***Belangrijk**: Schrijf je antwoorden kort en bondig in de daartoe voorziene velden. Elke theorie-vraag staat op 2 punten (totaal op 24). De oefening staan in totaal op 16 punten. Het geheel staat op 40 punten.*

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

Wat is een *"Sprint"* in het *SCRUM* proces?

- Een sprint is een periode (typisch 2 tot 4 weken) waarin een aantal backlog items ontwikkeld worden.
- De sprint is de periode in het SCRUM proces waar er geprogrammeerd wordt.

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]

Geef twee voordelen en twee nadelen voor het gebruik van *Use Cases*.

Voordelen: [0.5 pnt per voordeel]

- requirements more understandable (actors provide end users perspective)
- requirements more precise. (scenarios are sufficiently detailed to test)
- requirements open. (Actors perspective emphasizes the what (and much less the how)
- Helps to validate solution against requirements
- Helps to verify the requirements against users needs

Nadelen: [0.5 pnt per nadeel]

- requires close interaction with various stakeholders
- needs iterations to improve earlier misconceptions
- a lot of hard work
- Use cases tend to result in hard to maintain systems
- Identifying actors and use cases may omit requirements (Completeness not guaranteed)
- Focus on scenarios restricts evolving requirements.

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

Leg uit wat *Object-georiënteerde en functionele decompositie* is en leg uit wanneer je ze zou gebruiken.

Functionele decompositie: Decompose according to the functions a system must perform. --> single "subfunction-of" hierarchy. [0.5 pnt voor de wat]

Good with stable requirements or single function. [0.5 pnt voor gebruik]

Object-georiënteerde decompositie: Decompose according to the objects a system must manipulate --> several coupled "is-a" hierarchies. [0.5 pnt voor de wat]

Better for complex and evolving systems [0.5 pnt voor gebruik].

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

Wat is het verschil tussen *Stress-Testing* en *performance Testing*?

Stress-Testing [1pnt]: Tests exrteme conditions- tries to break the system.

Performance testing [1pnt]: Test run-time performance in normal conditions, time consumption memory consumtion. Gaat na of systeem performant genoeg is voor requirements.

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

Wat is het onderscheid tussen *Design by Contract* en *Testing*?

Design by contract prevents defects [0.5 pnt]

Testing detects defects [0.5 pnt]

Geef twee redenen waarom dit *complementaire* technieken zijn:

- [0.5 pnt] Design by contract can be used in Equivalence Partitioning & Boundary Value Analysis (Black box testing) to determine classes of input data & prediction of corresponding output.
- [0.5 pnt] (Condition) Testing used to verify whether parties satisfy their pre- & postconditions (design by contract)

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

Wat is het onderscheid tussen een *semi-formele specificatietaal* en een *formele specificatietaal*?

Semi-Formal [0.5 pnt]: Notation with precise syntac but loose semantics.

Formal [0.5 pnt]: Model with precise syntax & semantics

Geef voor elk een voorbeeld:

semi-formele specificatie taal: [0.5 pnt] UML class &sequence diagrams, ...

formele specificatie taal: [0.5 pnt] Z, B, VDM, OCL, Petri-nets, StateCharts,....

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

Wat is een *pattern*?

[1 pnt] The essence of a solution to a recurring problem in a particular context.

Geef 2 redenen waarom dit bruikbaar is om een *architectuur* te beschrijven:

1) [0.5 pnt] Experts recall a similar solved problem and customize the solution

2) [0.5 pnt] Patterns document existing experience

3) [0.5 pnt] The context of a pattern states when and when not to apply the solution

4) [0.5 pnt] A pattern lists the tradeoff's involved in applying the solution

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

Hoewel we hoofdzakelijk geïnteresseerd zijn in "external product quality" focussen alle kwaliteitsstandaarden bijna uitsluitend op "process" en "internal quality attributes". Waarom?

Als de kwaliteit van het product slecht is is het te laat om er nog iets aan te verbeteren.

