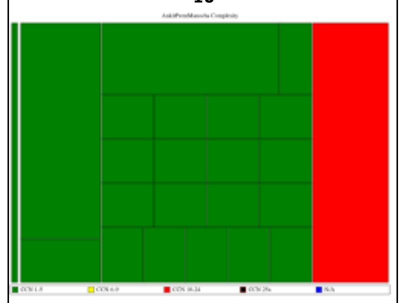
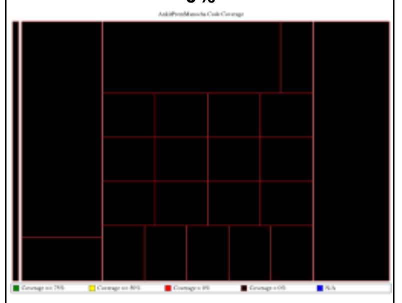
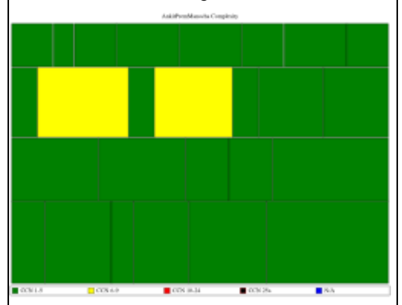
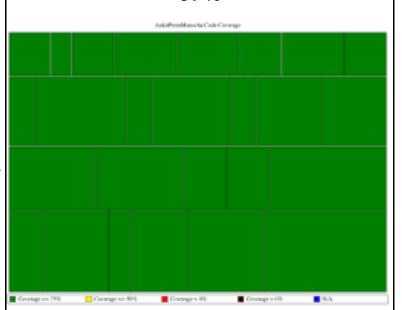
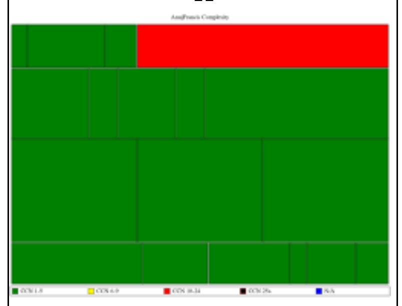
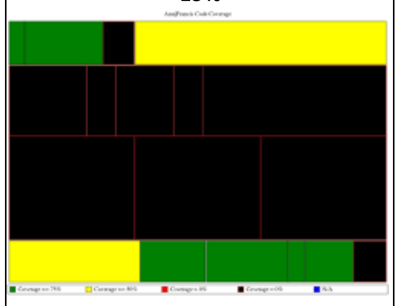
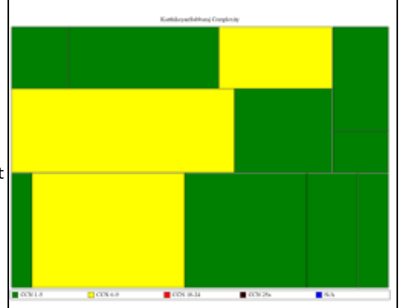
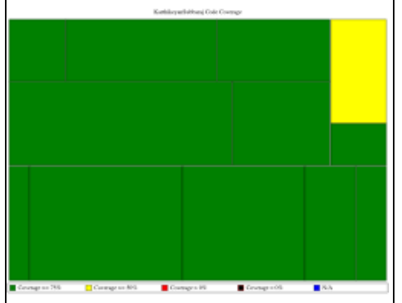
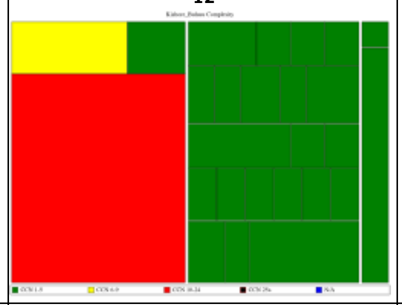
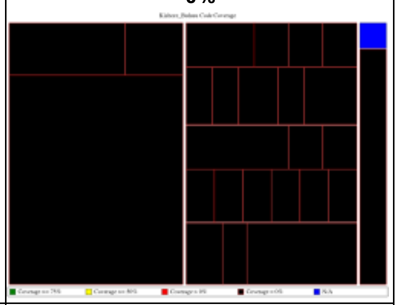
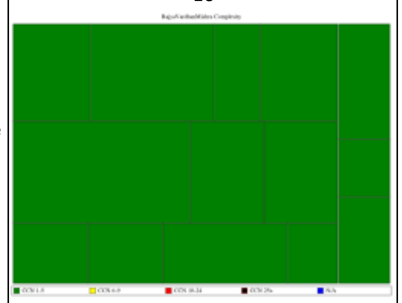
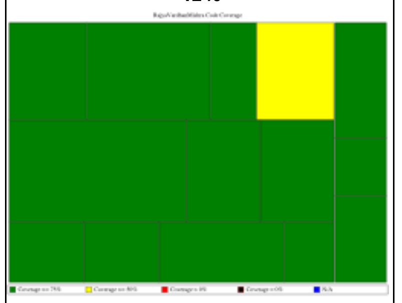
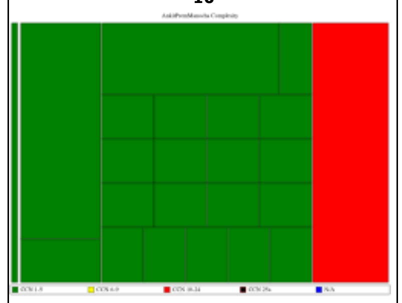
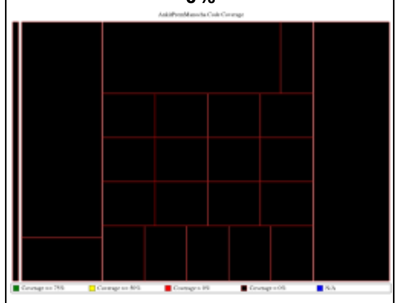
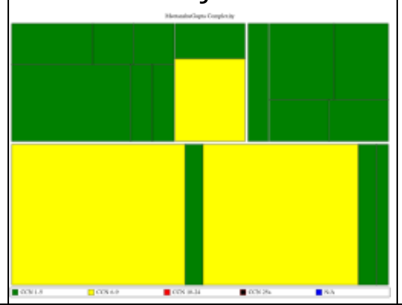
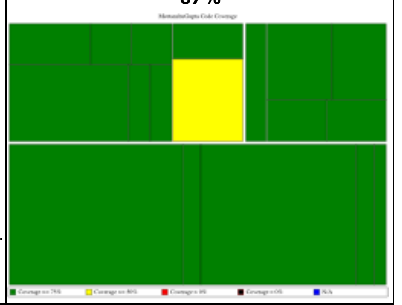
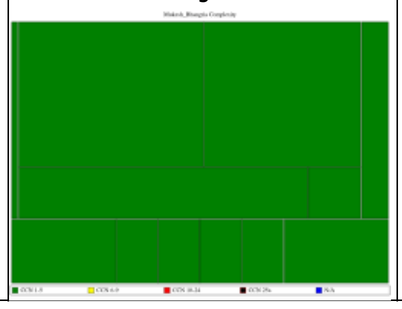
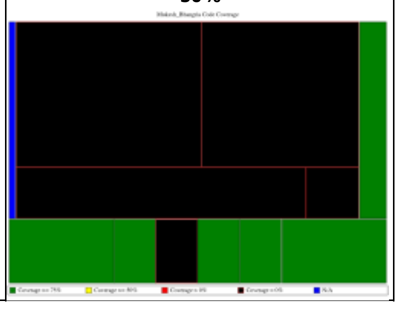
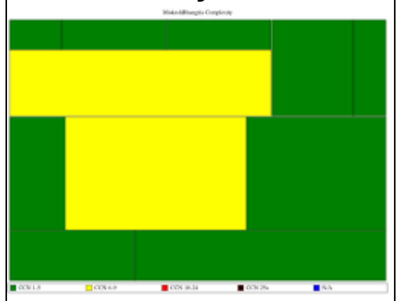
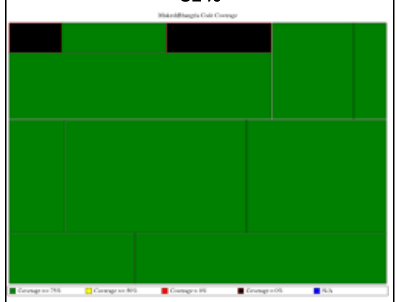
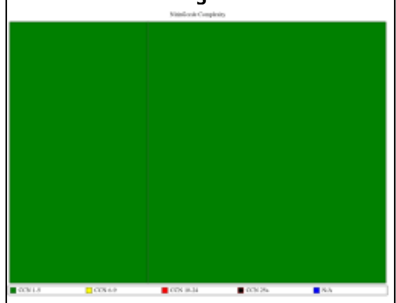
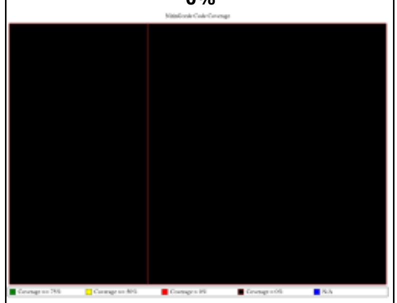
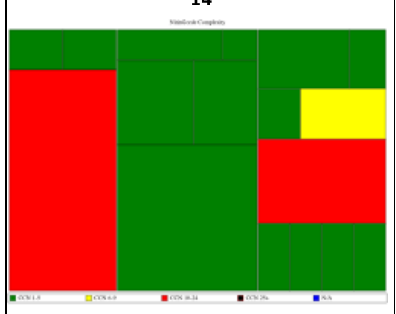
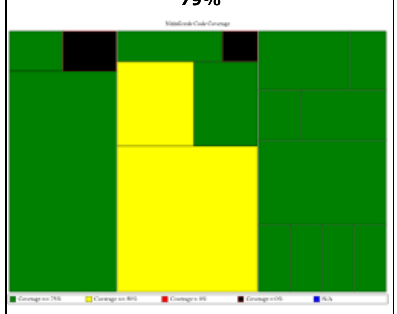
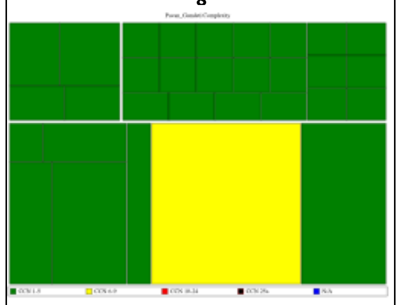
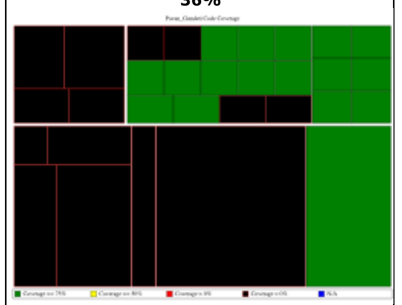
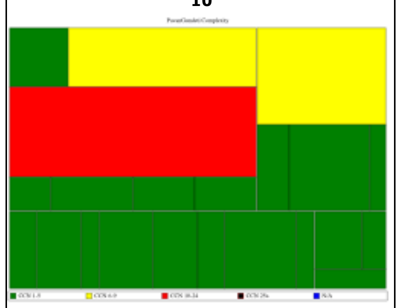
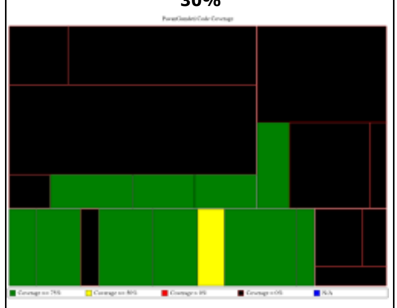
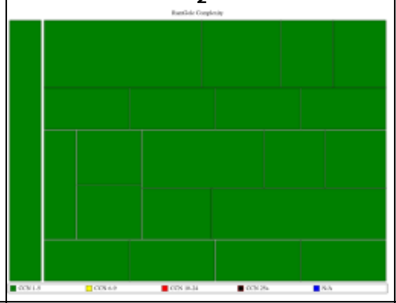
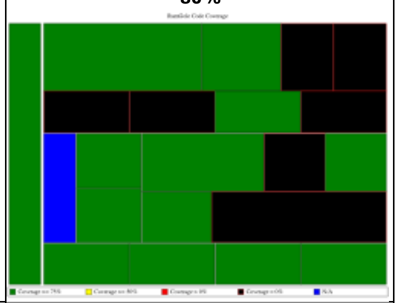
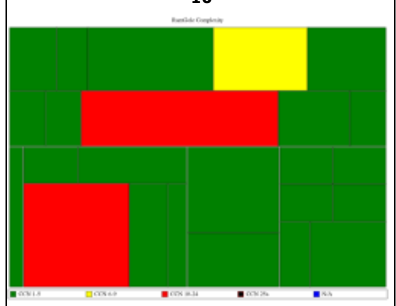
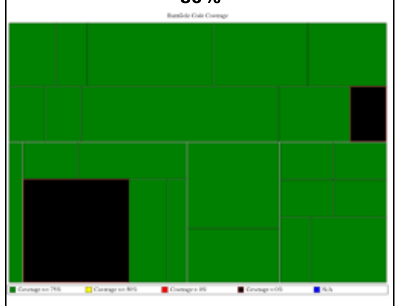
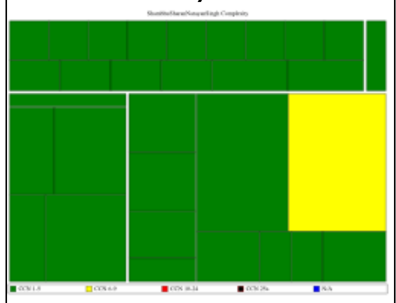
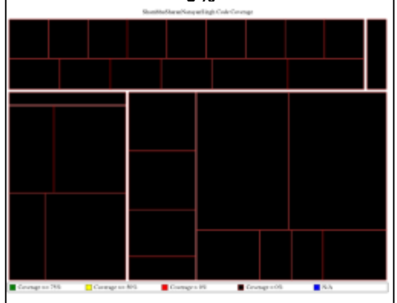
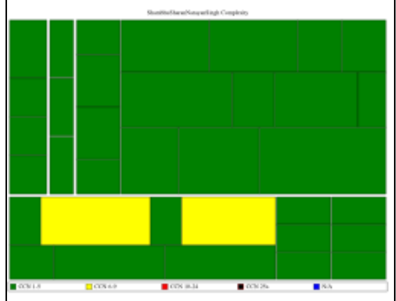
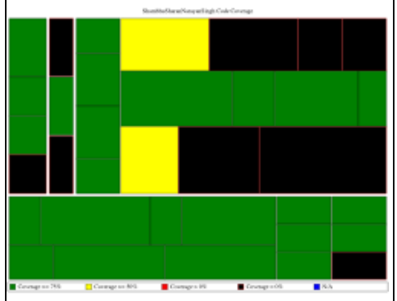
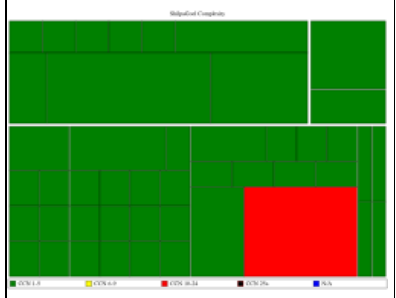
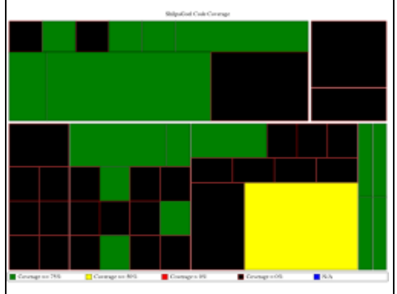
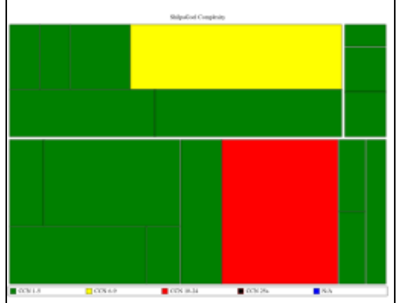
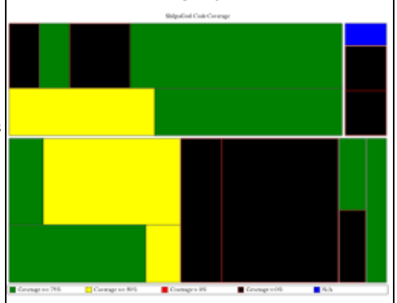
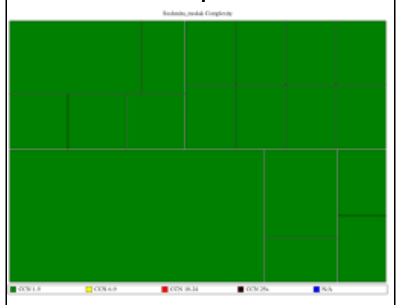
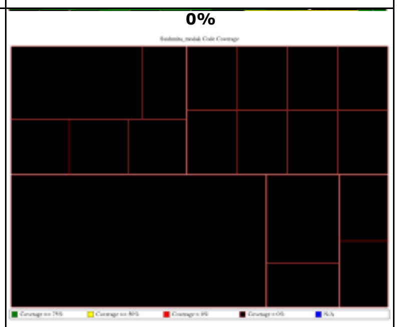
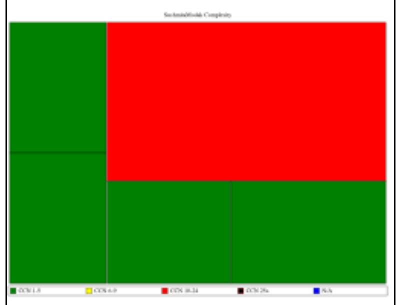
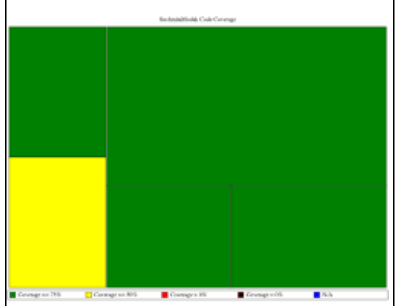
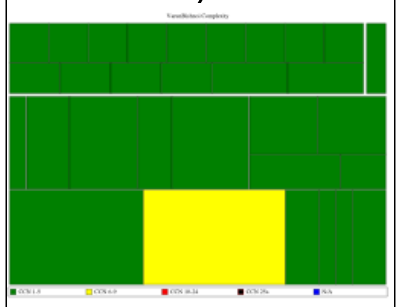
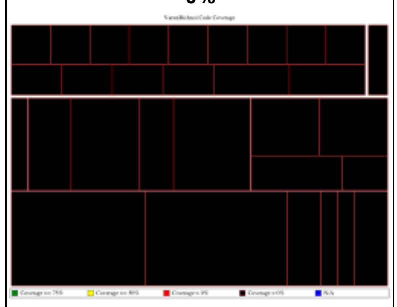
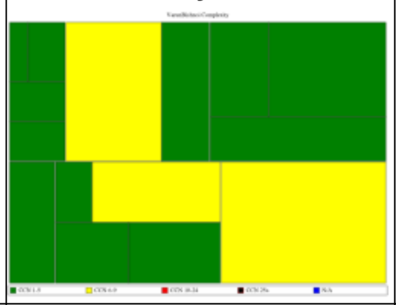
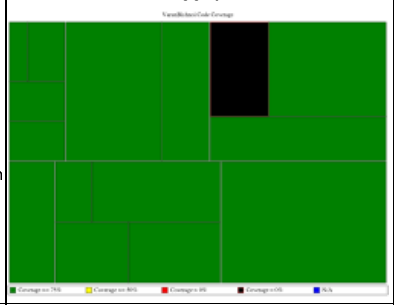
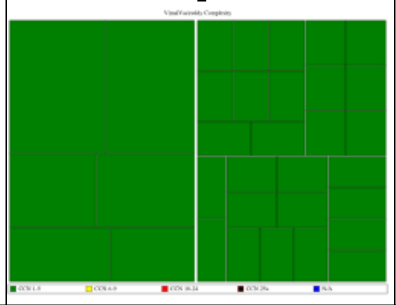
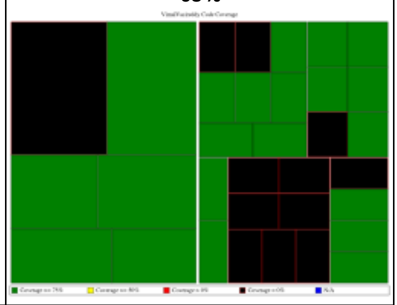
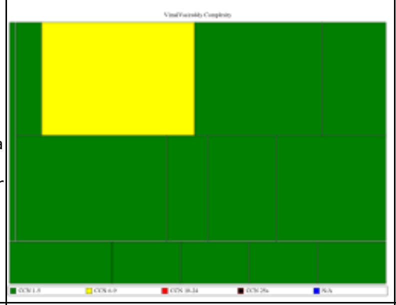
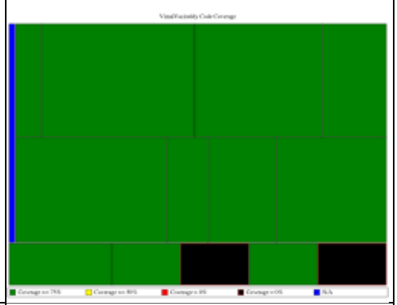
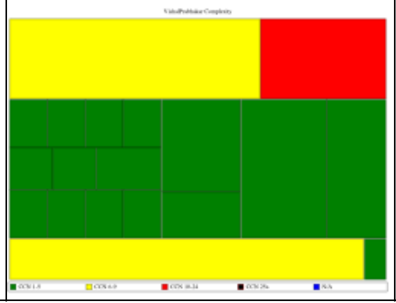
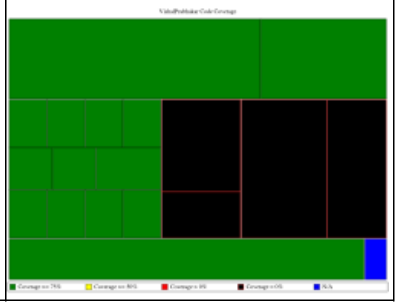
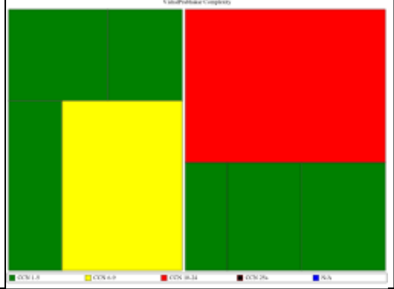
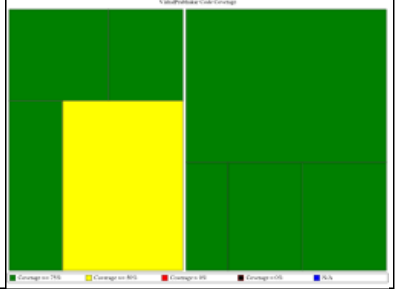
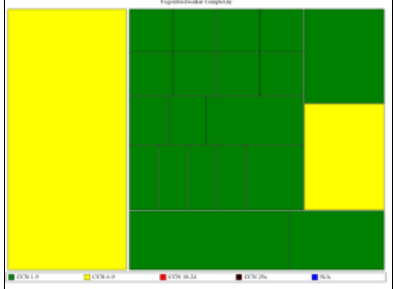
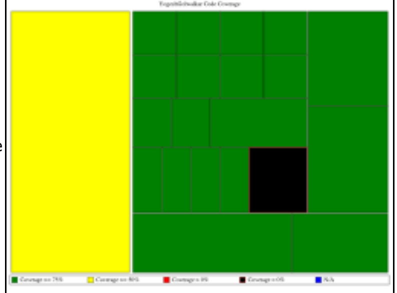
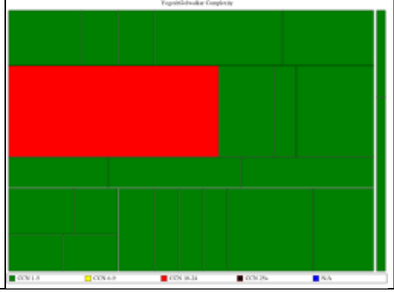
				Production Code			Test Code				OO Design Skill					Testing Skill	
Name	Assessment	Overall Score	Program Meets Requirements	# of Classes	Total Code Size	Method Length Range	# of Test Cases	Total Code Size	Method Length Range	Knowledge of xUnit	Code Smells in the code	Object Oriented Design Skill	Cyclomatic Complexity	NPath Complexity	Fan-Out Complexity	Quality of Unit Test	Test Coverage
Student 1	Pre	**Poor. Word to word copy of Meetanshu Gupta's solution**	No. The exempted items are hard coded. The tax % is hard coded. Rounding logic is missing.	4	106	1-17	4	68	11-11	None	Conditional Complexity (10) Long Method (17) Magic Numbers Duplicate Code Data Class Switch Statements Feature Envy Indecent Exposure Primitive Obsession Dead Code Temporary Field Inappropriate Naming	Code is extremely Procedural in nature. Violates most basic OO design principles (SOLID, Tell Don't Ask, DRY, etc.)	**10** 	10	3	Has written the tests in Main method. No assert statements, have to manually check. Checks only a few end-to-end happy path cases. Massive duplication in the tests.	**0%** 
