Chapter 19

Test-Driven Development

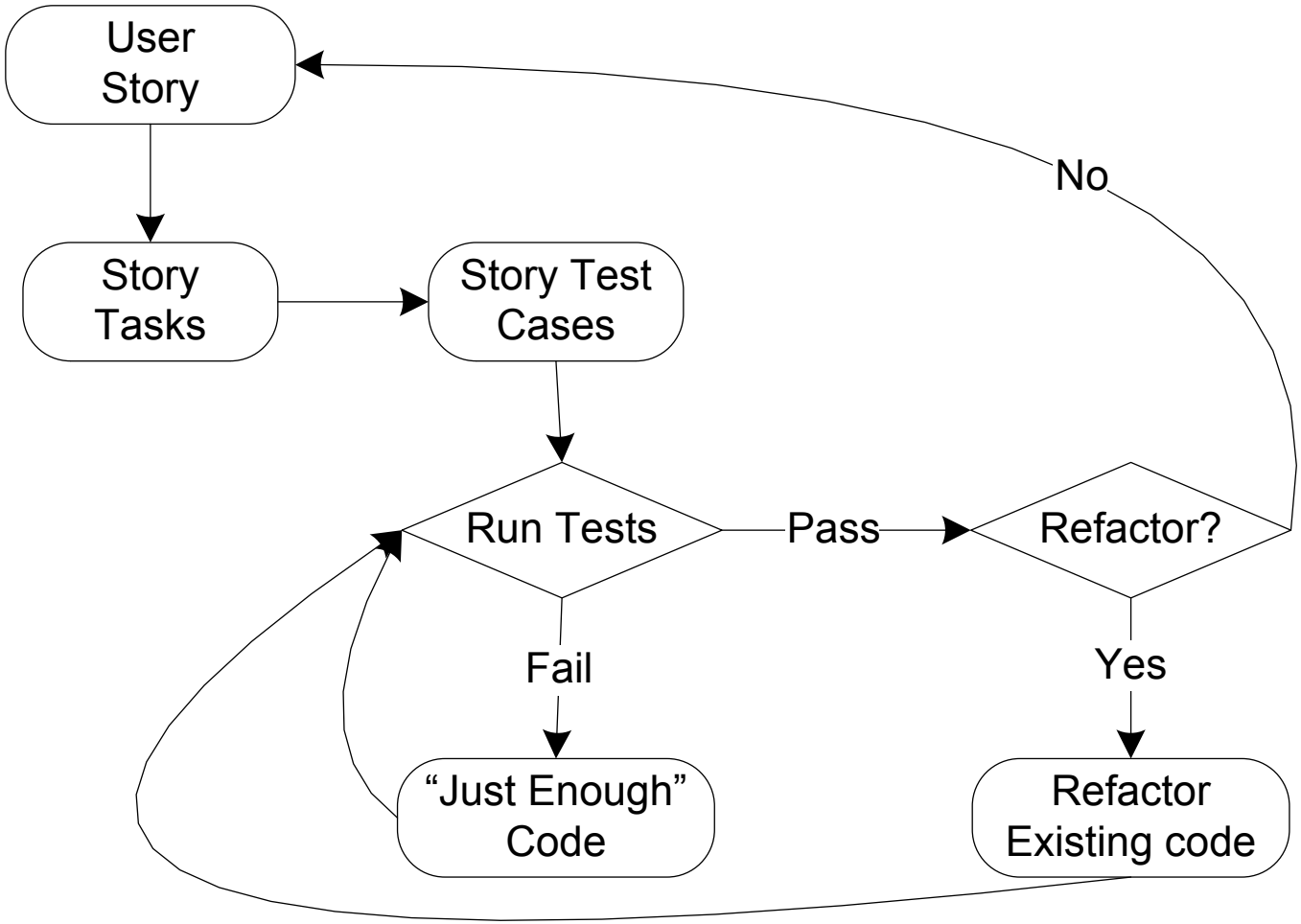
Test-Driven Development

- The essence of agility "Test, then Code cycles"
- Also known as Test-First Development
- Requires automation
- A bottom-up process
- Presumes refactoring
- An automated test execution framework is a MUST
- Excellent for fault isolation
- Test "granularity" is an issue
- Big Question: does TDD result in a good design?

