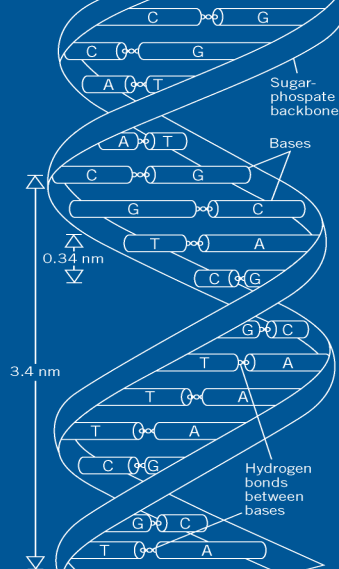
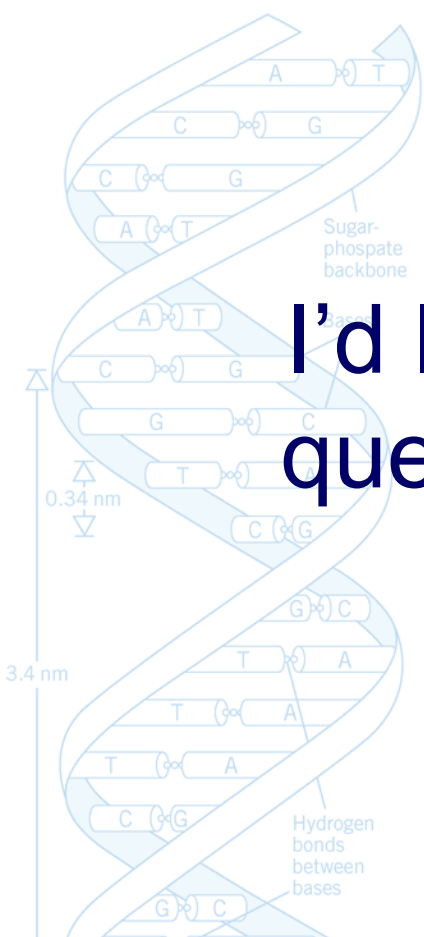