	Post	**Good Overall Good improvement on both OO design and Quality of Unit Tests compared to Pre-Assessment.**	Yes. Only thing is your program could avoid aking the startDateTime and use Now instead.	**2**	**120**	12 (>10 - Watch Out)	3	224	20 (>10 - Watch Out)	Good	Conditional Complexity (4)	Good. Nice distribution of responsibility. Good to see well named small methods. I don't like the fact the your GetEarliestFreeSlot returns a null if it can't find a slot. A large number of methods in Calendar class are static, this could be avoided. Good to see that you've externalized DayStartHour and DayEndHour to properties file.	**6** 	2	3	Pretty Good. Nice test names. Each test has a single responsibility. I'm not a fan of TestData classes, as they make tests harder to understand, but I like the fact that you've used it to eliminate duplcate data. Overall the coverage is really good.	**97%** 
Student 2	Pre	**Very Poor. Feels like the training had no impact.**	Its hard to understand what the code is doing. Even if it were to meet the requirements, the code is pretty much useless.	5	182	1-19	1	141	2-19	Very Poor. Production code contains test code. Also one test class is a hotch-potch of stuff. Can't make any sense of it.	Verbose Code Inappropriate Naming Comments Dead Code Duplicated code Primitive Obsession Lazy Class Long Method Oddball Solution Feature Envy Black Sheep Conditional Complexity Data Class Indecent Exposure Solution Sprawl Temporary Field	Extremely poor OO Design skills. Solution contains the following classes: Calendar (Just a data class with all public fields) FreeTimeSlot (contains one large method to calculate common free slot) Main (Contains test data) MeetingAssisstant (Contains a method to initialize the system) Participant (Just a data class with all public fields)	**11** 	9	5	Pretty low coverage. Most code in the test is commented out. Extremely poor test names. I've not idea what the objective/purpose of the test is. Lot of duplication in the test.	**28%** 
	Post	**Slightly Above Average**	Mostly. Works only for 2 participants. Good initial thoughts on optimizing your solution. However overall solution could be much simpler.	2	**76**	**21 (>10 - Watch Out)**	1	131	12 (>10 - Watch Out)	Decent	Conditional Complexity Magic Numbers	Decent. Good separation of responsibilities. Except one method, rest of them are small and have single responsibility. Good to see you use _schedule and _emptySlots to speed up the process of finding an empty slot. This could be further simplified. Your Assistant class works only with 2 participants. Why? This will not satisfy our requirement. Using out parameters is a bad practice, makes it hard to understand what is going on. Should be avoided.	**8** 	1	1	Good coverage of different scenarios. Tests are getting a little repetitive and verbose, would be good to address this issues. If conditions and Sysouts in the tests are an anti-pattern which needs to be avoided. No tests for exceptional conditions.	**83%** 
Student 3	Pre	**Slightly below average**	Partially. The logic is implemented correctly. The exempted items are hard coded. The tax % is hard coded as magic numbers.	7	165	1-48	0	0	0	?	Dead Code Magic Numbers Duplicated code Large Class Long Method Switch Statements Speculative Generality Feature Envy Conditional Complexity Data Class Temporary Field	Couple of good abstractions. OK in terms of modularity. But the BillingSystem class is very monolithic, complex and procedural. Little bit of speculative generality with the TaxCalculator	**12** 	39	9	No Tests	**0%** 
	Post	**Good Overall Good improvement on both OO design and Quality of Unit Tests compared to Pre-Assessment.**	Mostly. Your program only searches for available slots in the same day, but the requirements to search in future dates as well. Nice initial thoughts on optimizing the data structure to store the schedule.	**4**	**162**	**31 (>25 - Scary)**	1	129	18 (>10 - Watch Out)	Good	Magic Numbers Duplicate Code Long Method	Nice distribution of responsiblities. Good to see much crisper code. Some duplication between Participant and MeetingScheduler could be avoided. Why is all the production code inder the Test namespace? Consider breaking up ScheduleMeeting into smaller methods. Also consider using recursion to simplify your logic.	**10** 	4	3	Good. Nice test names. Each test has a single responsibility. Some duplication between the tests could be reduced. Overall good coverage.	**42%** 