Steps in Test-Driven Development

- TDD begins with a (customer provided) User Story
  - possibly broken into tasks
  - test cases developed for each task
  - (no code is written yet)
- Run the test cases; they fail
- Write just enough code to implement the task
- Run the test cases again
  - if they fail, something is wrong, but the fault is isolated
  - if they pass, decide whether or not to refactor
- If code is refactored, run the test cases again
  - if they fail, the problem is in the refactoring
  - if they pass, begin the next User Story
- illustrated next with User Stories 14 – 17 from text

Test-Driven Development (TDD) Life Cycle



TDD Example: a Boolean Function to Determine Leap Years

- *Definition: A year is a leap year if it is a multiple of 4, but century years are leap years only if they are multiples of 400*.
- Test-Driven Development would break this into small, individual user stories.
- "Coded" here in a pseudo-code (a *lingua franca*) that resembles Visual Basic.
- Try it yourself (with no looking ahead) in your favorite programming language.

User Story 14: A year divisible by 4 is a leap year

Test Case 1

Input: 2004

Expected Output: True

(existing) Pseudo-Code in normal font

```
Function isLeap(year) As Boolean
End isLeap
```

Running Test Case 1 on this code fails.
Add just enough code to make the test pass.

User Story 14: A year divisible by 4 is a leap year

Test Case 1

Input: 2004

Expected Output: True

(updated) Pseudo-Code in bold face font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   'MOD is the modulo arithmetic built-in operator in most languages
   If (( year MOD 4) = 0) Then
      IsLeap = True
   EndIf
End isLeap
```

Test Case 1 passes. Now do User Story 2.

User Story 15: A year not divisible by 4 is a common year

Test Case 1 Input: 2004
Expected Output: True
Test Case 2 Input: 2007
Expected Output: False

(existing) Pseudo-Code in normal font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   If (( year MOD 4) = 0) Then
       IsLeap = True
   EndIf
End isLeap
```

Test Case 1 passes. Test Case 2 fails. Now add just enough code so that Test Case 2 passes.

User Story 15: A year not divisible by 4 is a common year

Test Case 1 Input: 2004
Expected Output: True
Test Case 2 Input: 2007
Expected Output: False

(updated) Pseudo-Code In bold face font

```
Function isLeap(year) As Boolean
   dim year AS Integer

   If (( year MOD 4) = 0) Then
      IsLeap = True
   Else isLeap = False
   EndIf
End isLeap
```

Test Cases 1 and 2 pass. Now do User Story 3

Refactoring…

(updated) Pseudo-Code In bold face font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   isLeap = FALSE              ‘eliminates the else rule
   If (( year MOD 4) = 0) Then
      IsLeap = True
   EndIf
End isLeap
```

Test Cases 1 and 2 pass. Now do User Story 3

User Story 16: A century year not divisible by 400 is a common year

Test Case 1 Input: 2004, Expected Output: True
Test Case 2 Input: 2007, Expected Output: False
Test Case 3 Input: 1900, Expected Output: False

(existing) Pseudo-Code In normal font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   isLeap = False
   If (( year MOD 4) = 0) Then
       IsLeap = True
   EndIf
End isLeap
```

Test Cases 1 and 2 pass. Test Case 3 fails. Now add just enough code so that Test Case 3 passes.

