1. **The original code and automated test cases for the "smelly" code including two separate examples of bad smells along with a screen dump showing that the automated tests pass with the smelly code**

**The code for the user story:**

The user story for Sprint 1 is to check the GEDCOM file that a child must be born only after the marriage of parents, not later than 9 months after their divorce and not after the death of parents.

# Vaishnavi Gopalakrishnan

from gedcom.validate import validator

@validator

def validate\_birth\_before\_marriage(individuals, families):

""" US09 - Birth should occur after the marriage of parents """

return\_flag = True

# Loop through individuals to compare their birth date

# with the marriage/divorce dates of their parents

for individual in individuals:

# Some individuals do not have parents defined

# if they are the oldest generation in the gedcom file,

if len(individual.child\_to) > 0:

# locate family of individual

for family in families:

if family.id == individual.child\_to[0]:

# Checks for a child born before marriage

if family.marriage:

if family.marriage > individual.birthdate:

print("Child is born before marriage")

return\_flag = False

return return\_flag

# Vaishnavi Gopalakrishnan

from gedcom.validate import validator

@validator

def validate\_birth\_after\_divorce(individuals, families):

""" US09 - Birth should occur after the marriage of parents """

return\_flag = True

# Loop through individuals to compare their birth date

# with the marriage/divorce dates of their parents

for individual in individuals:

# Some individuals do not have parents defined

# if they are the oldest generation in the gedcom file,

if len(individual.child\_to) > 0:

# locate family of individual

for family in families:

if family.id == individual.child\_to[0]:

# checks for child born after divorce

if family.marriage and family.divorce:

if family.divorce < individual.birthdate:

print("Child is born after divorce")

return\_flag = False

return return\_flag

# Vaishnavi Gopalakrishnan

from gedcom.validate import validator

@validator

def validate\_birth\_after\_parent\_dead(individuals, families):

""" US10 - Birth should occur before the death of parents """

return\_flag = True

# Loop through individuals to compare their birth date

# with the death date of their parents

for individual in individuals:

if len(individual.child\_to) > 0:

father = None

father\_id = None

mother = None

mother\_id = None

fam = None

for family in families:

if family.id == individual.child\_to[0]:

father\_id = family.husband

mother\_id = family.wife

fam = family

break

for ind in individuals:

if ind.id == father\_id:

father = ind

if ind.id == mother\_id:

mother = ind

# Case when father dies more than 9 months before

from datetime import timedelta

if father and father.death is not None and \

father.death < individual.birthdate - timedelta(days=266):

print("Child is born in more than 9 months after death of father")

return\_flag = False

# Case when mother dies before birth of child.

if mother and mother.death is not None and mother.death < individual.birthdate:

print("Child is born after death of mother")

return\_flag = False

return return\_flag

**The test code:**

The file contains the sample family data to check if the user stories are validated and produces appropriate output. The code is not refactored and contains spaces, comments and definitions.

# Vaishnavi Gopalakrishnan

from nose.tools import \*

from gedcom.models import \*

from gedcom.validate import validators

def test\_birth\_before\_marriage\_false():

# Test with birth that happens after marriage

individual1 = Individual('11')

individual1.birth = Birth(datetime.date(1945, 4, 23))

individual1.death = Death(datetime.date(1995, 3, 26))

individual1.name = Name('Leandro', 'Ritter')

individual1.sex = Sex('M')

individual1.spouse\_to = [FamilySpouse('11'), ]

individual2 = Individual('12')

individual2.birth = Birth(datetime.date(1950, 12, 24))

individual2.name = Name('Alyssia', 'Holcomb')

individual2.sex = Sex('F')

individual2.spouse\_to = [FamilySpouse('11'), ]

individual3 = Individual('21')

individual3.birth = Birth(datetime.date(1969, 5, 13))

individual3.name = Name('Anita', 'Ritter')

individual3.sex = Sex('F')

individual3.child\_to = [FamilyChild('11'), ]

family = Family('11')

family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]

family.husbands = [Husband('11'), ]

family.wives = [Wife('12'), ]

family.children = [Child('21'), ]

individuals = [individual1, individual2, individual3]

families = [family, ]

result = validators['validate\_birth\_before\_marriage'](individuals, families)

assert\_equal(True, result)

def test\_birth\_before\_marriage\_true():