Student 4	Pre	**Poor. Word to word copy of Ankit Prem Manocha's solution**	No. The exempted items are hard coded. The tax % is hard coded. Rounding logic is missing.	4	106	1-17	4	68	11-11	None	Conditional Complexity (10) Long Method (17) Magic Numbers Duplicate Code Data Class Switch Statements Feature Envy Indecent Exposure Primitive Obsession Dead Code Temporary Field Inappropriate Naming	Code is extremely Procedural in nature. Violates most basic OO design principles (SOLID, Tell Don't Ask, DRY, etc.)	**10** 	10	3	Has written the tests in Main method. No assert statements, have to manually check. Checks only a few end-to-end happy path cases. Massive duplication in the tests.	**0%** 
	Post	**Slightly below Average. Slight improvement compared to pre-assessment when we look at the testing side of things, however OO Design is pretty much the same. Large number of code smells in the code. Looks like you've at least applied some TDD principles.**	Mostly.	6	163	2-29	5	277	2-10	Good	Magic Numbers Verbose Code Comments Dead Code Duplicated code Primitive Obsession Large Class Lazy Class Long Method Black Sheep Conditional Complexity Data Class Indecent Exposure Temporary Field	OO design needs significant improvement. Out of the 6 classes, the core logic is just in 1 static class. AppointmentType (enum) EmployeeCalendar (Data class - basically a list wrapper no real logic) EmployeeCalendarEntry (Struct - no logic) Employee (Data class - no logic) EmployeeFinder (List wrapper) SchedulingAssistant (Static class. Only meaning full code which is directly related to the problem at hand. Rest all classes are just orthogonal classes)	**9** 	27	7	Good test coverage. Good test names. Since the setup data is in a different class from the test, it gets quite hard to understand why certain test results are showing up. The tests are very verbose and duplicated. Noise to Signal ratio is very large. Most complex method is least covered.	**87%** 
Student 5	Pre	**Average**	Partially. The logic is implemented correctly. The exempted items are hard coded. The tax % is hard coded as magic numbers.	5	94	1-20	2	32	2-3	Basic	Magic Number Duplicate Code Comments Large Class Lazy Class Long Method Feature Envy Data Class Indecent Exposure	Code is small and concise. Its mostly procedural, with a little bit of objects sprinkled over. The BigBill class is a God Object which does most of the logic. Its mostly static methods, not really an Object.	**5** 	12	7	Did not test the core of the logic. Poor test names. Assert Statement syntax is wrong.	**30%** 

				Production Code			Test Code				OO Design Skill					Testing Skill	
Name	Assessment	Overall Score	Program Meets Requirements	# of Classes	Total Code Size	Method Length Range	# of Test Cases	Total Code Size	Method Length Range	Knowledge of xUnit	Code Smells in the code	Object Oriented Design Skill	Cyclomatic Complexity	NPath Complexity	Fan-Out Complexity	Quality of Unit Test	Test Coverage
Student 5	Post	**Good.** **Nice improvements from the pre-assessment.** **Looks like you've applied a good number of TDD principles. More scope for improvement.**	Yes	3	78	1-13	3	194	2-101	Good	Magic Numbers Dead Code Conditional Complexity	Good OO Design. Minor improvements possible.	**9** 	7	3	Good test coverage. Good test names, can slightly improve it. A huge method to set up the test data. It gets quite hard to understand why certain test results are showing up. The tests are a bit verbose. Since the participants are statically setup using @BeforeClass, one test can have a side effect on another test.	**82%** 