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

Noem de 4 *scale types* en geef voor elk een voorbeeld:

1) [0.5 pnt] Nominal: {male, female}, {dogs, cats, rabbits, fish, cows}

2) [0.5 pnt] Ordinal: hardness, {trivial, minor, normal, major, critical, blocker}

3) [0.5 pnt] Interval: calendar time, centigrade temperature

4) [0.5 pnt] Ratio: weight measured in grams, length measured in meters

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

Leg uit hoe *refactoring* verschilt van *gewoon programmeren*.

[1 pnt] Refactoring does not change the observable behavior of a system.

Geef 2 redenen *waarom* je refactoring zou toepassen?

[0.5 pnt] To increase understandability,

[0.5 pnt] Lower Complexity,

[0.5 pnt] make it easier to adapt

[0.5 pnt] make it easier to debug

[0.5 pnt] improve internal structure of a system

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.

Vraag 13. Oefeningen [..../6]

Jouw ontwikkelingsteam wordt aangenomen door een hotel om een systeem dat hun kamers en gasten beheert te ontwerpen en te implementeren.

Het hotel wil een systeem waarmee ze ondermeer:
- Kamers kunnen toevoegen en verwijderen.
- Een gast kunnen inchecken. Deze functie krijgt als input een gast en een kamer.
- Een gast kunnen uitchecken. Deze functie krijgt als enige input de gast.
- Kunnen opvragen hoeveel kamers er bezet zijn.

Je collega's hebben reeds een gedeeltelijke analyse van het domein gedaan, maar weten echter niet zo veel van formele specificatietechnieken. Vandaar dat jij nog een deel van het werk moet doen.

Meer bepaald moet je de hieronder beschreven Z schema's nakijken en aanvullen waar er "..." staat (als je denkt dat er niets moet staan laat je het open).
Op de koop toe hebben ze gevraagd om nog enkele schema's volledig uit te werken.

We introduceren de basistypes *ROOM* en *PERSON* die respectievelijk de sets van alle kamers en van alle personen voorstellen.

[ROOM, PERSON]

(In hetvolgende schema maken we gebruik van de notatie $\mathbb{F}$ die staat voor finite set. Dit is gelijkaardig aan de notatie $\mathbb{P}$ alleen hebben we het nu over eindige verzamelingen, dus verzamelingen waarvan we het totaal aantal elementen kan tellen. Het aantal elementen van zo een verzameling duidt men trouwens aan met het symbool "#"
rooms: $\mathbb{F}$ ROOM is dus een eindige verzameling van kamers en het totaal aantal kamers is #rooms.)

Hotel

$rooms : \mathbb{F}\ ROOM$
$guests : \mathbb{F}\ PERSON$
$occupies : PERSON \rightarrowtail\!\!\!\!\!\!+\ ROOM$
$nrOfFreeRooms : \mathbb{N}$

$\mathrm{dom}\ occupies = guests$
$\mathrm{ran}\ occupies \subseteq rooms$

AddRoom

$\Delta Hotel$
$room? : ROOM$

$room? \notin Hotel.rooms$
$Hotel.rooms' = Hotel.rooms \cup \{room?\}$
$Hotel.nrOfFreeRooms' = Hotel.nrOfFreeRooms + 1$

RemoveRoom

$\Delta Hotel$
$room? : ROOM$

$room? \in Hotel.rooms$
$Hotel.rooms' = Hotel.rooms \setminus \{room?\}$
$Hotel.nrOfFreeRooms' = Hotel.nrOfFreeRooms - 1$

CheckIn

$\Delta Hotel$
$room? : ROOM$
$guest? : PERSON$

$room? \in Hotel.rooms$
$guest? \notin Hotel.guests$
$Hotel.nrOfFreeRooms > 0$
$Hotel.occupies' = Hotel.occupies \cup \{guest? \mapsto room?\}$
$Hotel.nrOfFreeRooms' = Hotel.nrOfFreeRooms - 1$