User Story 16: A century year not divisible by 400 is a common year

Test Case 1 Input: 2004, Expected Output: True
Test Case 2 Input: 2007, Expected Output: False
Test Case 3 Input: 1900, Expected Output: False

(updated) Pseudo-Code In bold face font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   isLeap = False
   If ((( year MOD 4) = 0) AND NOT((year MOD 100) = 0))) Then
      IsLeap = True
   EndIf
End isLeap
```

Test Cases 1, 2 and 3 pass. Now do User Story 4

User Story 17: A century year divisible by 400 is a leap year

Test Case 1 Input: 2004, Expected Output: True
Test Case 2 Input: 2007, Expected Output: False
Test Case 3 Input: 1900, Expected Output: False
Test Case 4 Input: 2000, Expected Output: True

(existing) Pseudo-Code In normal font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   isLeap = False
   If ((( year MOD 4) = 0) AND NOT((year MOD 100) = 0))) Then
       IsLeap = True
   EndIf
End isLeap
```

Test Cases 1, 2 and 3 pass. Test case 4 fails. Now add just enough code so that test case 4 passes.

User Story 17: A century year divisible by 400 is a leap year

Test Case 1	Input: 2004, Expected Output: True
Test Case 2	Input: 2007, Expected Output: False
Test Case 3	Input: 1900, Expected Output: False
Test Case 4	Input: 2000, Expected Output: True

(updated) Pseudo-Code In bold face font

```
Function isLeap(year) As Boolean
   dim year AS Integer
   isLeap = False
   If ((( year MOD 4) = 0) AND NOT((year MOD 100) = 0))) OR
            ((year MOD 400 = 0)) Then
      IsLeap = True
   EndIf
End isLeap
```

Test Cases 1, 2, 3 and 4 pass. Done with function isLeap.

Advantages of Test Driven Development

- In this example, the steps are deliberately small.
- Customer and developer can (should!) jointly determine granularity of user stories.
- Fault isolation is greatly simplified (in fact, trivial). If a test case fails, the fault must be in the most recently added code.
- Once a new test case passes, a working (subset) of the desired software can always be delivered.
- Something always works!

Disadvantages of Test Driven Development

- Useful granularity is an issue.
- There is no guarantee that user stories "arrive" in a sensible order.
- There is no guarantee that user stories are the "same size" (or require similar effort)
- Bottom-up coding often results in poorly structured code, making refactoring necessary.
- Tool support (*e.g.* jUnit) is essential.
- What are the implications for
  - configuration management?
  - software maintenance?

Automated Test Execution Frameworks

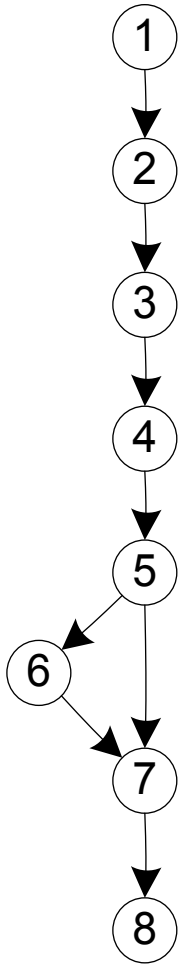
- Test execution frameworks exist for most common programming languages, *e.g.*
  - cUnit for C
  - jUnit for java
- Test cases are written as assertions
- Assertions for the isLeap method in ValidDate
  - assertEquals(true, ValidDate.isLeap(204));
  - assertEquals(false, ValidDate.isLeap(1900));
  - assertEquals(false, ValidDate.isLeap(1999));
  - assertEquals(true, ValidDate.isLeap(2000));

Comparison of (MDD) and (TDD)

- First American's view of Eagles and Mice
  - Eagles have the "big picture"
  - Mice focus on the details
  - (both views are important!)
- MDD is a rigorous, top-down approach.
- TDD is a bottom-up approach.

TDD isLeap in Visual Basic (refactored)

```
Public Function isLeap(year) As Boolean
        Dim year As Integer
        Dim c1, c2, c3 As Boolean
1.      c1 = (year Mod 4 = 0)
2.      c2 = (year Mod 100 = 0)
3.      c3 = (year Mod 400 = 0)
4.      isLeap = False
5.      If ( (c1 AND NOT(c2)) OR (c3)) Then
6.              IsLeap = True
7.      EndIf
8.      End Function
```