Student 6	Pre	**Slightly below average**	Partially. The logic is implemented correctly. Rounding logic implemented correctly, but logic to display double is wrong. The exempted items are hard coded. The tax % is hard coded as magic numbers.	3	37	1-19	3	61	4-23	None	Inappropriate Naming Magic Number Long Method Feature Envy Conditional Complexity Data Class Indecent Exposure	Completely procedural code. (Class with bunch of static methods and 1 enum + a data class.)	**5** 	16	2	Has written the tests in Main method. No assert statements, have to manually check. Checks all the end-to-end happy path cases.	**0%** 
	Post	**Below Average.** **At least has written one Junit test. So is a slight improvement from before. But OO skills remain the same.** **I don't think you've used any TDD Principles. Looks like the test was retro-fitted in the end.**	Partially. Assumes that a month is 30 days long.	3	101	2-22	1	102	3-11	Basic	Magic Numbers Verbose Code Inappropriate Naming Dead Code Duplicated code Primitive Obsession Large Class Long Method Black Sheep Conditional Complexity Indecent Exposure Temporary Field	OO Design skills needs improvement. Slot - Is mostly a data class except for one method. Also exposes all its internals. CalendarAssistUtilities - Extremely complex util class. CalendarAssist - Basically a static class with one long, complex method. Overall code is obsessed with boolean flags.	**14** 	66	3	Coverage is not adequate. Test names can be improved. Lot of duplication in the test. Its very hard to understand what the tests are doing. Tests are not very communicative. Looks like the test were retro-fitted in the end.	**79%** 
Student 7	Pre	**Slightly below average**	Mostly. The logic is implemented correctly. The exempted items are hard coded. The tax % is hard coded as magic numbers.	7	200	2-37	1	101	2-36	Basic	Magic Number Comments Dead Code Duplicated code Primitive Obsession Long Method Switch Statements Conditional Complexity Data Class Indecent Exposure Temporary Field	Modular, but mostly procedural design. Obsessed with Getters/Setters. Quite a bit of duplication in code.	**8** 	7	7	Tests check all the end-to-end happy path cases. Poor test names. Assert Statement syntax is wrong.	**36%** 
	Post	**Slightly Below Average.** **Not much improvement since the pre-assessment.** **I don't think you've applied TDD principles.**	Partially. Lunch hour is hard-coded.	4	167	2-32	4	201	3-23	Basic	Magic Numbers Verbose Code Inappropriate Naming Comments Dead Code Duplicated code Primitive Obsession Large Class Lazy Class Long Method Switch Statements Black Sheep Conditional Complexity Indecent Exposure Temporary Field	OO design needs a lot of improvement. If you look at the solution, it contains the following 4 classes: Assistant (Main class, but contains only bunch of static methods) Calenders (Wrapper around a map. A wanna-be domain object) Day (Wrapper around free-slots. A wanna-be domain object) TimeDay (Data class, no logic) Lots of sysouts in the code. Obsessed with primitive types. setStatus method returns a boolean flag and also throws an exception.	**10** 	22	6	Most complex part of the code is not tested. Some tests are failing when I execute them. Tests are very verbose and contains a lot of duplication. Extremely hard to understand what the tests are trying to test.	**30%** 
Student 8	Pre	**Above Average**	Mostly. The logic is implemented correctly. The exempted items are hard coded. The tax % is hard coded as magic numbers.	7	92	1-6	3	55	1-12	Decent	Magic Number Comments Dead Code Duplicated code Lazy Class Data Class	Good OO design. Small well defined classes with small focused methods. Extremely low complexity. Can avoid some unnecessary gets. It breaks encapsulation. Overall best design so far.	**2** 	2	4	Tests check all the end-to-end happy path cases. Assert statement syntax is correct. Some duplication can be avoided. Overall, the best test cases so far.	**80%** 