A Case Study in Using Unit Testing as a Method of Primary Validation

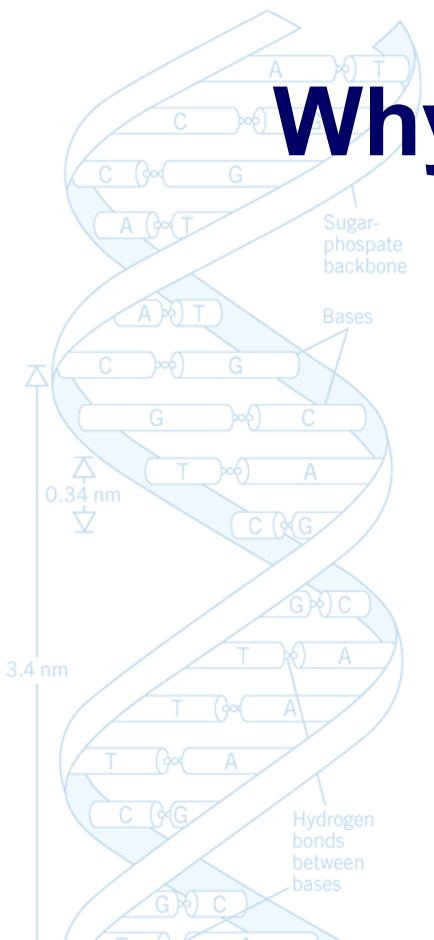
Ross Farrugia, Roche Products Ltd



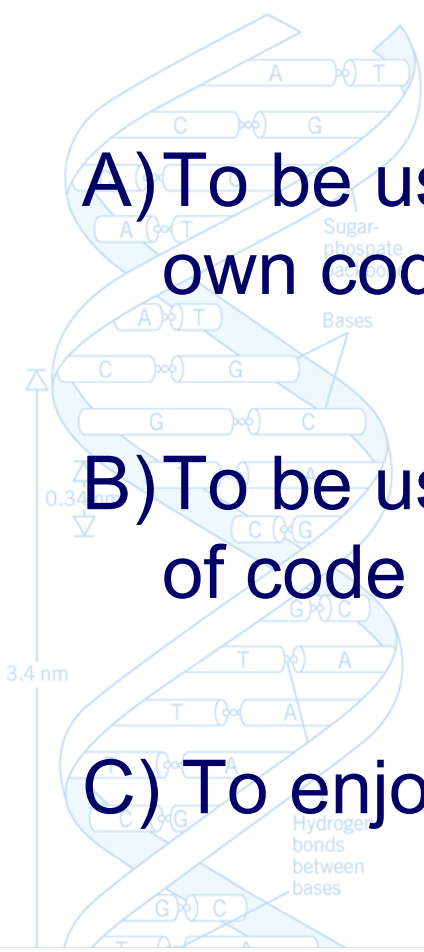
A diagram of a DNA double helix structure. The two strands are represented by thick blue ribbons that spiral around each other. Between the strands are horizontal bars representing nitrogenous bases, labeled with their initials: A (Adenine), T (Thymine), C (Cytosine), and G (Guanine). The bases are connected to the sugar-phosphate backbone by small circles. Labels include 'Sugar-phosphate backbone' pointing to the ribbons and 'Hydrogen bonds between bases' pointing to the circles. Two vertical dimension lines are shown: one on the left indicating the width of the helix as 3.4 nm, and another on the right indicating the distance between two base pairs as 0.34 nm.

I'd like to start with the age-old question many of you will have pondered previously...

Why did the unit test program cross the road???



What does the audience think??



A) To be used by a developer to informally test their own code

B) To be used by a peer to formally validate a piece of code

C) To enjoy the Berlin nightlife

A Definition of Testing

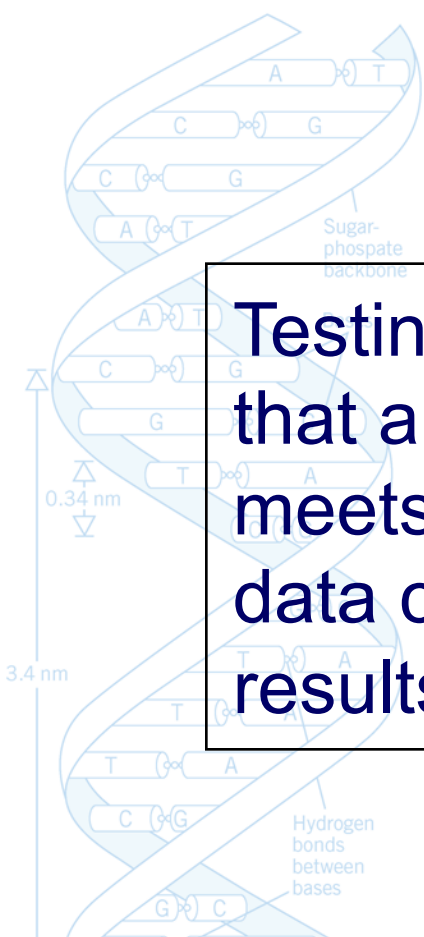
Testing is the process of establishing confidence that a program does what it is supposed to do.

William Hetzel, 1973

Testing is the process of executing a program with the intent of finding errors.

Glen Myers, 1979

A Definition of Unit Testing



Testing where a programmer proves that a module of code (the “unit”) meets its requirements using test data cases, and comparing expected results versus actual

Ross Farrugia, 2010



Warning on format of slides!

The 2 best things Phuse has to offer (in my opinion)

...

Knowledge Sharing – Slides Content
Social Networking – Slides Format

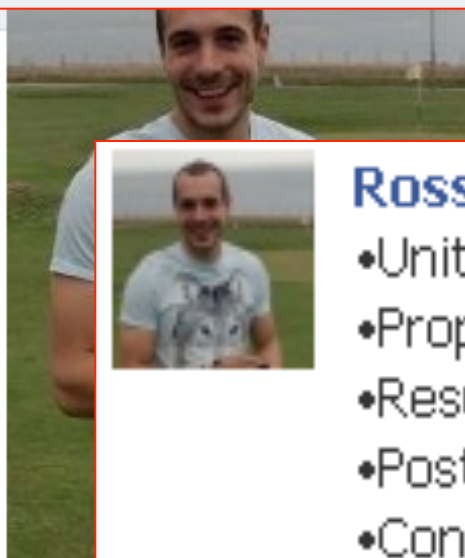
facebook



Search



Ross Farrugia is presenting the Contents of his presentation a moment ago [clear](#)



What's on your mind?