# Test with birth that happens before marriage

individual1 = Individual('11')

individual1.birth = Birth(datetime.date(1945, 4, 23))

individual1.death = Death(datetime.date(1995, 3, 26))

individual1.name = Name('Leandro', 'Ritter')

individual1.sex = Sex('M')

individual1.spouse\_to = [FamilySpouse('11'), ]

individual2 = Individual('12')

individual2.birth = Birth(datetime.date(1950, 12, 24))

individual2.name = Name('Alyssia', 'Holcomb')

individual2.sex = Sex('F')

individual2.spouse\_to = [FamilySpouse('11'), ]

individual3 = Individual('21')

individual3.birth = Birth(datetime.date(1960, 5, 13))

individual3.name = Name('Anita', 'Ritter')

individual3.sex = Sex('F')

individual3.child\_to = [FamilyChild('11'), ]

family = Family('11')

family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]

family.husbands = [Husband('11'), ]

family.wives = [Wife('12'), ]

family.children = [Child('21'), ]

individuals = [individual1, individual2, individual3]

families = [family, ]

result = validators['validate\_birth\_before\_marriage'](individuals, families)

assert\_equal(True, result)

def test\_birth\_before\_parent\_death\_false():

# Test with birth that happens before parent death

individual1 = Individual('11')

individual1.birth = Birth(datetime.date(1945, 4, 23))

individual1.death = Death(datetime.date(1995, 3, 26))

individual1.name = Name('Leandro', 'Ritter')

individual1.sex = Sex('M')

individual1.spouse\_to = [FamilySpouse('11'), ]

individual2 = Individual('12')

individual2.birth = Birth(datetime.date(1950, 12, 24))

individual2.name = Name('Alyssia', 'Holcomb')

individual2.sex = Sex('F')

individual2.spouse\_to = [FamilySpouse('11'), ]

individual3 = Individual('21')

individual3.birth = Birth(datetime.date(1969, 5, 13))

individual3.name = Name('Anita', 'Ritter')

individual3.sex = Sex('F')

individual3.child\_to = [FamilyChild('11'), ]

family = Family('11')

family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]

family.husbands = [Husband('11'), ]

family.wives = [Wife('12'), ]

family.children = [Child('21'), ]

individuals = [individual1, individual2, individual3]

families = [family, ]

result = validators['validate\_birth\_after\_parent\_dead'](individuals, families)

assert\_equal(True, result)

def test\_birth\_before\_parent\_death\_true():

# Test with birth that happens after parent death

individual1 = Individual('11')

individual1.birth = Birth(datetime.date(1945, 4, 23))

individual1.death = Death(datetime.date(1960, 3, 26))

individual1.name = Name('Leandro', 'Ritter')

individual1.sex = Sex('M')

individual1.spouse\_to = [FamilySpouse('11'), ]

individual2 = Individual('12')

individual2.birth = Birth(datetime.date(1930, 12, 24))

individual2.name = Name('Alyssia', 'Holcomb')

individual2.sex = Sex('F')

individual2.spouse\_to = [FamilySpouse('11'), ]

individual3 = Individual('21')

individual3.birth = Birth(datetime.date(1969, 5, 13))

individual3.name = Name('Anita', 'Ritter')

individual3.sex = Sex('F')

individual3.child\_to = [FamilyChild('11'), ]

family = Family('11')

family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]

family.husbands = [Husband('11'), ]

family.wives = [Wife('12'), ]

family.children = [Child('21'), ]

individuals = [individual1, individual2, individual3]