CheckOut

$\Delta Hotel$
$guest? : PERSON$
$tmp : ROOM$

$guest? \in Hotel.guests$
$tmp = Hotel.occupies(guest?)$
$Hotel.occupies' = Hotel.occupies \setminus \{guest? \mapsto tmp\}$
$Hotel.nrOfFreeRooms' = Hotel.nrOfFreeRooms + 1$
$Hotel.guests' = Hotel.guests \setminus \{guest?\}$

QueryOccupiedRooms

$\Xi Hotel$
$nrOfRooms! : \mathbb{N}$

$nrOfRooms! = \# \operatorname{ran} Hotel.occupies$
OF
$nrOfRooms! = \# Hotel.rooms - Hotel.nrOfFreeRooms$

Vraag 14. Oefeningen - Project Management [..../7]

Gegeven de *Pert-chart* op de volgende pagina:

a) Vul voor elke node in de *Pert-chart* de *earliest start date* in (uitgedrukt in weeknummers). Beschrijf kort hoe je dit berekend hebt: ....

....

....

....

....

b) Vul voor elke node in de *Pert-chart* de *latest end date* in (uitgedrukt in weeknummers). Beschrijf kort hoe je dit berekend hebt. ....

....

....

....

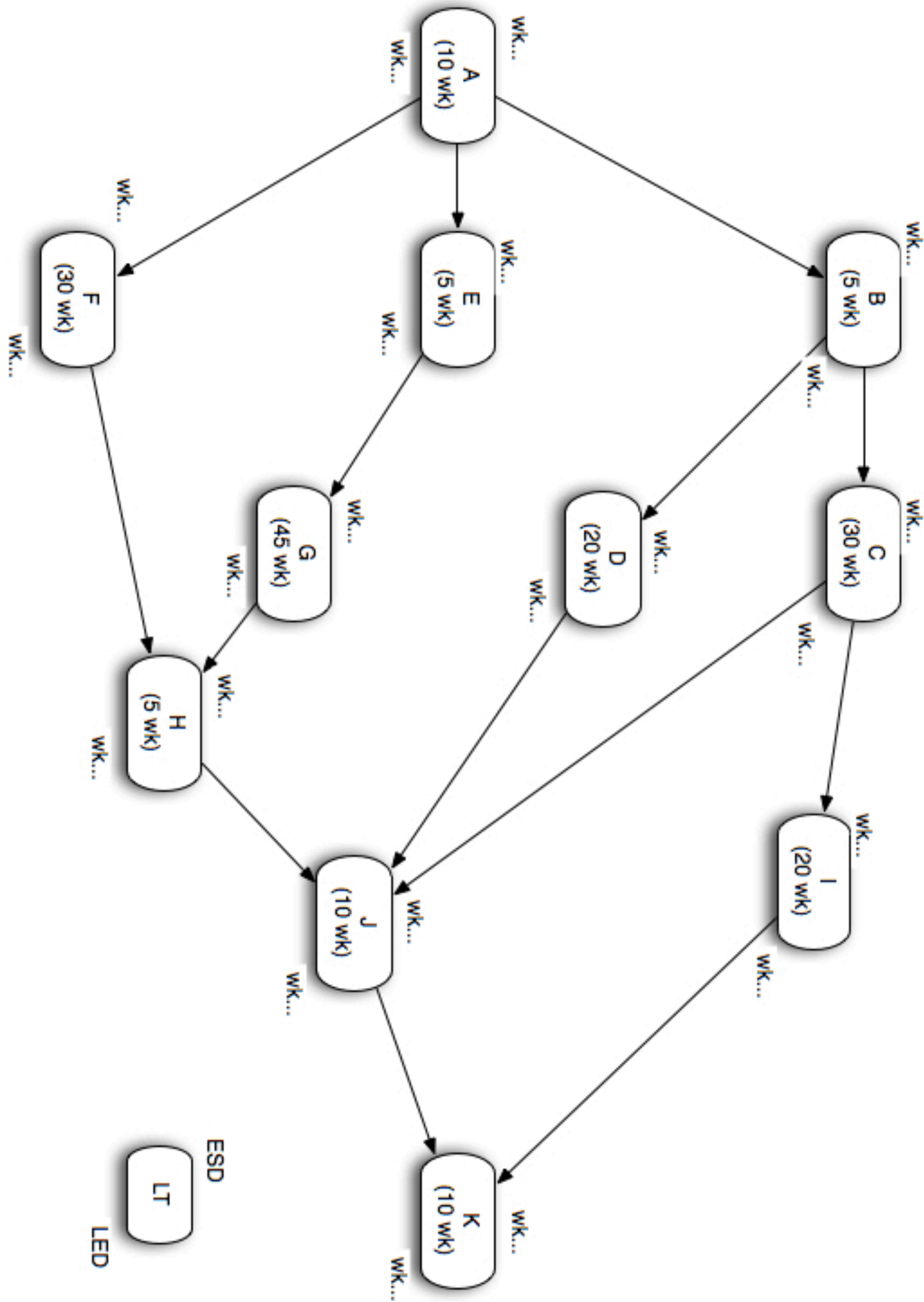
....