- Ross Farrugia**
- Brief overview of Unit Testing
 - Unit Testing Implementation – with a simple example
 - Proposed and Actual Usage on our Pilot Study
 - Results of the Pilot Study
 - Post-Study Findings
 - Conclusions – Benefits and Considerations

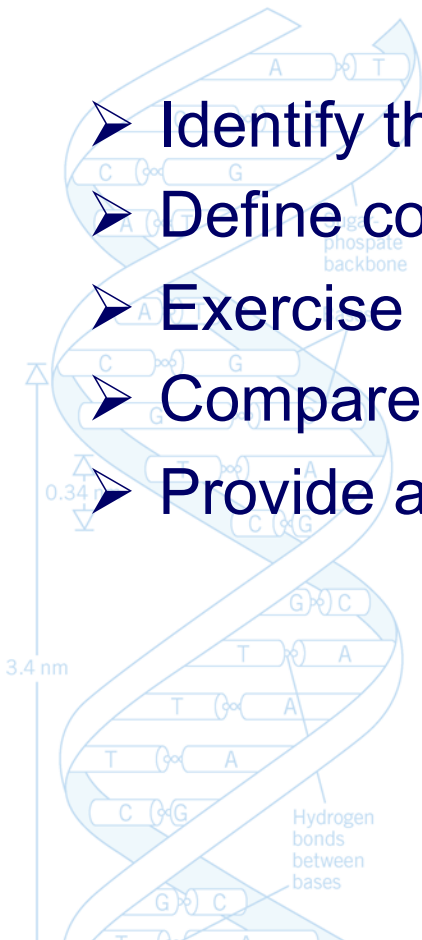
a few seconds ago · [Comment](#) · [Like](#)

[Share](#)[Options](#)[Remove](#)[Remove](#)[View P](#)[View V](#)[Edit My Profile](#)[Conclusions – Benefits and Considerations](#)

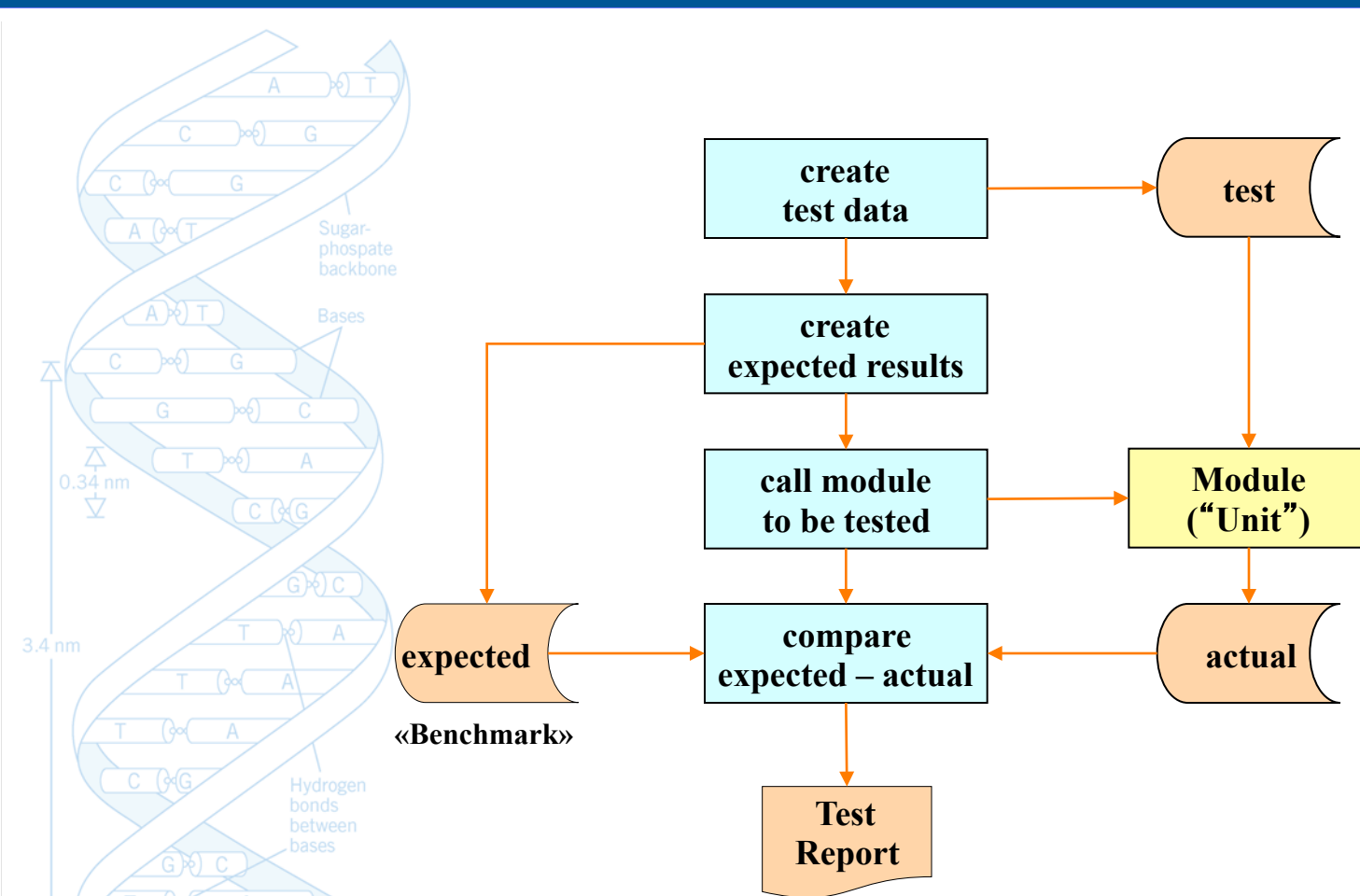
a few seconds ago · [Comment](#) · [Like](#)

