Klausur Sommersemester 2018, Fragen und Antworten

Einführung in die Softwaretechnik (IN0006) (Technische Universität München)

Introduction to Software Engineering (SS 2018)

Repeat Exam Solution

Grading Scheme:

Grade Intervall		Grade
0	17.5	5.0
18	20.5	4.7
21	23.5	4.3
24	26.5	4.0
27	29.5	3.7
30	32.5	3.3
33	35.5	3.0
36	38.5	2.7
39	41.5	2.3
42	44.5	2.0
45	47.5	1.7
48	50.5	1.3
51	60	1.0

Question 1 — Multiple Choice [6 points]

a) [1p] A UML activity diagram describes:

- [X] **The dynamic behavior of a system.**
- [ ] The functionality of a system.
- [ ] The functionality and the static structure of a system.
- [ ] The static structure of a system.
- [ ] The static structure and dynamic behavior of a system.

b) [1p] Which of the following statements is correct?

- [X] **<<extends>> models exceptional functionality.**
- [ ] <<includes>> is a base use case.
- [ ] <<extends>> models common functionality.
- [ ] <<includes>> describes additional behavior.
- [ ] <<extends>> is a base use case.

c) [1p] The teaching assistants discuss the exam in a meeting in Prof. Brügge's office. Which of the following aspects describe this communication event?

- [X] **Scheduled, synchronous and formal**
- [ ] Unscheduled, synchronous and informal
- [ ] Unscheduled, asynchronous and formal
- [ ] Unscheduled, asynchronous and informal
- [ ] Scheduled, asynchronous and informal

d) [1p] Which of the following statements is correct?

- [X] **Control objects represent the tasks to be performed by the system**
- [ ] Control objects represent the interaction between the user and the system.
- [ ] Boundary objects represent the persistent information tracked by the system.
- [ ] Boundary objects represent the tasks to be performed by the system.
- [ ] Entity objects represent the interaction between the user and the system.

e) [1p] The following terms hint at using design patterns. Which pattern is correctly identified?

- [X] **None of the mentioned**
- [ ] "extensible", "scalable" –> Composite
- [ ] "interface to an existing API" –> Bridge
- [ ] "set at startup time" –> Strategy
- [ ] "complex", "variable length & height" –> Facade
- [ ] All of the mentioned

f) [1p] Which of the following statements is correct? In object-oriented testing . . .

- [X] **Doubles replace the SUT's collaborators**
- [ ] None of the mentioned
- [ ] All of the mentioned
- [ ] The SUT initializes Mock Objects
- [ ] The System Model contains dummy objects
- [ ] SUT stands for System Unit Test

Question 2 — Project Management [9 points]

a) **[3p]** Compare the three different project organization types with their advantages and disadvantages.

Solution:

- **Functional Organization**: people are grouped into departments, each addressing an activity ("function")
  - **Advantages:**
    - Requires little communication, role definitions are clear, the more people on a project, the more the need for a formal structure
    - Members of a department have a good understanding of the functional area they support
  - **Disadvantages:**
    - It is difficult to make major investments in equipment and facilities
    - There is a high chance of overlap or duplication of work among departments.
- **Project-Based Organization**: people are assigned a project, addressing a problem to be solved within a specified time and budget
  - **Advantages**
    - Very responsive to new requirements (because the project is newly established and can be tailored specifically to address the problem)
    - New people can be hired who are familiar with the problem or who have special capabilities
  - **Disadvantages**
    - Teams cannot be assembled rapidly. Often it is difficult to manage the staffing/hiring process
    - Because there are „no predefined lines" (as in functional organizations), roles and responsibilities need to be defined at the beginning of the project.
- **Matrix Organization**: people from different departments of a functional organization are assigned to work on one or more projects, participants are usually assigned to a project < 100 % of their time.
  - **Advantages**
    - Teams for projects can be assembled rapidly from the existing line organization
    - Rare expertise can be applied to different projects as needed
    - Consistent reporting and decision procedures can be used for projects of the same type
  - **Disadvantages**
    - Participants are usually not familiar with each other
    - Participants have different working styles
    - Participants must get used to each other

Correction Criteria:
- + 0.5 points for each correct difference, correct advantage and correct disadvantage per organization type (max. 1 point per organization type).

b) **[2p]** Explain the differences between iterative, incremental, and adaptive development.