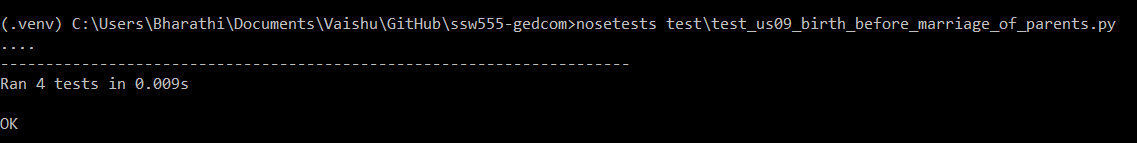
families = [family, ]

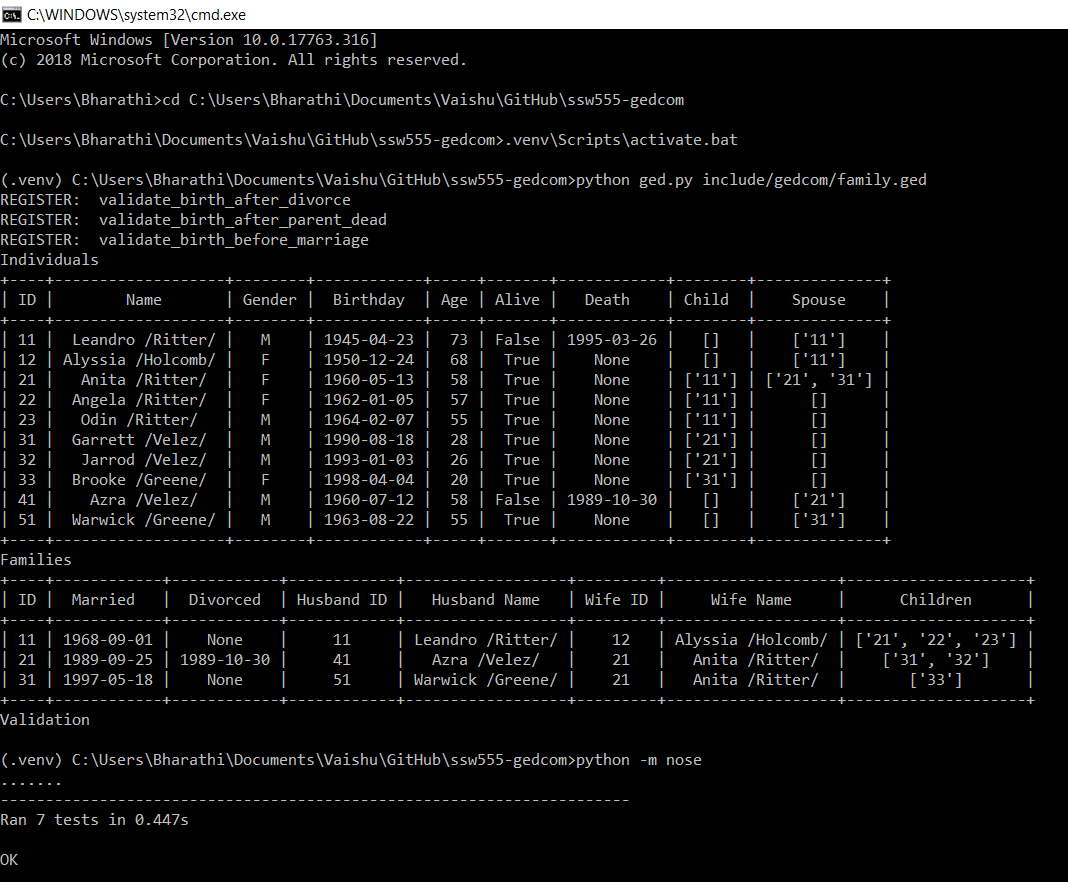
result = validators['validate\_birth\_after\_parent\_dead'](individuals, families)

assert\_equal(True, result)

**The picture of the screen dump with the test results:**

4 tests are created for the user story of sprint 1. All the tests ran successfully. Test\_us09\_birth\_before\_marriage\_death is the test file.