What is a Test?

- Identify the feature to be tested
- Define controlled inputs
- Exercise the feature with the controlled inputs
- Compare the result with the expected behavior
- Provide a test and incidence report (evidence)

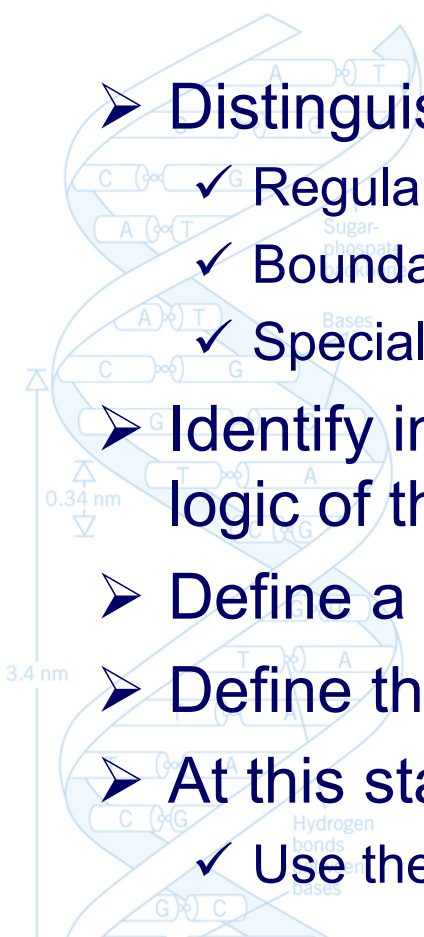


How does this apply in Unit Testing?