Solution:

- **Incremental** means to "add onto something"
  - Incremental development improves the process
  - No change during the project
- **Iterative** means to "re-do something"
  - Iterative development debugs and improves the product
  - Infrequent changes during the project
- **Adaptive** means to "react to changing requirements"
  - Adaptive development improves the reaction to changing customer needs

Correction Criteria:
- + 0.5 points for each correct difference including an explanation
- 0 points if the explanation in own words is missing

b) **[2p]** Explain the differences between defined and empirical process control.

Solution:
- **Empirical process control:**
  - **Summary:** Control software development through **agility**: not entirely planned / understood; **inspect and adapt**, see deviations as opportunities, expects the unexpected, Example: Scrum
  - Imperfectly defined process, not all pieces of work are completely understood
  - Given a well-defined set of inputs, different outputs may be generated when the process is executed
  - Control and risk management is exercised through frequent inspection
- **Defined process control:**
  - **Summary:** Control software development through a (supposed) **repeatable process**: planned, **follows strict rules**, avoids deviations, requires that everything is completely understood, Example: Waterfall Model
  - Given a well-defined input, the same output is generated every time
  - Precondition to apply this model: All activities and tasks are well-defined
  - If the preconditions are not satisfied: Lot of surprises, loss of control, wrong work products, ...
  - Conditions when to apply this model: Change can be ignored, the output is predictable

Correction Criteria:
- + 1 point for each correct difference including an explanation
- 0 points if the explanation in their own words is missing

c) **[2p]** Explain the differences between horizontal and vertical integration.

Solution:

- **Horizontal Integration**: Components are integrated into layers, following the subsystem decomposition. As development responsibilities also follow the subsystem decomposition, horizontal integration is straightforward to manage, as tests verify the interfaces that have been negotiated between teams. The main drawback, however, is that an operational system that can be a release candidate, is only available very late during development.
- **Vertical Integration:** Focus on early integration. For a given use case, the needed parts of each component, such [as] the user interface, business logic, middleware, and storage, are identified and developed in parallel and integration tested. […] A system build with a vertical integration strategy produces release candidates.

Correction Criteria:
- + 1 point for each correct difference including an explanation
- 0 points if the explanation in own words is missing

Question 3 — Requirements Analysis [9 points]

a) **[3p]** Extract requirements from the problem statement shown below. For both, non-functional requirements and constraints, also define the category (e.g. supportability). Identify at least 2 functional and 4 non-functional requirements / constraints.

Problem Statement:
"I want a web-based pizza order service (written in Javascript) that enables customers to choose between different kinds of pizzas (thick vs. thin dough), different sizes (large, medium, small) and different toppings (extra cheese, pineapple, bacon). The software should be easily extensible to support new pizza types. The customer must be able to order a pizza in three steps: once customers have selected the pizza, they specify a delivery address and proceed to the check-out to pay via PayPal. After successful payment, the pizza baker is notified and sets an approximate delivery time. At the end of the ordering process, the customer receives a notification with this delivery time."

Solution:

Functional requirements:
- customers can choose between different kinds, sizes, and toppings of pizzas
- the pizza baker sets an approximate delivery time
- the customer receives an email with the approximate delivery time

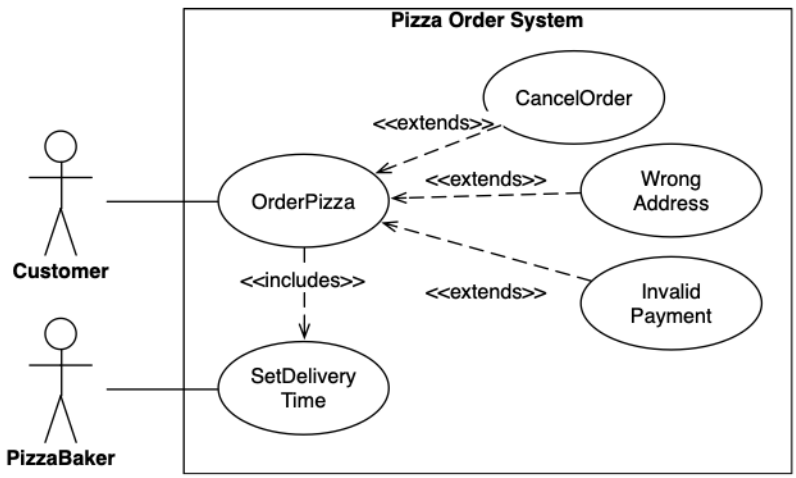
Non-functional requirements:
- Usability: customer must be able to order a pizza in three steps
- Supportability - Extensibility/Adaptability: extensible to support new pizza types
- Performance: receives directions to the customer on a smartphone within 20 seconds via a text message

