***Belangrijk**: Schrijf je antwoorden kort en bondig in de daartoe voorziene velden. Elke theorie-vraag staat op X punten (totaal op X). De oefening staan in totaal op X punten. Het geheel staat op 40 punten.*

Vraag 1. Introduction [..../2]

Waarom is het "waterval" model onrealistisch (3 antwoorden is ok)?

- ***complete**: the customer is forced to define all the requirements soon and very explicitly (very difficult and in practice not feasable), future changes on system will be more difficult.*
- ***time**: only at the very end of development we have a working version of the product. This may cause the customer to change his mind (no feedbacks in the process, customers may not agree on the late delivery or worse they had another product in mind)*
- ***idealistic**: in real projects iteration occurs.*
- ***change**: difficult and costly to adapt later steps when requirements change.*

Waarom wordt het nog steeds gebruikt?

Popular because it is visible for upper management, easy to control project progress

Vraag 2. Project Management [..../2]

Waarom heeft het vervangen van een persoon in een ontwikkelingsproces een negatieve impact op de productiviteit?

First, the person that is leaving will have to do additional work(taking notes for the successor). Moreover, this persion is typically not motivated anymore.

Second, the replacement will have many questions in the beginning regarding the working of the system. These questions are often asked to co-workers, the newcomer bothers his/her colleages => productivity is negative.

Vraag 3. Use Cases [..../2]

Waarom zijn "use cases" goed geschikt voor gebruik in een iteratief/incrementeel ontwikkelingsproces?

(Use cases specify the requirements of a system, they focus on what a system should do.)

In iterative/incremental development the first iterations/increments can focus on only basic behaviour, for example only implementing the succes flow. The functionality can then be extended in a step-by-step fashion according to the iterations/increments.

First primary scenario, secondary scenarios later in process

Vraag 4. Domain Models [..../2]

Leg uit hoe domein models kunnen bijdragen voor het bereiken van correctheid van uw code (2 antwoorden).

Correctness

1) model the problem domain from the **customer perspective**

2) role-playing scenarios helps to **validate use cases**

    1) paper CRC cards are easy to reorganize

3) feature diagrams focus on commonalities/variations

    1) makes differences (and choices) explicit

Hoe zit het met traceerbaarheid(traceability)?

1) requirement <=> system via proper **naming conventions**, especially names of classes and operations

Vraag 5. Testing [..../2]

Leg telkens uit wat basic path testing, condition testing en loop testing is. Waarom zijn deze technieken complementair?

- basic path testing tests all possible paths **by total coverage of every statement, branch**.
- condition testing tests all **conditions by making them true or false**. (not all these possibilities were done by basic path testing)
- loop testing tests every loop by passing it (n is number of allowable passes)
    - **0, 1, 2,**
    - **m passes with 2 < m < n**
    - **n - 1, n, n +1 passes**

- (not all these possibilities were done by basic path testing and condition testing)

each technique tests a different aspect since not all posibilities were done by the previous technique(s) => complementary

Vraag 6. Design by Contract [..../2]

Wat is het *Liskov substitutie principe*?

a class must be replacable by any of its subclasses[1]

Waarom speelt dit een belangrijk rol binnen een object georiënteerde ontwikkeling?

precondition must be weaker or equal and postcondition stronger [1].

Vraag 7. Formal Specifications [..../2]

Geef drie argumenten *tegen* het gebruik van formele methoden. Geef telkens ook een tegenargument.

- high cost of specification BUT results better verifiable, lower cost implementation
- require highly trained staff BUT formal methods constists basically only of elementary math logic
- customers cannot understand specification BUT depends on customer and representation technique
- not applicable for all parts BUT maybe only for critical parts? (gui...)
- lack of tools BUT tool support is growing
- formal specifications don't scale well BUT same holds for programming languages

Vraag 8. Software Architecture [..../2]

Wat is een *pattern*?

the essence of a solution to a recurring problem in a particular context [1]

Leg de volgende zin uit in de context van het *Observer* pattern: “Beperkt de communicatie tussen *subject* en *observer*.”?

communication between the Model (Subject) and the Controller (Observer) is restricted through **only the update/notify methods** in the observer pattern[1]

Vraag 9. Quality Control [..../2]

Wat is het verschil tussen correctheid (correctness), betrouwbaarheid (reliability) en robuustheid (robustness)?

- correctness: a system is correct if it behaves according to its specification; this is an **absolute property**(i.e., a system cannot be “almost correct”; in theory and practice undecidable
- reliability: the system will operate as expected over a specified interval; **relative property**(e.g., a mean time to failure of 3 weeks)

- robustness: the system still behaves reasonably even in circumstances were not specified; **a vague property** because once you specify the abnormal circumstances, they become part of the requirements.

Vraag 10. Software Metrics [..../2]

Je bent een projectleider en wenst uw team te overtuigen om algoritmische kost modelling toe te passen binnen jullie projecten. Hoe leg je deze techniek uit aan je team?

choose a system model (0.5 each):

1) choose your **formula** with product and process attributes
2) **calibrate system model**: choose parameters for the parameters of the formula based on historical data
3) **Measure (or estimate) the attributes**: some are fixed, some may vary
4) **calculate cost/effort, ... and iterate...**

Vraag 11. Refactoring [..../2]

Geef 4 symptomen op die je met behulp van refactoring oplost. Geef telkens op welke specifieke refactoring je hier dan zou toepassen.

1. Duplicated code: extract code in new method
2. Nested conditionals: add/extract code in method using inheritance and polymorhism
3. Large classes/methods: split/add class/method
4. Abusive inheritance: add (sub)class to hierarchy

Vraag 12. Conclusion [..../2]

Als je het “No Silver Bullet” artikel hebt gelezen:

Waarom is programma verificatie geen silver bullet?

Programma verificatie levert geen foutloze programma’s + programma verificatie zegt enkel dat een programma voldoet aan zn specificatie. De specificatie kan fouten bevatten en bovendien nog incompleet zijn. Het moeilijkste aan het bouwen van software is echter het opstellen van een complete en consistente specificatie.

Als je het “Killer Robot” artikel hebt gelezen:

Werd code reviewing toegepast als deel van het kwaliteitsbewakings proces(quality assurance process)? Waarom/Waarom niet?

0,5: neen te weinig tijd, taken geminimaliseerd

0.5: code werd gereviewed

1: ja efficienter algoritme gevonden via code reviewing

2: ja maar niet goed, programmeurs stonden niet open voor code reviewing