Implementation – Unit Test Design



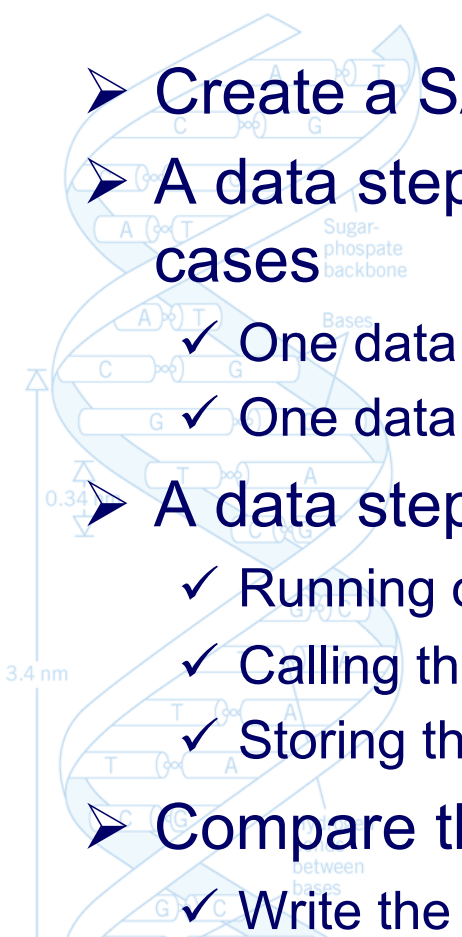
- 
- A faint, stylized background image of a DNA double helix. It shows the sugar-phosphate backbone and the nitrogenous bases (A, T, C, G) connected by hydrogen bonds. Dimensions like 0.34 nm and 3.4 nm are indicated, along with labels for "Sugar-phosphate", "Bases", and "Hydrogen bonds".
- Distinguish the following categories for test data cases
 - ✓ Regular Cases
 - ✓ Boundary Cases
 - ✓ Special Cases
 - Identify individual features and scenarios based on the logic of the requirements
 - Define a set of test cases to cover each feature
 - Define the expected result for each test case
 - At this stage you may hit upon more subtle issues
 - ✓ Use the particular test case to get clarification of the specs



ID	Values Category	Test Case	Independent Variables		Result
			refdt	eventdt	sday
01	Regular	refdt before eventdt	15JAN2006	25JAN2006	11
02			15JAN2006	10JUN2006	147
03			15JAN2006	12APR2008	819
04		eventdt before refdt	25JAN2006	15JAN2006	-10
05			10JUN2006	15JAN2006	-146
06			12APR2008	15JAN2006	-818
07	Special	character	"ref"	25JAN2006	missing
08			15JAN2006	"event"	missing
09			"ref"	"event"	missing
10		missing	missing	25JAN2006	missing
11			15JAN2006	missing	missing
12			missing	missing	missing
13	Boundary	refdt just before eventdt	15JAN2006	16JAN2006	2
14			15JAN2006	17JAN2006	3
15			31DEC2005	01JAN2006	2
16		eventdt just before refdt	16JAN2006	15JAN2006	-1
17			17JAN2006	15JAN2006	-2
18			01JAN2006	31DEC2005	-1
19		eventdt = refdt	01JAN2006	01JAN2006	1

Implementation

– Unit Test Program Execution

- 
- Create a SAS program containing the following steps
 - A data step reading data lines containing the test cases
 - ✓ One data set `testcases` with the independent variables
 - ✓ One data set `expected` with the expected results
 - A data step
 - ✓ Running over `testcases`
 - ✓ Calling the macro for each test case
 - ✓ Storing the result in a data set `actual`
 - Compare the `expected` and `actual` data sets
 - ✓ Write the unit test report to an output file for proof of formal testing



Ross tweets:

Example – Study Day Calculation Macro

```

*** Read test cases ***;

data testcases;
  infile datalines dsd;
  input testcase refdt:datetime20. eventdt:datetime20.;
datalines;
1,15JAN2006:00:00:00,25JAN2006:00:00:00
2,15JAN2006:00:00:00,10JAN2006:00:00:00
...etc
;
run;

*** Prepare expected results ***;

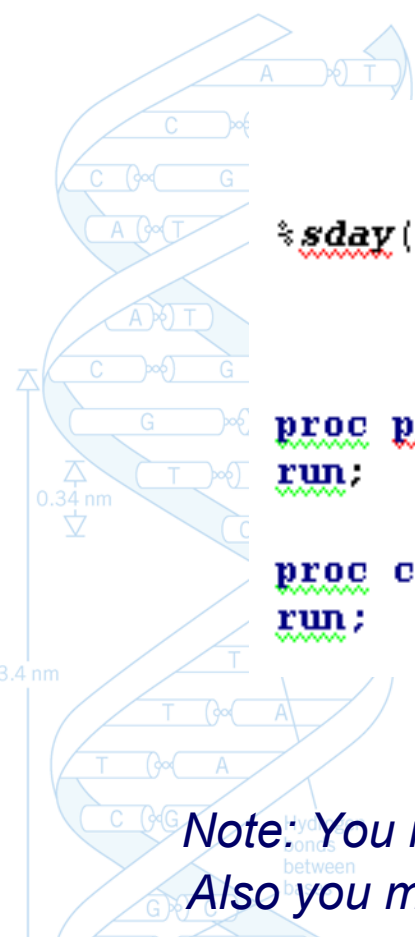
data expected.sday;
  infile datalines dsd;
  input testcase sday;
datalines;
1,11
2,147
...etc
;
run;