1. **The refactored code and automated test cases along with a screen dump showing that the automated tests pass with the refactored code.**

**The test code:**

The user stories are separated into 3 python files. Removed extra spaces, comments and definitions. It is kept inside a new folder “Validate”.

# Vaishnavi Gopalakrishnan  
from gedcom.validate import validator  
  
  
@validator  
def validate\_birth\_before\_marriage(individuals, families):  
 *""" US09 - Birth should occur after the marriage of parents """* for individual in individuals:  
 if individual.child\_to:  
 for family in families:  
 if family.id in individual.child\_to:  
 if family.marriage and family.marriage > individual.birthdate:  
 print('Individual {} born before marriage in family {}'  
 .format(individual.id, family.id))  
 return False  
 return True

# Vaishnavi Gopalakrishnan  
from gedcom.validate import validator  
  
  
@validator  
def validate\_birth\_after\_divorce(individuals, families):  
 *""" US09 - Birth should occur before the parents divorce """* for individual in individuals:  
 if individual.child\_to:  
 for family in families:  
 if family.id in individual.child\_to:  
 if family.divorce and family.divorce < individual.birthdate:  
 print('Individual {} born after divorce in family {}.'  
 .format(individual.id, family.id))  
 return False  
 return True

# Vaishnavi Gopalakrishnan  
from gedcom.validate import validator  
  
  
@validator  
def validate\_birth\_after\_parent\_dead(individuals, families):  
 *""" US10 - Birth should occur before the death of parents """* for individual in individuals:  
 if individual.child\_to:  
 for family in families:  
 if family.id in individual.child\_to:  
 mothers = family.wives  
 fathers = family.husbands  
 for mother, father in zip(mothers, fathers):  
 if individual.birthdate > mother.death.date:  
 print(  
 'Individual {} born after death of mother in family {}.' .format(  
 individual.id, family.id))  
 return False  
 if individual.birthdate > father.death.date:  
 print(  
 'Individual {} born after death of father in family {}.' .format(  
 individual.id, family.id))  
 return False  
 return True

**The test case:**

The test cases are also refactored to remove the extra spaces.

from nose.tools import \*  
  
from gedcom.models import \*  
from gedcom.validate import validators  
  
  
def test\_birth\_before\_marriage\_false():  
 # Test with birth that happens after marriage  
 individual1 = Individual('11')  
 individual1.birth = Birth(datetime.date(1945, 4, 23))  
 individual1.death = Death(datetime.date(1995, 3, 26))  
 individual1.name = Name('Leandro', 'Ritter')  
 individual1.sex = Sex('M')  
 individual1.spouse\_to = [FamilySpouse('11'), ]  
  
 individual2 = Individual('12')  
 individual2.birth = Birth(datetime.date(1950, 12, 24))  
 individual2.name = Name('Alyssia', 'Holcomb')  
 individual2.sex = Sex('F')  
 individual2.spouse\_to = [FamilySpouse('11'), ]  
  
 individual3 = Individual('21')  
 individual3.birth = Birth(datetime.date(1969, 5, 13))  
 individual3.name = Name('Anita', 'Ritter')  
 individual3.sex = Sex('F')  
 individual3.child\_to = [FamilyChild('11'), ]  
  
 family = Family('11')  
 family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]  
 family.husbands = [Husband('11'), ]  
 family.wives = [Wife('12'), ]  
 family.children = [Child('21'), ]  
  
 individuals = [individual1, individual2, individual3]  
 families = [family, ]  
  
 result = validators['validate\_birth\_before\_marriage'](  
 individuals, families)  
 assert\_equal(True, result)  
  
  
def test\_birth\_before\_marriage\_true():  
 # Test with birth that happens before marriage  
 individual1 = Individual('11')  
 individual1.birth = Birth(datetime.date(1945, 4, 23))  
 individual1.death = Death(datetime.date(1995, 3, 26))  
 individual1.name = Name('Leandro', 'Ritter')  
 individual1.sex = Sex('M')  
 individual1.spouse\_to = [FamilySpouse('11'), ]  
  