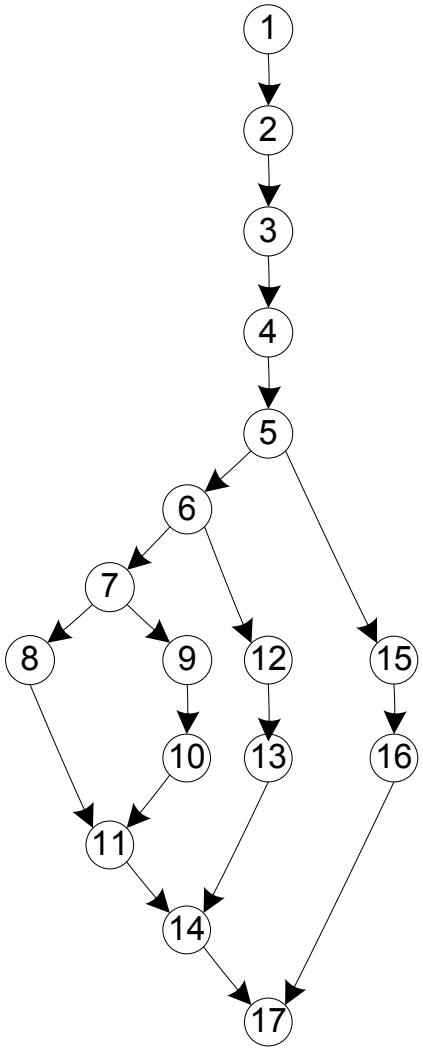


Decision Table Model of isLeap

Conditions	r1	r2	r3	r4	r5	r6	r7	r8
C1. year is a multiple of 4	T	T	T	T	F	F	F	F
C2. year is a century year	T	T	F	F	T	T	F	F
C3. year is a multiple of 400	T	F	T	F	T	F	T	F
Actions								
(logically impossible)			X		X	X	X	
A1. year is a common year		X						x
A2. year is a leap year	X			X				
test case: year =	2000	1900		2012				2011

MDD isLeap in Visual Basic

```
Public Function isLeap(year) As Boolean
    Dim year As Integer
    Dim c1, c2, c3 As Boolean
 1.  c1 = (year Mod 4 = 0)
 2.  c2 = (year Mod 100 = 0)
 3.  c3 = (year Mod 400 = 0)
 4,  isLeap = False
 5.  If c1 Then
 6.      If c2  Then
 7.          If c3 Then
 8.              isLeap = True     'rule r1
 9.          Else
10.              isLeap = False    'rule r2
11.          End If
12.      Else
13.          isLeap = True         'rule r4
14.      End If
15. Else
16.      isLeap = False            'rule r8
17. End If
End Function
```



Observations

- The TDD version is less complex (really?)
  - Why?
- The TDD version gradually built up to a compound condition (that might be hard to understand, and to modify).
- The decision table model assures completeness
- Both versions require 4 test cases

Conclusions

- The Agile community is VERY passionate about its methods.
- Agile Development clearly works well for SOME applications
  - time critical
  - uncertain customer (I'll know what I want when I see it!)
- Some agile methods apply to non-agile projects
  - Context-Driven Testing
  - Good Enough Testing
  - Exploratory Testing
- Test Driven Development is marvelous for fault isolation.

Open Questions

- Does Agile (bottom up) Development actually result in better designs?
- Can any form of Agile Testing reveal “deep faults”?
  - e.g.: faults revealed only by data flow (define/use) testing
  - computational faults
  - time-dependent faults
- What about maintenance? Agile code has
  - well-named variables and components
  - (hopefully) been refactored carefully
  - no comments
  - test cases are the specification