	Post	**Above Average** **Nice improvements from the pre-assessment.** **Looks like you've applied a good number of TDD principles. More scope for improvement.**	Mostly.	4	119	2-11	4	120	2-39	Good	Magic Numbers Conditional Complexity	Good OO Design. Minor improvements possible.	**10** 	16	4	Good test coverage. Good test names, can slightly improve it. A huge method to set up the test data. It gets quite hard to understand why certain test results are showing up. The tests are a bit verbose/repetitive. One of the most complex method is not tested.	**86%** 
Student 9	Pre	**Below average** **Code is exactly same as Varun Bishnoi's code. Except a few classes were renamed.**	Partially. The logic is mostly correct, except the rounding logic. The exempted items are hard coded. The tax % is hard coded as magic numbers.	6	169	2-17	0	0	0	?	Magic Number Comments Dead Code Lazy Class Long Method Speculative Generality Feature Envy Data Class Temporary Field	Mostly procedural design. Data and logic in separate places. Lots of getters and setters. Unnecessarily created an Exception class.	**7** 	4	3	No Tests	**0%** 
	Post	**Slightly below Average.** **Great improvement compared to pre-assessment when we look at the testing side of things, however OO Design is pretty much the same.** **Large number of code smells in the code.** **Looks like you've applied some TDD principles, but there is a large scope for improvement.**	Mostly.	10	179	2-11	3	189	2-6	Decent. Scope for improvement	Magic Numbers Verbose Code Inappropriate Naming Comments Duplicated code Primitive Obsession Large Class Lazy Class Long Method Black Sheep Conditional Complexity Data Class Indecent Exposure	OO design needs significant improvement. Out of the 10 classes, the core logic is just in 1 static class. Employee (Data class with no behavior in it) 3 exception classes. Not sure why we need them. Just clutters the method signature every where. Also all of them have the same serialVersionUID. A classic example of IDE vomit. MyOffice (Data class - just wraps a map of employees) SchedulingAssistant (Contains the main logic. However all methods are static.) 3 Util classes: (Util classes are a good example of procedural code. Also not sure why these classes need to be constructed.)	**7** 	2	3	Good test names. Nice use of fluent interfaces to make the tests crisp and communicative. Code Coverage can be improved. Should stick to one scenario per test method. Also could look at using Parameterized Tests to run same scenario with multiple test data. Should avoid catching unexpected exceptions in the test. Ideally the test should fail if such exceptions occur. Some tests are failing when I execute them.	**63%** 

				Production Code			Test Code				OO Design Skill					Testing Skill	
Name	Assessment	Overall Score	Program Meets Requirements	# of Classes	Total Code Size	Method Length Range	# of Test Cases	Total Code Size	Method Length Range	Knowledge of xUnit	Code Smells in the code	Object Oriented Design Skill	Cyclomatic Complexity	NPath Complexity	Fan-Out Complexity	Quality of Unit Test	Test Coverage
Student 10	Pre	**Average**	Mostly. The logic is implemented correctly. Both the exempted items and tax % is being externalized to the DB. However its not clear how the taxes get assigned in the first place.	9	231	2-22	1	50	7-15	Good	Magic Number Comments Dead Code Duplicated code Large Class Lazy Class Long Method Speculative Generality Feature Envy Data Class Verbose Code	The design in very enterprisy (verbose.) Very procedural design with a bunch of data classes. Lots of getters and setters breaking encapsulation. Data and logic are not together. SalesTaxCalculator is a God object. The Item class is constructed with the applicable tax; breaks the temporal symmetry principle.	**16** 	128	11	Has written one test using mock objects, but only partially tests the core logic. Test has a huge setup highlighting problem with the design (tight coupling). Since we are testing from a higher point, the test does not communicate the intent very well.	**47%** 