```

Note: sometimes you may have multiple input datasets, or more than one separate expected outcome dataset



Ross tweets: Example – Study Day Calculation Macro



```

*** Execute Study Day Macro using test cases ***;

%sday(source=testcases, target=actual.sday);

*** Test Actual versus Expected Results ***;

proc printto file=sday_test.report;
run;

proc compare base=expected.sday compare=actual.sday;
run;

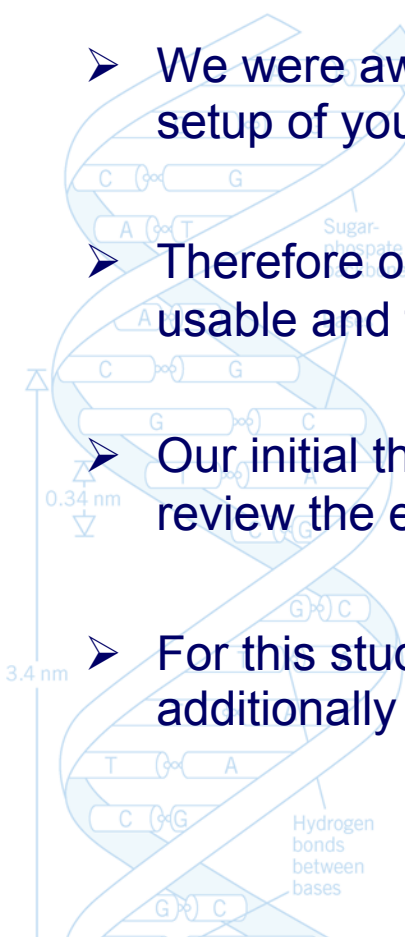
```

Note: You may require repeats of this to test different macro flexibilities. Also you may need different test data cases for these, but for efficiency purposes this should be avoided.

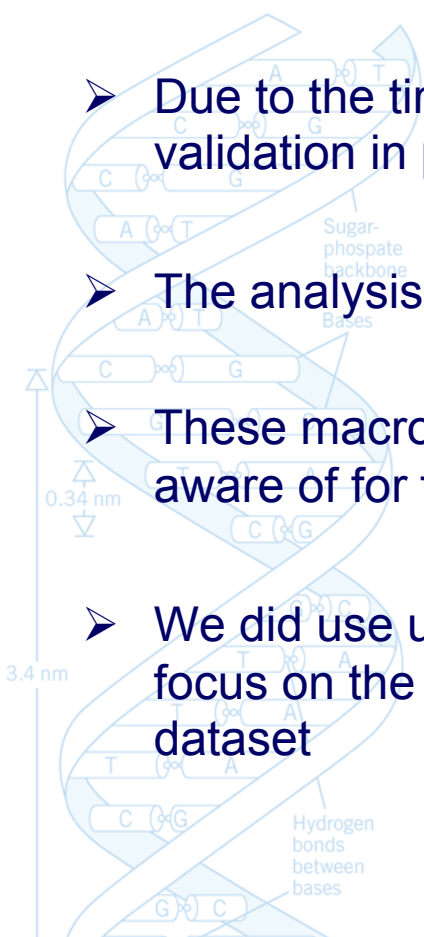
Proposed Usage on our Pilot Study

- Commonly used solely as a method of developer testing
- We proposed to use as a formal method of high risk level QC, as an alternative option to double programming
- An oncology Phase II study was chosen as a pilot
- Decision was taken for the pilot to still double program anything PQC'ed using unit testing

