Naam:.....

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

1. Introduction [... / 2]

   Geef de 6 principes van *Extreme Programming*.

   - *Fine scale feedback*
   - *Continous process*
   - *Shared Understanding*
   - *Programmer welfare*
   - *Coding -> first unit test then code*
   - *Testing -> all code must have tests, acceptance tests are run often*

2. Project Management [... / 2]

   Wat is een "*known known*", een "*known unknown*" en een "*unknown unknown*"?

   Known knowns: *The things you know you know, you can safely make assumptions here.*

   Known Unknowns: *The things you know, you don't know. You can prepare for these during planning.*

   Unknown unknowns: *The things ou do not know, you don't know. These you cannot prepare for during planning. The best you can do is being aware and spot opportunities and do a thorough risk analysis.*

3. Use Cases [... / 2]

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

   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

4. Domain Models [... / 2]

   Wat specifiëren "*feature models*"?

   define a set of reusable and configurable requirements for specifying the systems in a domain

5. Testing [... / 2]

   Wat is het verschil tussen "*stress-testing*" en "*performance testing*"?

   Stress-Testing: *Tests extreme conditions - tries to break the system.*

   Performance testing: *Test run-time performance in normal conditions, time consumption memory consumtion. Checks if the system is performant enough according to the requirements.*

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

   Wat is het *Liskov substitutie principe*?

   if S is a subtype of T, then objects of type T may be replaced with objects of type S (i.e., objects of type S may substitute objects of type T) without altering any of the desirable properties of that program (correctness, task performed, etc.).
   Each instance of a superclass may be replacing by instances of a superclass without altering the correct behaviour.

   Waarom speelt dit een belangrijke rol binnen het object georiënteerde paradigma en voor design contracten?

   It defines the rules for inheritance and therefore subcontracts.

7. Formal Specifications [... / 2]

   Geef 3 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 consists basically only of elementary math logic so amount of training is limited*
   - *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*

8. Software Architecture [... / 2]

   Definieer de volgende begrippen uit de ATAM terminologie:

   Sensitivity point: *A sensitivity point is a property of one or more components (and/ or component relationships) that is critical for achieving a particular quality attribute response.*
   Tradeoff point: *A trade-off point involves two (or more) conflicting sensitivity points.*

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.*

10. Software Metrics [... / 2]

Geef 3 mogelijkheden om de grootte van een software product te meten.

- *Lines of code*
- *Use case points*
- *Function points*

11. Refactoring [... / 2]

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

- *Duplicated code: extract code in new method*
- *Nested conditionals: add/extract code in method using inheritance and polymorhism*
- *Large classes/methods: split/add class/method*
- *Abusive inheritance: add (sub)class to hierarchy*

12. Conclusion [... / 2]

Als je het "No Silver Bullet" artikel hebt gelezen:

Waarom is programma verificatie geen silver bullet?

Programma verificatie doet iets aan de accidentele complexiteit : nagaan of een programma voldoet aan zijn specificatie. Maar de specificatie kan fouten bevatten en bovendien nog incompleet zijn. De essentiële complexiteit (= hetgeen inherent moeilijk is aan het bouwen van software) is net 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?

Waarom wel/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

13. Oefeningen – Project Management [... / 5]

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

a) Bereken voor elke node de *earliest start date* en geef hieronder de gebruikte formule.

For each task n: compute earliest start-date
= Latest of all incoming paths
ESD (n) := latest of (ESD (preceding) + estimated time (preceding))

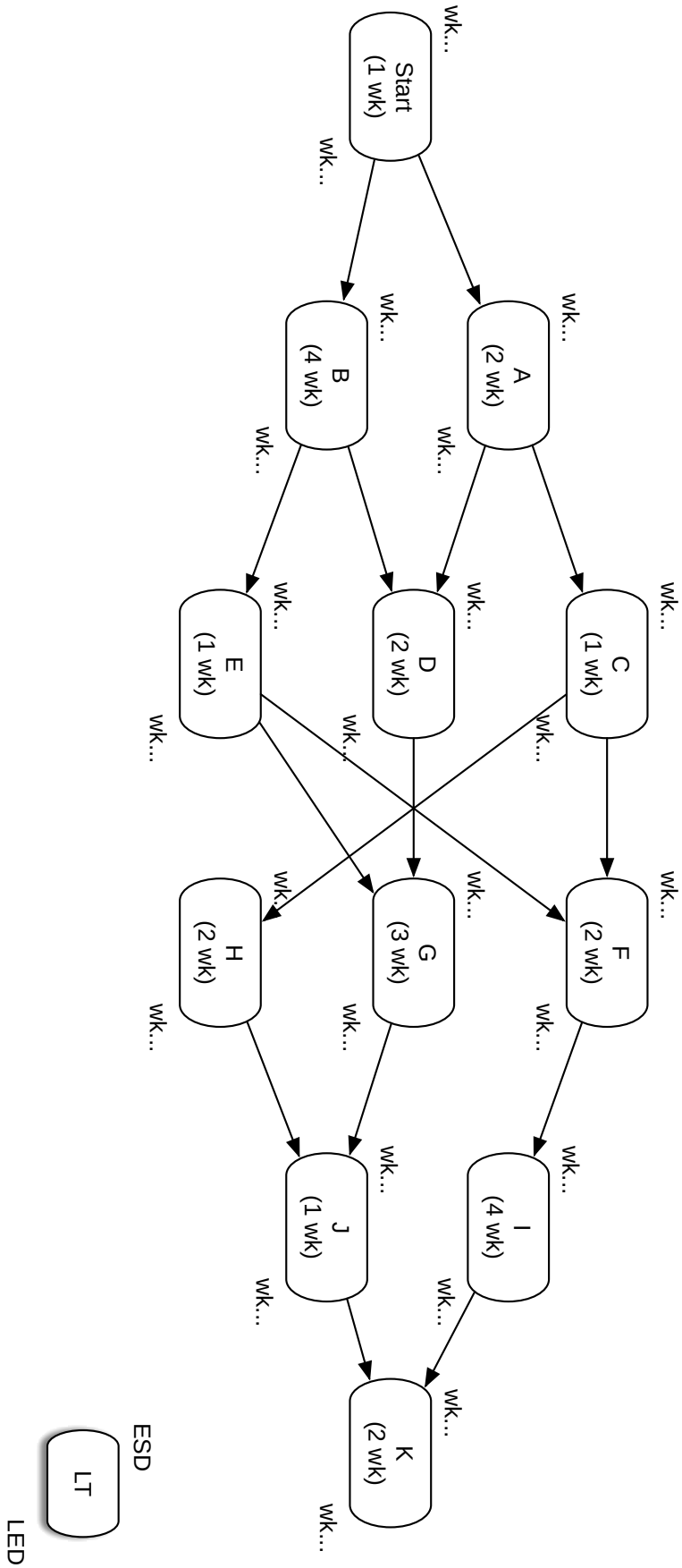
b) Bereken voor elke node de *latest end date* en geef hieronder de gebruikte formule.