Constraints:
- Implementation: written in Javascript
- Interface: payment method must be PayPal

Correction Criteria:
- + 0.5 points for each correct functional requirement
- + 0.5 points for each correct non-functional requirement including the correct category

b) **[3p]** Draw a UML use case diagram for a pizza order service. The system includes two actors: a `Customer` and a `PizzaBaker`. The diagram should include the use cases `OrderPizza` and `SetDeliveryTime`. Also model the following exceptions when ordering a pizza: `WrongAddress`, `InvalidPayment`, and `CancelOrder`.

Solution:



Correction Criteria:
- + 0.5 points for modeling the `OrderPizza` use case including the correct association to the Customer actor
- + 0.5 points for modeling the `SetDeliveryTime` use case including the correct association to the `PizzaBaker` actor and the `<<includes>>` relationship to `OrderPizza`
- + 1 point for modeling the three exceptions with `<<extends>>` relationships
- 0 points for the wrong notation or the wrong UML diagram type

c) **[3p]** Provide a textual description of the use case `OrderPizza` by filling out the following table. **Hint:** When you describe the event flow, make sure to use indentation to distinguish between the actor and the system steps.

Solution:

Use case name	OrderPizza
Participating Actors	Customer, PizzaBaker
Flow of events	1. The Customer selects the kind and the size of the pizza as well as its toppings.
	2. The Customer enters his delivery address.
	3. The Customer enters the PayPal details. He clicks on “Complete Order”.
	4. The PizzaService confirms the order and notifies the PizzaBaker.
	5. The PizzaBaker reviews the order and enters an approximate delivery time.
	6. The Customer receives a notification with the approximate delivery time.
Entry condition	The Customer has a valid address.
Exit condition	The money was transferred. The order is saved in the system and has the state “in progress”.
Special requirements	The “order confirmed” web page finishes loading 3 seconds after the Customer has clicked on “Complete Order”.

Correction Criteria:
- + 0.5 points for specifying steps 1 – 3 in the flow of events
- + 0.5 points for specifying step 4 in the flow of events
- + 0.5 points for specifying steps 5 and 6 in the flow of events
- + 0.5 points for filling out each of the other sections correctly (up to 1.5 points)
  - Entry condition, exit condition, and special requirements

Question 4 — System Design [8 points]

a) **[4p]** Compare the Pipes-and-Filter Architecture with the Layered Architecture. Describe real world examples for each architectural style and explain one design goal for each of these architectural styles.

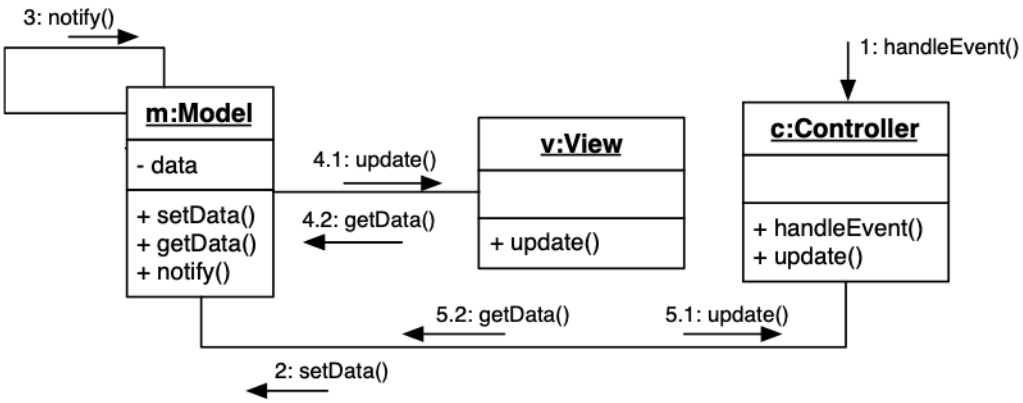
Solution:

Layered architectures are built hierarchical and make a distinction between each layer of subsystems and their access rights, whereas the **pipes-and-filters architectures** distinguish between two types of subsystems: pipes (input) and filters (output).
Design Goals Layered: reduce complexity, low coupling (closed), high coupling (open)
Design Goals Pipes-and-Filter: Parallelism (filters can be executed concurrently) and Flexibility (one filter can be substituted by another).
Example for Layered: ISO's OSI Reference Model
Example for Pipes-and-Filter: Unix Shell

Correction Criteria:
- + 1 point for each distinction including an explanation (max. 2 points)
- + 0.5 points for each correct design goal (max. 1 point)
- + 0.5 points for each good example (max. 1 point)

b) **[4p]** Create a UML communication diagram for the **pull notification variant** of the Model View Controller architectural style.

Solution:



Correction Criteria:
- + 0.5 points for each correct message (when the method for the message exists in the target object and the arrow has the right direction)
- 0 points for modeling the push notification variant
- 0 points for the wrong notation or the wrong UML diagram type

Question 5 — Object Design [12 points]

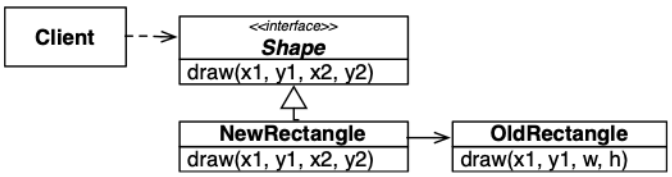
a) **[3p]** What are the different types of notifications in the observer pattern? Explain each type with one sentence.

Solution:

- **Push:** The Publisher notifies the subscribers about changes.
- **Push-Update:** The Publisher notifies the subscribers about changes including its new state.
- **Pull:** The subscribers ask the publisher if there is a change.

Correction Criteria:
- + 1 point for a good explanation for each type

Consider the following UML class diagram:



b) **[1p]** Which design pattern is used in the UML diagram above? Explain the purpose of this pattern.

Solution:

The Adapter pattern converts the interface of a legacy class (OldRectangle) into a different interface (Shape) expected by the client, so that the client and the legacy class can work together without changes.

Correction Criteria:
- + 1 point for correctly identifying the pattern and a good explanation
- 0 points if the explanation in own words is missing

c) **[3p]** Map the UML diagram shown above to Java code. Make sure to implement the draw method of `NewRectangle` properly.

Solution:

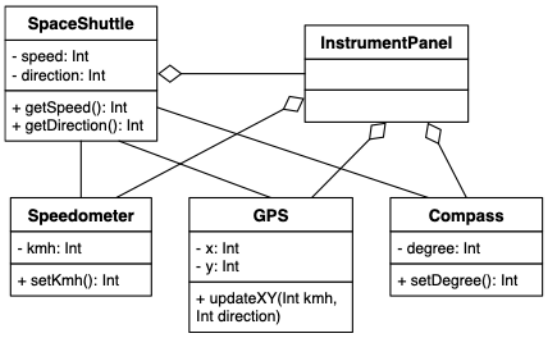
```
public interface Shape {
    void draw(int x, int y, int z, int j);
}

public class OldRectangle {
    public void draw(int x, int y, int width, int height) { … }
}

class NewRectangle implements Shape {
    private OldRectangle adaptee;

    public NewRectangle(NewRectangle rectangle) {
        this.adaptee = rectangle;
    }
```

```
    @Override
    public void draw(int x1, int y1, int x2, int y2) {
        adaptee.draw(x1, y2, x2-x1, y2-y1);
    }
}
```

Correction Criteria:

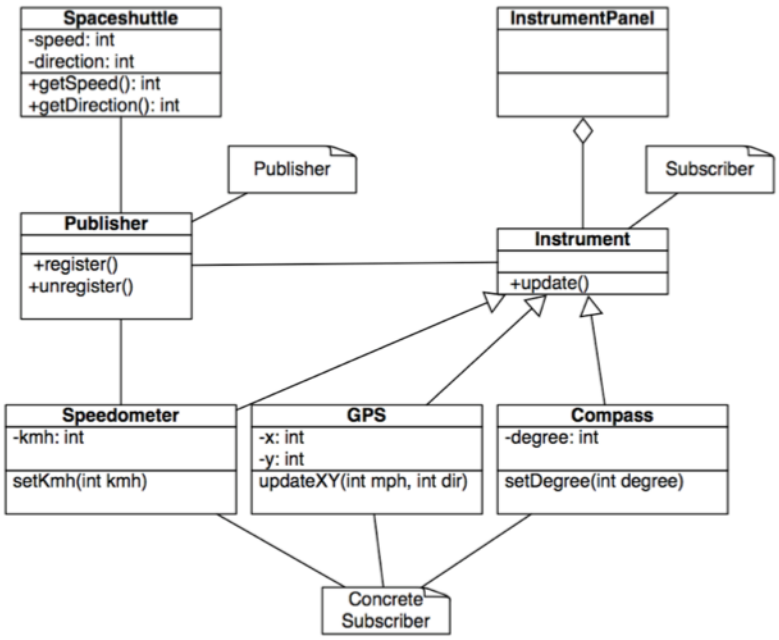
- + 0.5 points for the correct implementation of Shape
- + 0.5 points for the correct implementation of OldRectangle
- + 0.5 points for the correct class definition and the adaptee attribute in NewRectangle
- + 0.5 points for the NewRectangle constructor (or a mechanism to make sure that adaptee is **not** null)
- + 1 point for correctly overriding and implementing the NewRectangle's **draw** method

d) **[5p]** Review the object model shown below where the class `SpaceShuttle` needs to know all the instruments. Adding a new instrument requires to change the `SpaceShuttle` class. Explain why this is a bad design. Use the Observer Pattern to improve the design. Which non-functional requirements are now realizable? Draw a UML class diagram that improves the design and explain your changes.



Solution

- **Bad design**: functionality is spread all over the system, source code is hard to understand and complex, maintainer needs to understand the whole system to make a single change
- **Realizable NFRs** with observer pattern:
  - Adaptability: easy to add a new instrument because it only inherits from Instrument, no need to change another class
  - Maintainability: no need to know every single detail about the system any more, better structure, easier to find bugs



Correction Criteria:

- + 1 point for introducing the class Instrument (including all correct inheritance relationships)
- + 2 points for implementing the Publish-Subscriber-Pattern
- + up to 2 points for the following explanations:
  - + 0.5 points for explaining why this is a bad design (at least two valid reasons)
  - + 0.5 points for each realizable NFR
  - + 0.5 points for explaining the changes

Question 6 — Testing [4 points]

a) **[2p]** Explain the differences between system testing and integration testing.

Solution:

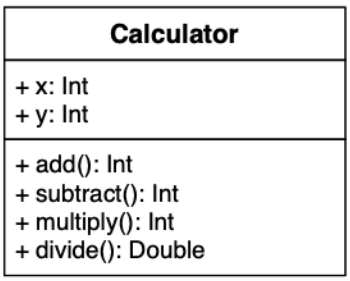
- **System testing:** The entire system is tested in the development environment to determine if the system meets the requirements
- **Integration testing:** Groups of subsystems are tested by developers to test the interfaces among the subsystems.

Differences:
- System testing tests the whole system while integration testing verifies the interfaces among subsystems
- System testing determines if the requirements are met, while integration testing verifies the APIs (contracts) of the different subsystems

Correction Criteria:
- + 1 point for each difference including an explanation

Consider the following object design model for a simple calculator app.

b) **[2p]** Write a JUnit for the method `divide()`. Before the unit test is executed, the input data has to be set. Apply good coding practices.



Solution:

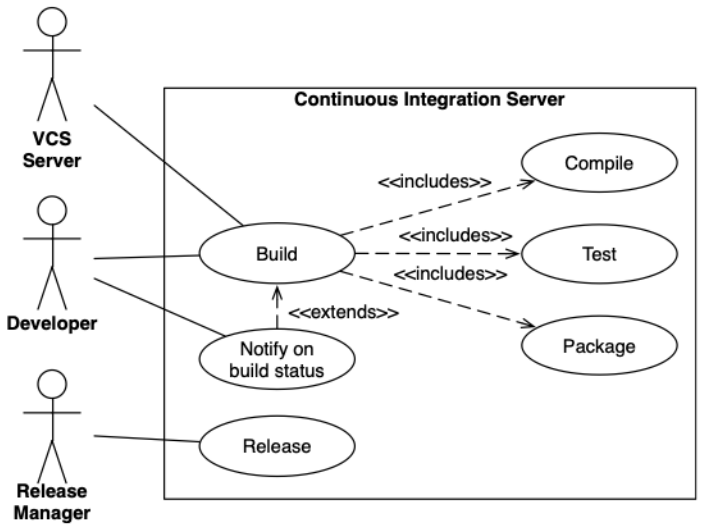
```
import org.junit.Test;
import static org.junit.Assert.*;
public class CalculatorTest {
      Calculator calculator = new Calculator();

      @Before
      public void setup() {
            calculator.x = 10;
            calculator.y = 2;
      }
      @Test
      public void testDivide() {
            double expectedResult = 5.0;
            double actualResult = calculator.divide();
            assertEquals(expectedResult, actualResult);
      }
}
```

