Agile Design Principles:
Single Responsibility Principle

Based on Chapter 8 of Robert C. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall, 2003.

SRP: Single Responsibility Principle

- "A class should only have one reason to change"
- Consequently most changes will affect a small proportion of classes.
- This is closely related to Information Hiding which says that each axis of change should be isolated in a small set of modules.

Responsibility = Reason to Change

- Martin defines a *responsibility* to be "a reason to change", in the context of the SRP.
- I don't particularly like this definition. I prefer responsibilities to be stated in terms of the services classes provide to clients. E.g.
  - *The Sorter class may be used to sort an array*.
- Implicit in this responsibility are several potential axes of change.
  - *What type of data is to be sorted?*
  - *What array is to be sorted?*
  - *What order is to be used?*
  - *Is the sort stable?*
  - *What algorithm is used?*
- You can think of SRP as the "Single Reason to Change Principle"

Refactoring

- “Refactoring” means improving the internal design without changing the external behaviour
- When a change is required that was not anticipated, we should identify a new axis of change
- When a new axis of change is discovered we should “refactor” first and then change
- Poor software engineers make software more brittle when they change it
- Good software engineers improve the flexibility of software when they change it

Don't over design.

- We can (and should) anticipate likely reasons to change.
- But: We should not make them up.
- There is no point protecting the design against classes change that are likely not to occur.

Don't Underdesign

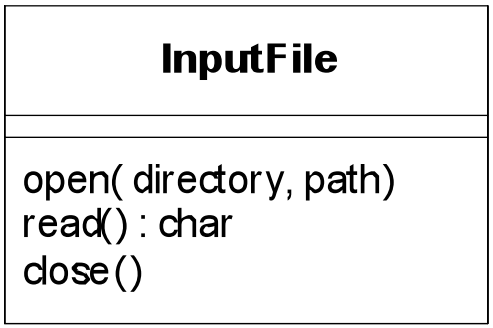
- One principle of Agile design is "Do the simplest thing that could possibly work." (From Beck and Cunningham.)
- However the simplest thing is often unnecessarily brittle.
- It is very good to ask the question: "What's the simplest thing that could possibly work", but you should avoid building in brittleness.

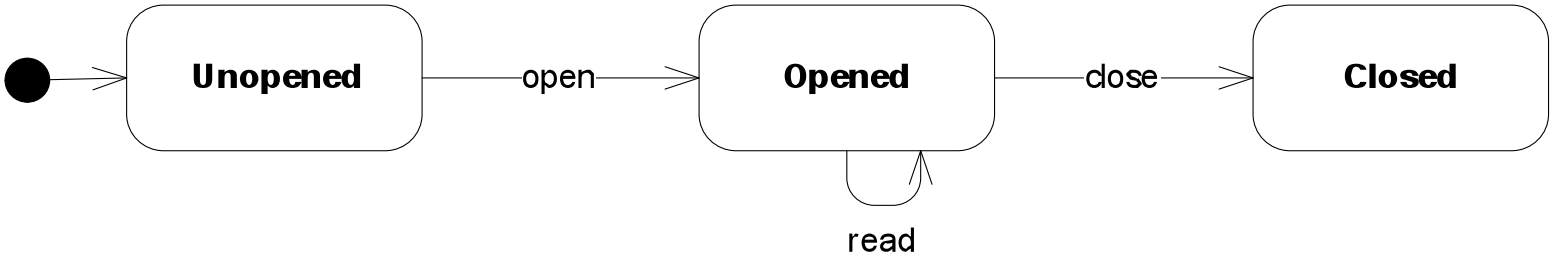
The SRP in action

- Consider this problem. We need to process some information that comes from a local file.
- We need to be able to open files, close files, read from files.

```
InputFile f = new InputFile() ;
f.open( directoryPath, relativePath ) ;
process(f) ;
f.close() ;
```

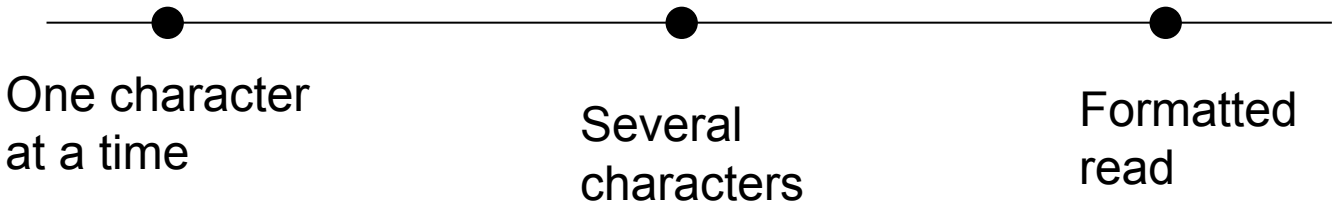
SRP in Action





The SRP in action

- What might change
  - How files are read: Do we read one character at a time or can we do formatted input.