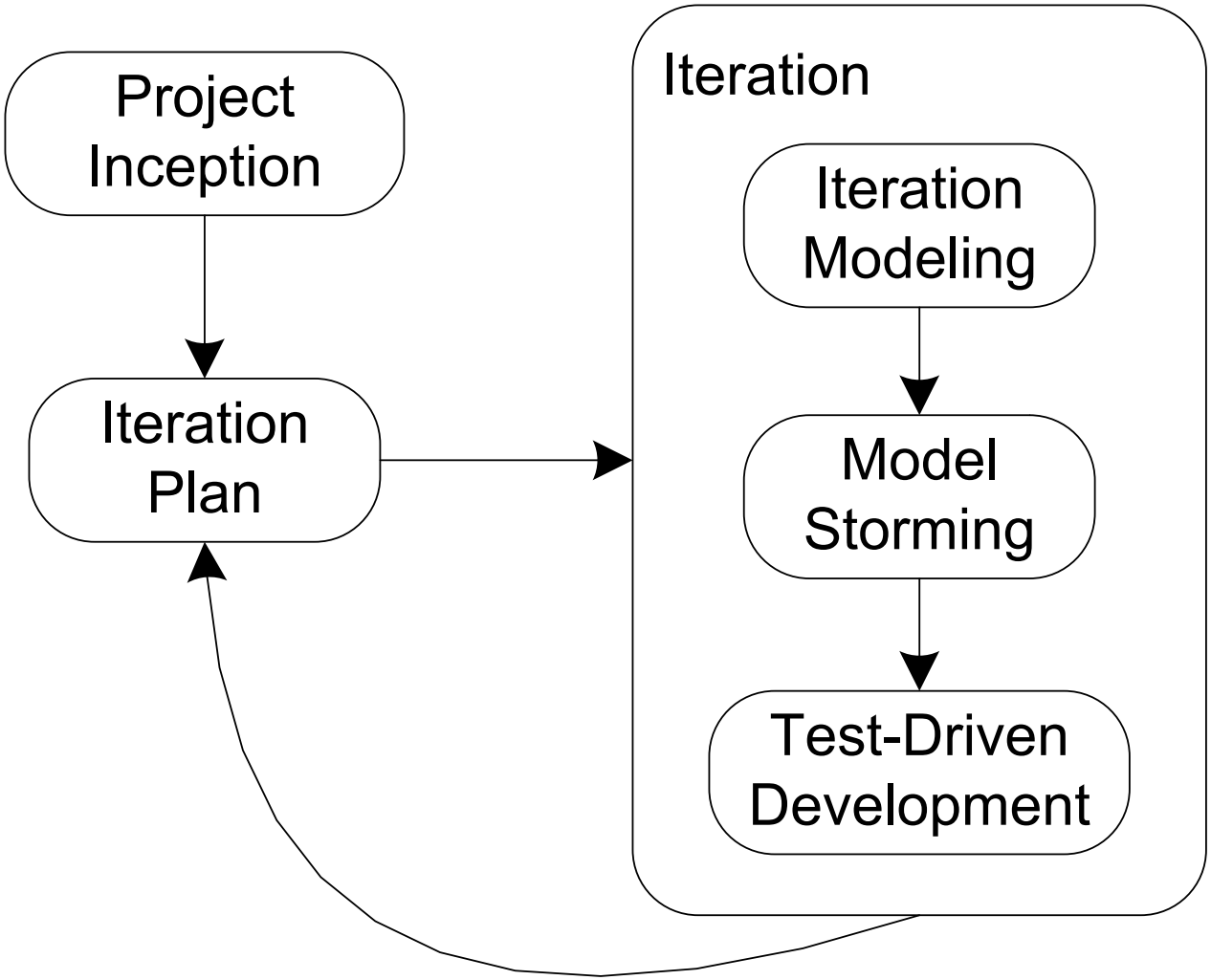
A Compromise?

- How might we capture/combine the advantages of various lifecycles to avoid the known deficiencies?
- Lessons from my friend Georg (German Ph.D. mathematician) and Go player
- "A successful Go player must have both good strategy and good tactics."
- Georg's contention...
  - Design = strategy
  - TDD = tactics

Agile Model-Driven Development

- Scott Ambler
- Model just enough for the present user story
- Design is necessary!
- Questions:
  - can bottom-up design deal with complex situations?
  - does this mean designs must be refactored?

Agile Model-Driven Development



Model-Driven Agile Development

- Re-arrangement of AMDD
- Early emphasis on design as one step (thanks, Georg)
- Implementation uses Test-Driven Development

Model-Driven Agile Development