	Post	**Average.** **Good improvement compared to the pre-assessment, but still there is a lot of room for improvement.** **Looks like you've at least applied some TDD principles. But there is a large scope for improvement.**	Mostly.	7	138	2-17	2	112	6-17	Decent. Scope for improvement	Verbose Code Dead Code Duplicated code Primitive Obsession Large Class Lazy Class Switch Statements Black Sheep Conditional Complexity Data Class Indecent Exposure Temporary Field	OO design needs a lot of improvement. If you look at the solution, it contains 7 classes, out of which only 2 make sense, rest of them are examples of poor OO. 4 Domain classes (Mostly lazy classes. They don't seem to be pulling their weight, except the RealClock) Calendar, Hour, HourStatus, RealClock Clock (Interface to abstract from system time) SystemUserdetailsProvider (Not sure of the value of this abstraction. Could have simply passed in the Calendar class) MeetingTimeSelector (Main class containing all the logic. Mostly a static class, except for the forced setter based dependency)	**14** 	64	6	Good test names. Code Coverage can be improved. Most complex piece of code is not tested. Test code is very verbose and contains lots of duplication. Nice to see you use Mockito, however I'm concerned that its not being used in the most effective manner. Test should not catch and suppress exceptions.	**54%** 
Student 11	Pre	**Below average**	The code is buggy it won't work. Has tried to externalize the exempted items to the DB, however the tax % are still hard-coded. Rounding logic is missing.	5	82	2-21	0	0	0	None	Dead Code Lazy Class Long Method Feature Envy Inappropriate Intimacy Data Class Indecent Exposure	Mostly procedural design. Data and logic in separate places. Lots of getters and setters. PriceCalculator is the God object which does all the calculation.	**4** 	5	4	Started to write a test but abandoned it mid way.	**0%** 
	Post	**Below Average.** **Some improvement since the pre-assessment. Good to see you write some unit tests. But you have a long way to go.** **I don't think you've applied TDD principles.**	No. I love the fact that you've tried to come up with the simplest possible solution for this problem. However I feel you've over-simplified the solution and its not usable any more.	2	40	4-14	1	36	3-15	Basic	Magic Numbers Verbose Code Inappropriate Naming Dead Code Primitive Obsession Lazy Class Conditional Complexity Data Class Indecent Exposure Temporary Field	OO design needs improvement. Out of the 2 classes, the core logic is just in 1 class. Calender class is just a lazy class which really can be replaced with a list. Scheduler class which contains bulk of the logic, is just a static class with one large method.	**11** 	28	2	Decent coverage. Test names can be improved. Lot of duplication in the test. Tests are very implementation specific, had to spend time understanding what the test is doing.	**93%** 
Student 12	Pre	**Below average** **Code is exactly same as Shambhu Singh's code. Except a few classes were renamed.**	Partially. The logic is mostly correct, except the rounding logic. The exempted items are hard coded. The tax % is hard coded as magic numbers.	6	167	2-17	0	0	0	?	Magic Number Comments Dead Code Lazy Class Long Method Speculative Generality Feature Envy Data Class Temporary Field	Mostly procedural design. Data and logic in separate places. Lots of getters and setters. Unnecessarily created an Exception class.	**7** 	4	3	No Tests	**0%** 