Proposed Usage (continued)

- 
- We were aware unit testing works best with a modular approach taken to the setup of your programs
 - Therefore our analysis dataset programs were split into a number of reusable and flexible macro modules, usually one per derivation
 - Our initial thoughts were to unit test each of these macros and then just code review the entire analysis dataset program
 - For this study we took a number of these macros to unit test, and then additionally double programmed the analysis dataset

Actual Usage

- 
- Due to the time constraints of having to essentially perform two methods of validation in parallel, we opted to pilot this on just one key analysis dataset
 - The analysis dataset was broken up into 12 individual macros
 - These macros were given project-level flexibility for requirements we were aware of for future studies (and using learnings from past studies)
 - We did use unit testing for a few other project-level macros also but we'll focus on the results from it's major use which was on this key analysis dataset

- Unit testing did find some errors in code which double programming missed.
- Why??
 - ✓ some of the possible data scenarios had yet to occur in our clinical data
 - ✓ some of the macro flexibilities hadn't been used on this study
- Unit Testing gave confidence that our programs were worthy of validated status for all macro calls and all data scenarios not just those seen at that time



Results (continued)

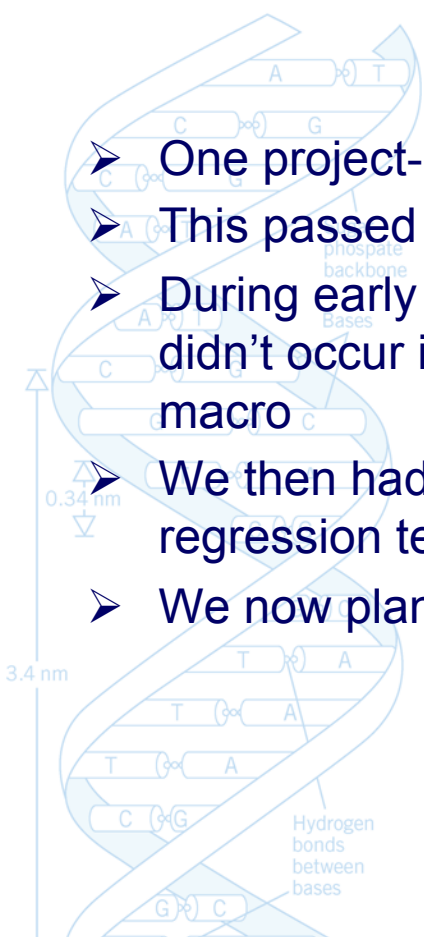
People ▼

Search

POWERED BY
Google

- Programmers became more familiar with the requirements and data, and this lead to much clearer specifications
- Unit testing initially takes longer as you have to become comfortable with this new skill, but as familiarity grows the time cost is no different to double programming
- With a modular approach further time saving will be found with the re-use of the macros on future studies and not needing to update for later data
- Unit testing only tests the cases you put in, so won't cover data scenarios you haven't anticipated!

Post-Study Findings

- 
- A diagram of a DNA double helix is positioned on the left side of the slide. It shows two strands twisted around each other, with nitrogenous bases (A, T, C, G) connected by hydrogen bonds. Labels include 'phosphate backbone' for the outer strands, 'bases' for the inner rungs, and 'Hydrogen bonds between bases' for the connections. Two vertical scale markers are present: one indicating 0.34 nm for the distance between adjacent base pairs, and another indicating 3.4 nm for the length of one full helical turn.
- One project-level macro not unit tested was a partial date imputation macro
 - This passed QC and worked fine for our pilot study
 - During early reporting of our next study we found a new data issue that didn't occur in our pilot study, and this has lead to an error spotted in the macro
 - We then had to update the macro and perform backward compatibility/ regression testing for its usage in our pilot study
 - We now plan to unit test the macro to ensure it is correct

