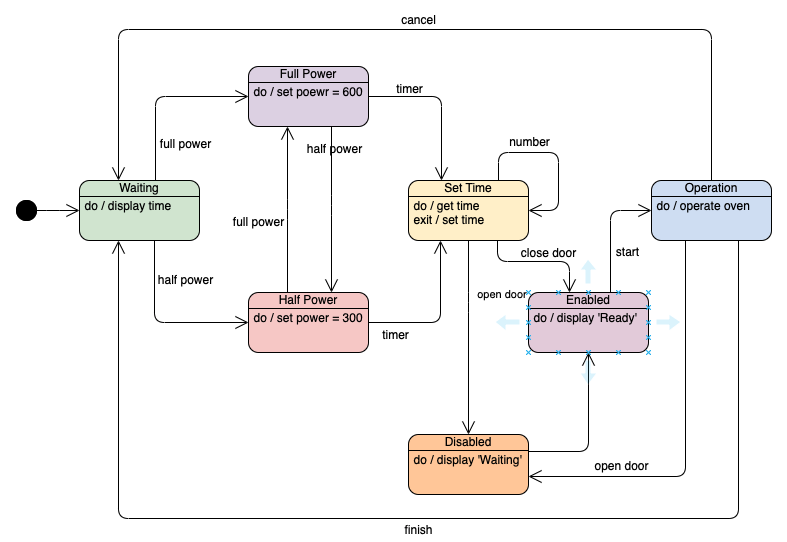


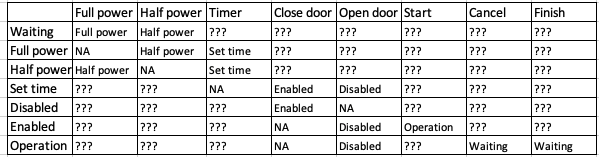
States + Events = new State



What happens if we’re setting the alarm and the clock matches the current set alarm time? Does the alarm start buzzing? Should it? These are the cells where we don’t know what will, or what should, happen. They contain “???” in the table. These possible but unexplored transitions are ripe for discovering surprises.

**OVEN**

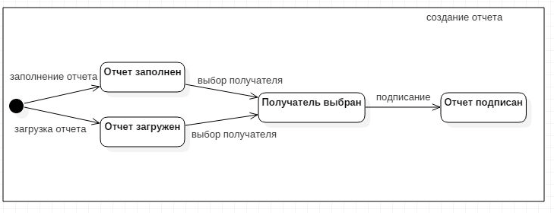




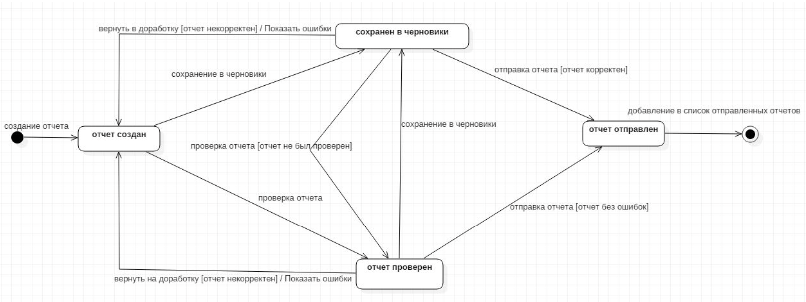
**Создание диаграммы для сайта с отчетами**

****

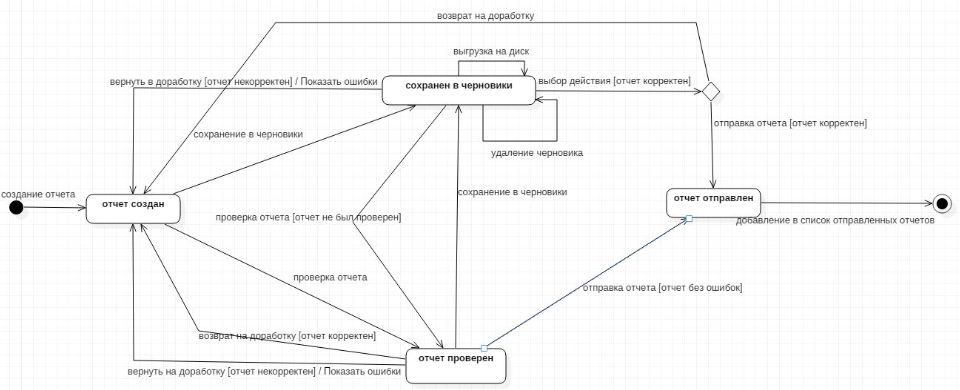
Далее обратим внимание на состояние «отчёт создан». Здесь возникают вопросы: каким образом создаётся отчёт? Наверняка создать отчёт невозможно нажатием одной кнопки. Это значит, что должны быть этапы создания отчёта. Каким образом вводится отчёт? Как узнать, кому отчёт должен быть отправлен? Должен ли он быть подписан перед отправкой?

****

Теперь вернёмся к основной модели и подумаем, какие ещё возможности мы могли бы добавить? Например, что, если пользователь после создания или проверки отчёта не захочет его сразу отправлять? Например, в отчёте найдены ошибки, и пользователь в данный момент не знает, как их исправить. Или же он заполнял отчёт, но не заполнил до конца и хочет отложить заполнение на потом. Для таких отчётов должна быть возможность сохранения в черновик. Добавим состояние «сохранён в черновики» на нашу основную диаграмму. Какие переходы могут быть в это состояние? Созданный отчёт можно сохранить в черновик, проверенный отчёт тоже можно сохранить в черновик. Также из черновиков можно отправить отчёт, если он был проверен и ошибок в нём не было, если же отчёт содержал ошибки, то мы можем вернуться на стадию создания отчёта. Добавив состояние «сохранён в черновики» и все перечисленные переходы из и в это состояние, получим такую схему:

****

Далее подумаем, какие ещё функции мы могли пропустить. Например, обратим внимание на состояние «отчёт проверен». Из этого состояния есть переход на доработку отчёта только в том случае, когда в отчёте были найдены ошибки. Однако ​**пользователю может понадобиться вернуться к доработке** даже тогда, когда отчёт корректен с точки зрения правил формирования отчётов подобного типа. Например, пользователь забыл добавить какую-то информацию в отчёт. Но в нашей схеме не предусмотрена такая возможность. Поэтому нужно её добавить. Аналогично и для случая, когда отчёт находится в черновиках. Теперь, если отчёт корректен, пользователь сможет выбрать, что он хочет сделать с сохранённым отчётом – отправить или доработать.

****