Improving Reliability

Typestate Pattern

Typestates

- Leverage type system to encode state changes
- Implemented by creating a type for each state
    - Use *move* semantics to invalidate a state
    - Return next state from previous state
    - Optionally drop the state
        - Close file, connection dropped, etc
- *Compile time* enforcement of logic

Example

```rust
struct BusTicket;
struct BoardedBusTicket;

impl BusTicket {
    fn board(self) -> BoardedBusTicket {
        BoardedBusTicket
    }
}

let ticket = BusTicket;
let boarded = ticket.board();

// Compile error
ticket.board();
```

Example

```rust
struct File<'a>(&'a str);
impl<'a> File<'a> {
    fn read_bytes(&self) -> Vec<u8> {
        // ... read data ...
    }
    fn delete(self) {
        // ... delete file ...
    }
}

let file = File("data.txt");
let data = file.read_bytes();
file.delete();

// Compile error
let read_again = file.read_bytes();
```

Recap

- Typestates leverage the compiler to enforce logic
- Can be used for:
  - Invalidating / consuming states
  - Properly transitioning to another state
  - Disallowing access to a missing resource