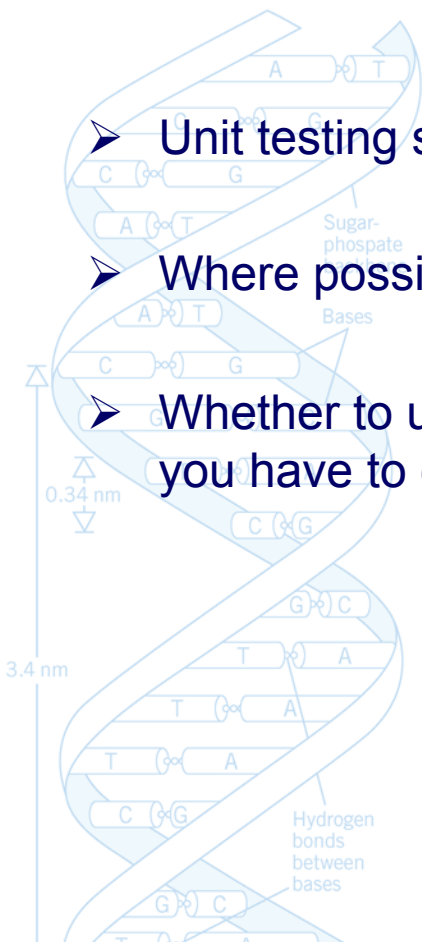
Conclusion - Considerations

Unit testing can be a good alternative option to double programming, but there are some considerations:-

- Modular programming approach is most suited
- To unit test a whole analysis dataset you'd then need some form of integration testing
- Unit testing relies on the quality of your test cases
- A data quality checking tool (or data quality reports from DM) are helpful to be aware of any invalid data scenarios occurring

Conclusion - Considerations (continued)

- Unit testing should be done on stable requirements
- Where possible limit macro flexibilities to only what could ever be needed
- Whether to use unit testing does depend on the macro module in question, you have to choose where appropriate

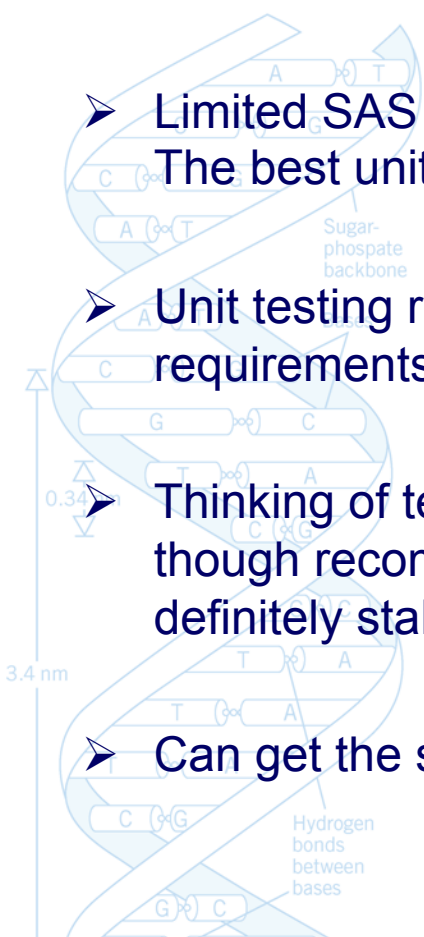


Conclusion - Benefits

Here are some of the benefits we saw:-

- Unit testing helped us identify issues in a program early that hadn't been found by double programming yet, because the data scenarios had yet to occur in the data
- Unit tests don't need to be re-ran every time we have new data. Once your program is validated it stays validated
- Macro modules full flexibility was tested early to reduce future updates to these programs when re-used on new studies

Conclusion – Benefits (continued)

- 
- Limited SAS knowledge needed to QC even the most complex of derivations. The best unit test programs are simple unit test programs
 - Unit testing requires the QC'er to really question and understand the requirements and input data, which can improve quality of specifications
 - Thinking of test data cases can start in parallel with first line programming, though recommendation to leave expected outcomes until specifications are definitely stable
 - Can get the statistician involved in reviewing our test data cases

I'd like to give thanks to...



- Frederik Malfait and Ryan Copping (for review of my paper)

- Cheryl Kubsch (for re-use of some training slides in this presentation)

[Join](#)

- Hinal Patel and Francis Kendall (for allowing and encouraging this pilot to go ahead)



AnyQuestions

Macro Modular QC BoundaryCases IntegrationTesting Implementation UnitTesting RegularCases TestCases