	Post	**Slightly below Average.** **Compared to the pre-assessment, there is improvement with regards to the unit tests. OO design pretty much the same.** **I think you've used very few TDD principles. Tests seemed to have been retro-fitted.**	Barely. Tried to simplify it by using boolean values to represent slots, but you lost me somewhere.	6	124	2-12	1	123	1-26	Basic	Magic Numbers Verbose Code Inappropriate Naming Comments Dead Code Duplicated code Primitive Obsession Large Class Lazy Class Long Method Black Sheep Conditional Complexity Data Class	OO Skills needs improvement. Obsessed with boolean arrays. CalendarPersistence - Place holder for persistence class. Failed attempt at making it Singleton. Not sure why. CommonSlotFinder - Basically a static helper class MeetingCalendar - Valid domain object with some logic. MeetingCalendarBusinessRules - Contains duplicated code which is already present in MeetingCalendar MyDate - Data class, no logic. SchedulingAssistant - Main class with one single, large complex method	**9** 	12	5	Good test coverage. Good test names, can be improved. A huge method to set up the test data. It gets quite hard to understand why certain test results are showing up. The tests are a bit verbose/repetitive. Tests heavily rely on matching boolean arrays, this can get very difficult to debug. Poor test data. Please use test data that is easy to understand.	**88%** 
Student 13	Pre	**Slightly below average**	Not really. Instead of figuring out exempted items, asks the user to enter them. Rest of the logic is implemented correctly except there is a small problem in the rounding logic. Tax % is being externalized to the DB.	6	141	2-14	1	58	2-12	Basic	Magic Number Comments Dead Code Primitive Obsession Lazy Class Speculative Generality Feature Envy Data Class Indecent Exposure	Mostly procedural design. Data and logic in separate places. Lots of getters and setters. SalesTaxCalculator is the God object which does all the calculation. Relies on primitives quite a bit. Tried to come up with a generic solution, which makes code today more complex than it needs to be.	**2** 	2	7	Tested the happy path of the basic logic. Does not know the basic use of Asset Statements	**68%** 
	Post	**Slightly below Average.** **Some improvement since the pre-assessment. But you have a long way to go.** **I don't think you've applied TDD principles.**	Partially.	3	81	2-11	2	45	3-17	Basic	Magic Numbers Verbose Code Duplicated code Primitive Obsession Large Class Lazy Class Long Method Black Sheep Conditional Complexity Data Class Temporary Field	OO design needs significant improvement. Out of the 3 classes, the core logic is just in 1 static class. Clock (Interface. But the production code does not contain any implementation of this interface.) MeetingAssistant (Contains the main logic. Mostly a static class with a lot of primitive obsession. Current design is flawed. The constructor leaves the object in un-stable state. Why did you choose to use setter based dependency injection for Clock? What if I don't call the set method?) User (Data class. No logic. Getters and Setters break encapsulation.)	**9** 	12	4	Decent coverage. Test names can be improved. Lot of duplication in the test. Test does not cover all the scenarios. Looks like the test were retro-fitted in the end.	**86%** 
Student 14	Pre	**Slightly below average**	Partially. The logic is implemented correctly. The exempted items are hard coded. The tax % is hard coded as magic numbers. Production code contains quite a bit of test code.	8	146	1-22	2	92	5-11	Basic	Magic Number Comments Dead Code Duplicated code Primitive Obsession Long Method Switch Statements Speculative Generality Conditional Complexity Combinatorial Explosion Data Class Indecent Exposure	Mostly procedural design. Data and logic in separate places. Lots of getters and setters. SalesTaxCalculator and ReceiptGenerator are the God classes which perform all the calculation.	**10** 	27	4	Tested the happy path of the basic logic. Does not know the basic use of Asset Statements Production Code has quite a bit of test code mixed up.	**62%** 