Correction Criteria:
- + 1 point for the correct implementation of the testDivide() method
- + 1 point for correctly initializing the calculator (e.g. in a setup method with @Before)
- - 0.5 points if the annotation @Test is missing

Question 7 — Continuous Integration [12 points]

a) **[3p]** Create a functional model for continuous integration using UML. **Hint:** Model the continuous integration server as system and the developer, release manager and version control server as actors.

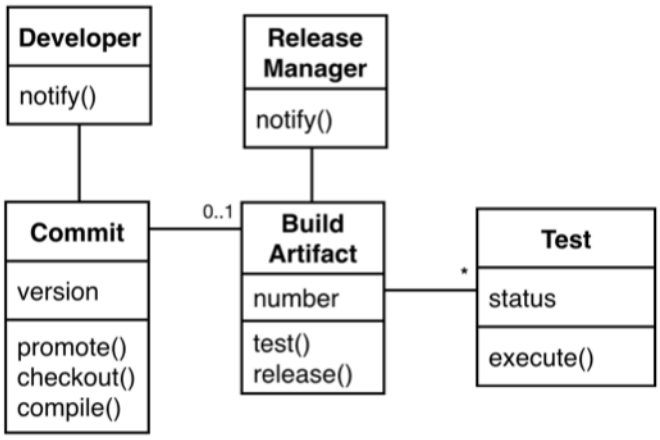
Solution:



Correction Criteria:
- + 0.5 points for each correctly modeled Continuous Integration activity as use case (including the correct association)
- 0 points for the wrong notation or the wrong UML diagram type

b) **[4p]** Create an analysis object model for continuous integration. **Hints:** Focus on the core concepts. Include attributes and methods in your model.

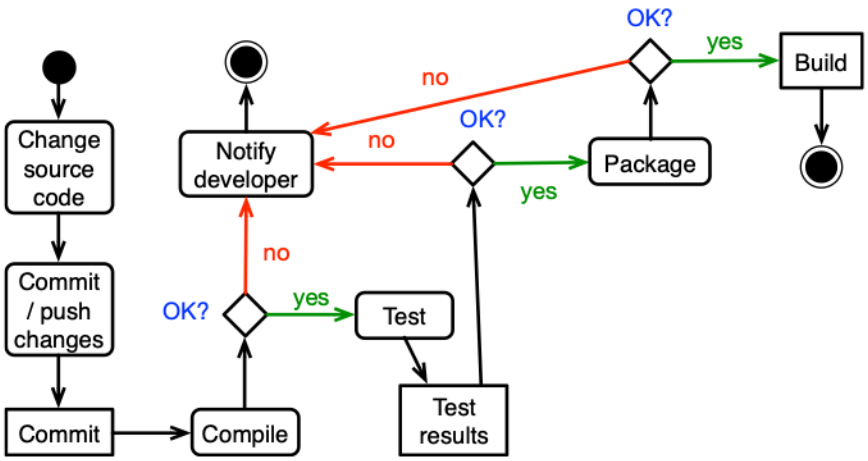
Solution:



Correction Criteria:
- + 1 point for every useful class (including at least one useful method, attribute and association)
- 0 points for the wrong notation or the wrong UML diagram type!

c) **[5p]** Model the dynamic behavior of continuous integration using a UML activity diagram.
Hint: Use decision nodes to model build failures.

Solution:



Correction Criteria:
- + 0.5 points for every correct activity, object and decision node (including the correct control flow)
- + 0.5 points for including start and end node(s)