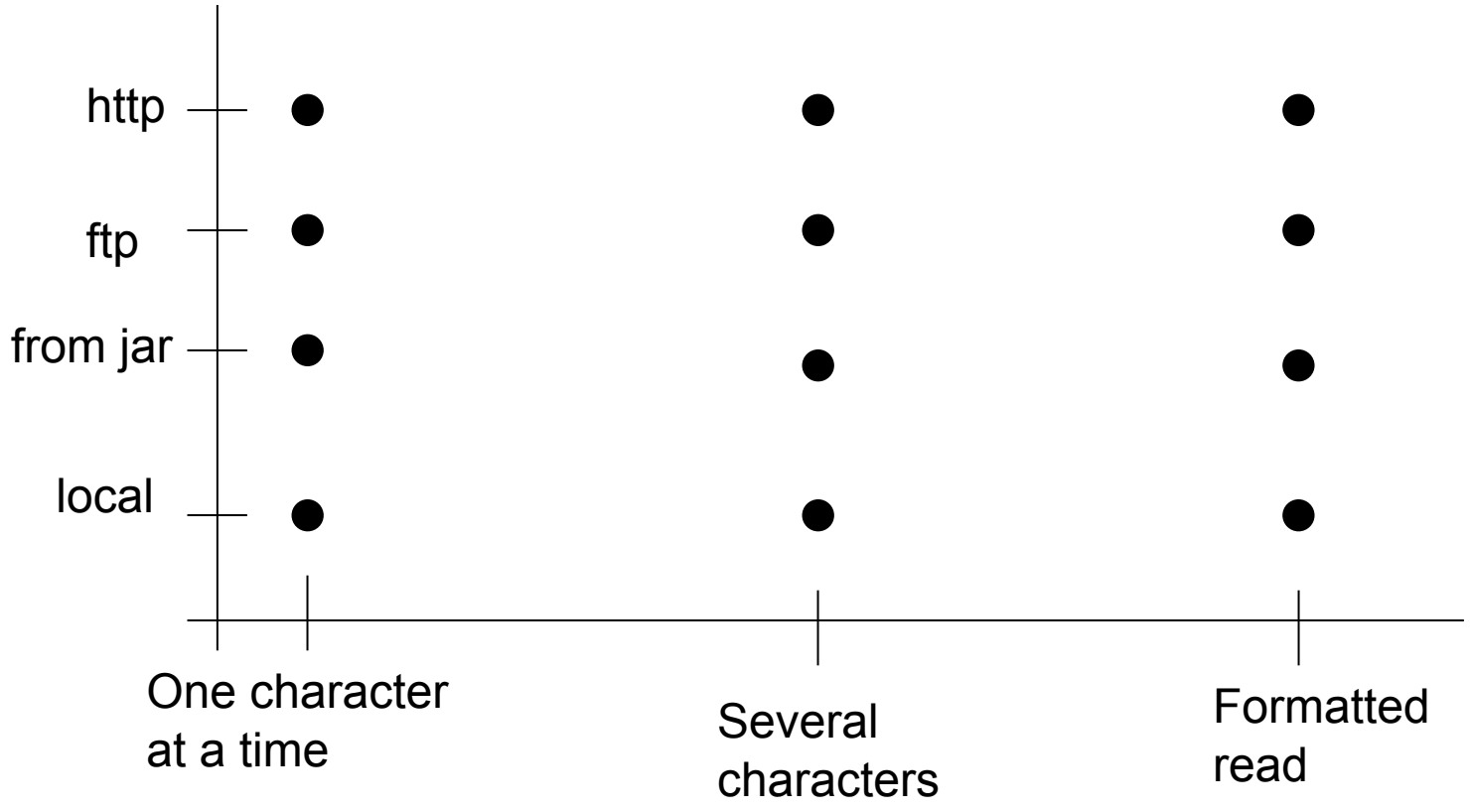
A Change

- All is well until the customer says:

  We need to be able to process data from the web.
- This uncovers a new axis of change: Where the file comes from
  - Is it local, accessed by http, read by ftp, or …

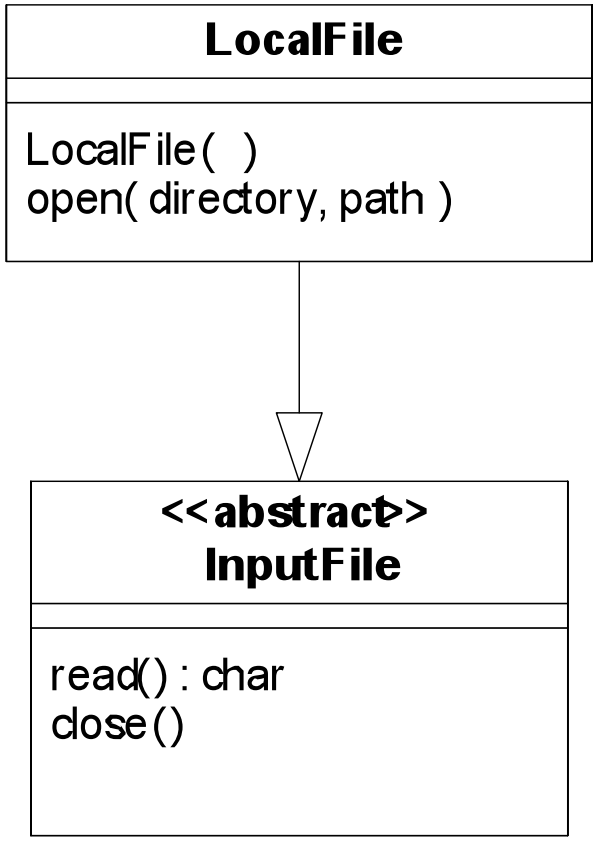
SRP in Action

- This axis is orthogonal to how the file is read



SRP in action

- We should refactor so that this unanticipated change becomes an *anticipated change.*



SRP in action

- Now add the new functionality
- Now each class represents commitment to a point on a single axis of change