				Production Code			Test Code				OO Design Skill					Testing Skill	
Name	Assessment	Overall Score	Program Meets Requirements	# of Classes	Total Code Size	Method Length Range	# of Test Cases	Total Code Size	Method Length Range	Knowledge of xUnit	Code Smells in the code	Object Oriented Design Skill	Cyclomatic Complexity	NPath Complexity	Fan-Out Complexity	Quality of Unit Test	Test Coverage
Student 14	Post	**Average.** **Good improvement compared to the pre-assessment, but still there is a lot of room for improvement.** **Looks like you've at least applied some TDD principles. But there is a large scope for improvement.**	Mostly.	3	107	4-27	2	202	2-33	Decent. Scope for improvement	Verbose Code Inappropriate Naming Duplicated code Primitive Obsession Long Method Conditional Complexity	OO design needs improvement. MeetingScheduler (Static util class contains core of the logic. Contains one, large complex method) MeetingCalendar (Good domain object but contains a large complex method. Avoid using deprecated methods.) InvalidDateSlotException (Not sure why you need this exception class.)	**10** 	4	1	Good test coverage. Good test names. The tests are a bit verbose and not very communicative. Test code contains a MockedCalendar which is extremely complicated. Each test should set its value rather than setting up the whole MockedCalendar once at the beginning. Since the MockedCalendar is hard-coded with slots, its not easy to understand why certain tests are behaving the way they are. Quite confusing. When I run the tests, they are failing.	**88%** 
Student 15	Pre	**Slightly below average**	Partially. The logic is implemented correctly except the rounding off logic. Both exempted items and tax % is hard coded as magic numbers.	4	134	2-33	2	119	10-32	Basic	Inappropriate Naming Comments Dead Code Duplicated code Primitive Obsession Large Class Long Method Switch Statements Combinatorial Explosion Indecent Exposure Temporary Field	Moving towards an OO design, but still room for improvement. Main class is a God class. Need to break it down. Does not close files after using them. Quite a bit of primitive obession. Need to use emuns more effectively. Lots of getters and setters breaking encapsulation.	**7** 	18	9	Testsed the happy path scenarios. More end to end tests rather than unit tests. Tests don't work on my machine, sicne they have hard coded file paths. Tests are quite long and complex.	**79%** 
	Post	**Average.** **Good improvement compared to the pre-assessment, more room for improvement.** **Looks like you've at least applied some TDD principles. Again more room for improvement.**	Mostly.	4	126	2-19	1	104	2-8	Decent. Scope for improvement	Duplicated code Primitive Obsession Large Class Lazy Class Long Method Black Sheep Conditional Complexity Data Class Temporary Field	OO design can be improved. MeetingCalendar (Main domain object which contains bulk of the logic. Contains a very large, complex method. Needs to be simplified) MeetingTime (Data Class - has not logic.) Person (Except one method, this class is mostly a data class. Getters and setters break encapsulation should be avoided.) NoFreeSlotForMeetingException (Not sure what is the value of this Exception class.)	**11** 	2	1	Good test names. Code Coverage can be improved. Some complex piece of code is not tested. Test code is quite verbose and contains some duplication. It can be avoided. When I run the tests, they are failing. Since the calendars for the individuals is set in the set inside a separate method, its hard to understand why certain tests results are showing up.	**72%**