Test Driven Development At Labman

Arthur Clarkson

20th September 2024

At Labman, one of our policies for developing methods with hundreds of code path permutations is to use Test Driven Development (TDD).

To use TDD you write a load of unit test cases before you start development, based off the specification. This allows the developer to run the tests during development, allowing for quick bug detection, and quick constant feedback for whether their algorithm equates for the whole specification.

On the intranet we needed a reminder system, which would send out persistent notifications, helping us maintain our business goal of timeliness. In the specification a user could set-up a schedule, which would create a reminder for every event in the schedule.

A schedule has a lot of variability to it and can be assigned: `every x days, every x weeks on y day, every x months on the yth week on z day, every x years on y month... etc. etc.`

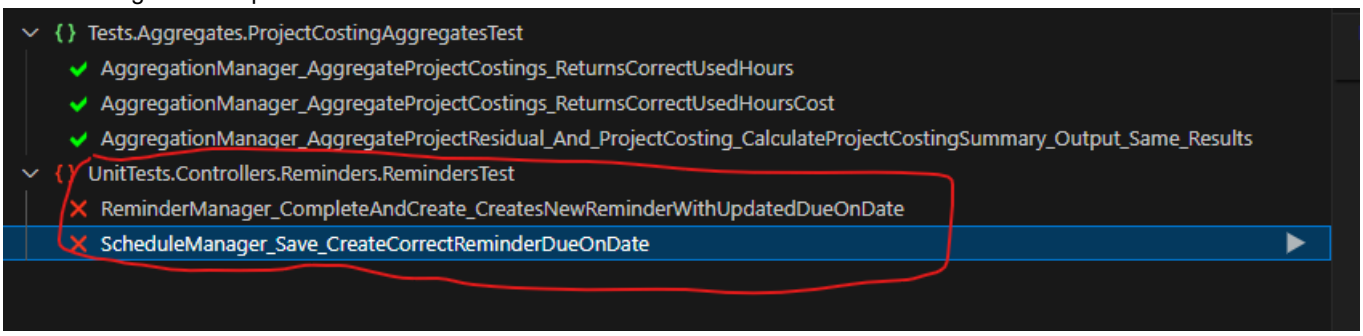
To develop the method which would calculate the next reminder date would be impossible to test without some sort of automated testing due to there being thousands of schedule permutations. With this knowledge, I decided to use TDD to create this method.

With us needing a lot of tests, the best way to manage this, was to create an excel file, read and parse the data to a TestCase object, and then run the TestCase data as arguments in the `CalcNextReminderDate` method. This is a snippet from the excel file:

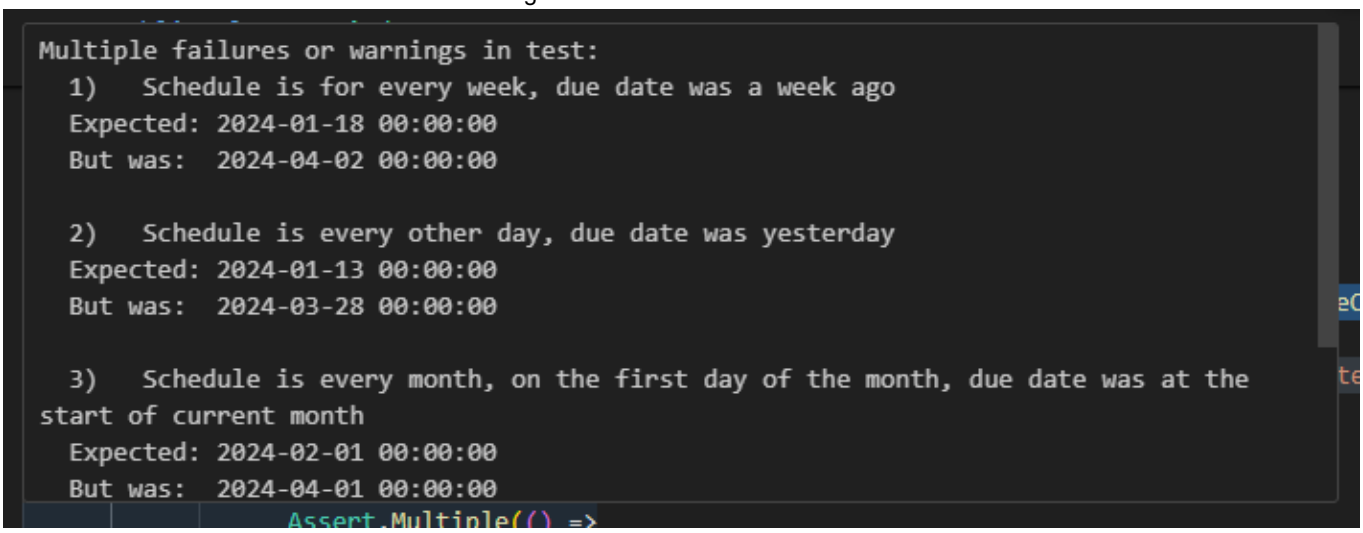
	A	B	C	D	E	F	G	H	I	J	K
1	Scenario	ExpectedOutput	DueOn	RecurrenceType	RecurrenceDayOfWee	RecurrenceDayOfMont	RecurrenceInterva	RecurrenceMonthOfYea	CreatedOn	RemindOn	RangeStartDate
2	Schedule is every week due on 31/12/209	07/01/2100	31/12/2099	Weeks	Day	None	1		31/12/2099	31/12/2099	31/12/2099
3	Reminder is due on 01/01/2100, schedule	01/01/2102	01/01/2100	Years	Day	First	2	1	01/01/2100	01/01/2100	01/01/2100
4	Reminder is due on 01/01/2100, schedule	01/01/2101	01/01/2100	Years	Day	First	1	1	01/01/2100	01/01/2100	01/01/2100
5	Schedule that runs every other year on th	01/01/2102	01/01/2100	Years	Day	First	2	1	01/01/2100	01/01/2100	01/01/2100
6	Schedule that runs every year on the first	01/01/2101	01/01/2100	Years	Day	First	1	1	01/01/2100	01/01/2100	01/01/2100
7	Reminder is due on 01/01/2100, schedule	08/01/2100	01/01/2100	Weeks	Friday	None	1		01/01/2100	01/01/2100	01/01/2100
8	Reminder is due on 01/01/2100, schedule	03/01/2100	01/01/2100	Days	Day	None	2		01/01/2100	01/01/2100	01/01/2100

To TDD, I would get the algorithm to a point of which I was happy with, then I'd run the tests, for any failed cases, I'd debug to see why they're failing, fix the issue, then re-run the tests. This process repeated until all tests passed.

The following are examples of failed cases:



I'd made it easier and wrote some custom messages to see what data failed the tests:



In conclusion, using Test Driven Development for our reminder system made a big difference. By writing the test cases before coding, we managed the complex scheduling options much more smoothly. The Excel file helped us quickly spot and fix issues. In the end, TDD helped us build a reliable system with accurate schedule dates.