c) Duid op de figuur het *critical path* aan.

Gegeven het schema op pagina 9. Bereken:

d) het *risky path*: ....

e) de *worst case delay* van het project: ....


A (10 wk)
B (5 wk)
C (30 wk)
D (20 wk)
E (5 wk)
F (30 wk)
G (45 wk)
H (5 wk)
I (20 wk)
J (10 wk)
K (10 wk)
ESD
LT
LED

	OT	LT	PT	ET	S_nominator	S_denominato	S	Path	SP	Worst case delay
A	8	10	12			6		A		
B	4	5	13			6		A-B		
C	22	30	42			6		A-B-C		
D	18	20	21			6		A-B-D		
E	3	5	6			6		A-E		
F	29	30	32			6		A-F		
G	40	45	57			6		A-E-G		
H	5	5	5			6		A-E-G-H		
								A-F-H		
I	18	20	26			6		A-B-C-I		
J	7	10	12			6		A-B-C-J		
								A-B-D-J		
								A-E-G-H-J		
								A-F-H-J		
K	9	10	11			6		A-B-C-I-K		
								A-B-C-J-K		
								A-B-D-J-K		
								A-E-G-H-J-K		
								A-F-H-J-K		

Vraag 15. Oefeningen - Design by Contract [..../3]

1) `ChemicalTank` is een klasse die de basisfunctionaliteit van een chemische tank met 2 kleppen (valves) bevat: 1 klep bovenaan om de tank te vullen en 1 onderaan om de tank te ledigen. Ze heeft ook een sensor die vaststelt of de tank vol is of niet. Beide kleppen mogen niet tegelijkertijd open staan.

`ChemicalTank` bevat onderstaande methoden (met omschrijving). Stel voor de onderstaande 4 methoden pre- en postcondities op. Dit mag in het Nederlands.

- `isTankFull()` ⇒ checkt of de tank vol is

....

....

....

- `closeBottomValve()` ⇒ Sluit de onderste klep van de chemische tank na het lozen van een bepaalde hoeveelheid product.

....

....

....

....

....

....

- `closeTopValve()` ⇒ Sluit de bovenste klep van de chemische tank.

....

....

....

....

....

....

- `openBottomValve()` ⇒ Opent de onderste klep van de chemische tank.

....

....

....

....

....

....

- `openTopValve()` ⇒ Opent de bovenste klep van de chemische tank. Enkel mogelijk wanneer de tank niet vol is.

....

....

....

2) Stel je volgende “schets” van een klasse telefoon voor:

```
class Telephone
@pre: connectedToWiredNetwork()
➔ call(number)

➔ connectedToWiredNetwork()
```

Later komt er een klasse `Cellphone`, afgeleid van de klasse `Telephone`. Bedenk een geschikte preconditie voor de methode `call(number)` van de klasse `Cellphone`, dewelke `call(number)` van `Telephone` vervangt (override). Je mag extra methoden bedenken om je preconditie samen te stellen.

....

....

....

....

....

....

....

....

....

....

....

....