 individual2 = Individual('12')  
 individual2.birth = Birth(datetime.date(1950, 12, 24))  
 individual2.name = Name('Alyssia', 'Holcomb')  
 individual2.sex = Sex('F')  
 individual2.spouse\_to = [FamilySpouse('11'), ]  
  
 individual3 = Individual('21')  
 individual3.birth = Birth(datetime.date(1960, 5, 13))  
 individual3.name = Name('Anita', 'Ritter')  
 individual3.sex = Sex('F')  
 individual3.child\_to = [FamilyChild('11'), ]  
  
 family = Family('11')  
 family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]  
 family.husbands = [Husband('11'), ]  
 family.wives = [Wife('12'), ]  
 family.children = [Child('21'), ]  
  
 individuals = [individual1, individual2, individual3]  
 families = [family, ]  
  
 result = validators['validate\_birth\_before\_marriage'](  
 individuals, families)  
 assert\_equal(True, result)  
  
  
def test\_birth\_before\_parent\_death\_false():  
 # Test with birth that happens before parent death  
 individual1 = Individual('11')  
 individual1.birth = Birth(datetime.date(1945, 4, 23))  
 individual1.death = Death(datetime.date(1995, 3, 26))  
 individual1.name = Name('Leandro', 'Ritter')  
 individual1.sex = Sex('M')  
 individual1.spouse\_to = [FamilySpouse('11'), ]  
  
 individual2 = Individual('12')  
 individual2.birth = Birth(datetime.date(1950, 12, 24))  
 individual2.name = Name('Alyssia', 'Holcomb')  
 individual2.sex = Sex('F')  
 individual2.spouse\_to = [FamilySpouse('11'), ]  
  
 individual3 = Individual('21')  
 individual3.birth = Birth(datetime.date(1969, 5, 13))  
 individual3.name = Name('Anita', 'Ritter')  
 individual3.sex = Sex('F')  
 individual3.child\_to = [FamilyChild('11'), ]  
  
 family = Family('11')  
 family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]  
 family.husbands = [Husband('11'), ]  
 family.wives = [Wife('12'), ]  
 family.children = [Child('21'), ]  
  
 individuals = [individual1, individual2, individual3]  
 families = [family, ]  
  
 result = validators['validate\_birth\_after\_parent\_dead'](  
 individuals, families)  
 assert\_equal(True, result)  
  
  
def test\_birth\_before\_parent\_death\_true():  
 # Test with birth that happens after parent death  
 individual1 = Individual('11')  
 individual1.birth = Birth(datetime.date(1945, 4, 23))  
 individual1.death = Death(datetime.date(1960, 3, 26))  
 individual1.name = Name('Leandro', 'Ritter')  
 individual1.sex = Sex('M')  
 individual1.spouse\_to = [FamilySpouse('11'), ]  
  
 individual2 = Individual('12')  
 individual2.birth = Birth(datetime.date(1930, 12, 24))  
 individual2.name = Name('Alyssia', 'Holcomb')  
 individual2.sex = Sex('F')  
 individual2.spouse\_to = [FamilySpouse('11'), ]  
  
 individual3 = Individual('21')  
 individual3.birth = Birth(datetime.date(1969, 5, 13))  
 individual3.name = Name('Anita', 'Ritter')  
 individual3.sex = Sex('F')  
 individual3.child\_to = [FamilyChild('11'), ]  
  
 family = Family('11')  
 family.marriages = [Marriage(datetime.date(1968, 9, 1)), ]  
 family.husbands = [Husband('11'), ]  
 family.wives = [Wife('12'), ]  
 family.children = [Child('21'), ]  
  
 individuals = [individual1, individual2, individual3]  
 families = [family, ]  
  
 result = validators['validate\_birth\_after\_parent\_dead'](  
 individuals, families)  
 assert\_equal(True, result)

**The picture of the screen dump with the test results:**

4 tests are created for the user story of sprint 1. All the tests ran successfully.

Test\_us09 is the test file.

