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 is het noodzakelijk om taken klein te definiëren? Geef 2 redenen.
1) Better estimations [1 pnt]
2) Traceability between plan &requirements/system.[1 pnt]

Vraag 3. Use Cases [..../2]
Waarom zijn "use cases" goed geschikt voor gebruik in een iteratief/incrementeel ontwikkelingsproces?
Pagina 1/6
(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. [..../2]
Waarom is het noodzakelijk om de requirements te *valideren* én te *analyseren*?
valideren: zijn we het juiste systeem aan het bouwen [1] .....
analyseren: modeleren we het probleem domein juist [1] .....

Vraag 5.
Hoe verschillen basis path testing, conditiën testing en loop testing van elkaar? Leg uit.
Basis path zorgt dat elke statement + elke tak van control flow uitgevoerd/getest wordt, maar niet noodzakelijk alle entry-exit paden + niet alle paden mogelijk. Condition testing zorgt dat elke deel van een conditie getest wordt (elke true/false en gedeeltelijke true/false waarde-combinatie voor gehele en simpele condities.) Loop testing zorgt er dan weer voor elke loop getest wordt in verschillende aantallen(0,1,2,m, n-1, n, n+1)

Vraag 6. Design by Contract [..../2]
Wat is het *Liskov substitutie principe*?

Replacing an instance of a superclass by each of its subclasses is allowed. .....
. .....
Waarom speelt dit een belangrijk rol binnen het object georiënteerde paradigma en voor design contracten?
It defines the rules for inheritance and therefore subcontracts. .....

Vraag 7. [..../3]
Wat betekent het voor een *state-based* specificatie om *consistent*, *complete* en *unambiguous* te zijn?
a) *Consistent*: elke state is bereikbaar vanuit de initial state + vanuit elke staat ..... is de eindstaat bereikbaar [1] .....
b) *Complete*: elk event/state koppel heeft een transitie [1] .....
.....
c)*Unambiguous*: zelfde event (incl guard [-0,5]) komt niet voor op meer dan ..... 1 transitie uit een zelfde state [1] .....

Vraag 8.
Geef twee redenen waarom je **geen** *adapter/wrapper* zou introduceren in een design.
1. twee uit: performance overhead, maintenance overhead, ...
2. whole hierarchy needs adapting, merging in two directions needed .....

Vraag 9.
Waarom richten kwaliteitsstandaarden zich steeds op proces en interne attributen ipv. op de gewenste externe attributen?
Interne qualiteit leidt tot externe qualiteit + proces qualiteit leidt tot product qualiteit + externe qualiteit is pas meetbaar na afleveren van een product

Vraag 10.
Waarvoor worden metrieken(metrics) gebruikt? Geef een woordje uitleg.
Effort/Cost estimation: measure early to deduce later production efforts
Quality assessment en improvement: controleren qualiteitsattibuten tijdens ontwikkeling, vergelijken (en verbeteren) van software productie processen
Performance
Productiviteit
grootte

Vraag 11
Waarom is het niet mogelijk om een methode te verplaatsen in de inheritance hierarchie dmv. een pull-up refactoring wanneer deze een attribuut leest dat in dezelfde klasse gedefineerd is?
Dan zou een superklasse een attribuut gebruiken dat het niet heeft
Hoe kan je dit oplossen?
-Pas extract method toe op code fragment(en) die het attribuut accessen. Maak protected. Voeg abstracte versie van deze methode toe aan super. Nu kan je pull up doen.
-Pull up member

Vraag 12. Conclusion [..../2]
Als je het "No Silver Bullet" artikel hebt gelezen:

Waarom is *Object georiënteerd programmeren* geen "silver bullet"?
OO removes a higher order kind of accidental difficulty and allows a higher order expression of design. Nevertheless OO can do no more than to remove all accidental difficulties from the expression of the design. The complexity of the design itself is essential and such attacks make no change whatever in that.
Als je het "Killer Robot" artikel hebt gelezen:
Waarom was in dit specifieke geval het *waterfall process* zo rampzalig?
(From KillerRobotCase/articel-4.html) The waterfall model goes through definite stages of development. As the project passes from one stage to the next, there are limited opportunities to change earlier decisions. A drawback of this approach is that potential users are not able to interact iwth the sytem until very late in the process.
The Robot project involves a high degree of interaction, both between the robot components and between the robot and the operator. Since operator interaction with the robot is so important, the interface cannot be designed as an afterthought.