For each task n: compute latest end-date
= Earliest of all outgoing paths
LED (n):= earliest of (LED (subsequent) - estimated time (subsequent))

c) Wat is het *critical path*? Let uit waarom.

Critical path: Start-B-E-F-I-K
LED(n) = ESD(n) + duration(n) for every node n on this path

d) Bepaal het *risky path* en *worst case delay* van dit pad door het schema op pagina 7 in te vullen.


Start
(1 wk)
A
(2 wk)
B
(4 wk)
C
(1 wk)
D
(2 wk)
E
(1 wk)
F
(2 wk)
G
(3 wk)
H
(2 wk)
I
(4 wk)
J
(1 wk)
K
(2 wk)
LT
ESD
LED
wk...

	OT	LT	PT	ET	S_nominator	S_denominator	S	Path	SP	Worst case delay
Start	1	1	1	1	0	6	0,00	Start	0,00000	0
A	1	2	2	2	1	6	0,17	Start-A	0,16667	0
B	4	4	5	4	1	6	0,17	Start-B	0,16667	1
C	1	1	1	1	0	6	0,00	Start-A-C	0,16667	0
D	1	2	4	2	3	6	0,50	Start-A-D	0,52705	3
E	1	1	2	1	1	6	0,17	Start-B-D	0,52705	3
F	2	2	4	2	2	6	0,33	Start-B-E	0,23570	2
G	2	3	4	3	2	6	0,33	Start-A-C-F	0,37268	2
H	1	2	3	2	2	6	0,33	Start-A-C-H	0,37268	1
I	2	4	6	4	4	6	0,67	Start-A-D-G	0,62361	1
J	1	1	2	1	1	6	0,17	Start-B-D-G	0,62361	4
K	2	2	2	2	0	6	0,00	Start-B-E-F	0,40825	4
								Start-B-E-G	0,40825	3
								Start-A-C-F-I	0,76376	4
								Start-A-C-H-J	0,40825	2
								Start-A-D-G-J	0,64550	2
								Start-B-D-G-J	0,64550	5
								Start-B-E-F-I	0,78174	6
								Start-B-E-G-J	0,44096	4
								Start-A-C-F-I-K	0,76376	4
								Start-A-C-H-J-K	0,40825	2
								Start-A-D-G-J-K	0,64550	2
								Start-B-D-G-J-K	0,64550	2
								Start-B-E-F-I-K	0,78174	6
								Start-B-E-G-J-K	0,44096	4

Risky path
Worst case delay

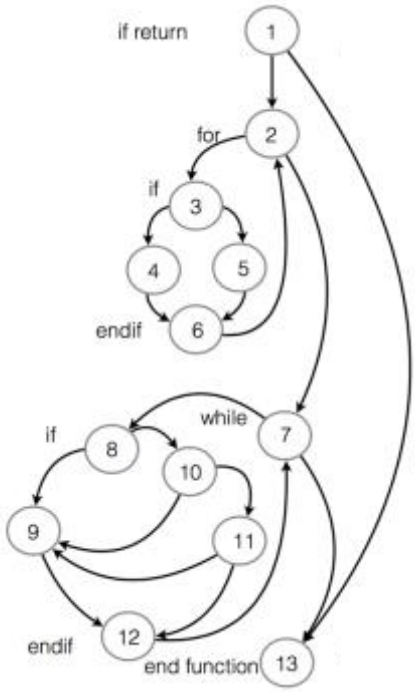
weeks

14. Oefeningen – Testing [... / 5]

Beschouw de onderstaande functie die een gegeven array sorteert door middel van Mergesort.

```
1  void mergeSort(int[] data) {
2    if(data.length <= 1)
3       return;
4
5    int[] a = new int[data.length / 2];
6    int[] b = new int[data.length - a.length];
7    for(int i = 0; i < data.length; i++) {
8       if(i < a.length)
9          a[i] = data[i];
10      else
11         b[i - a.length] = data[i];
12   }
13
14   mergeSort(a);
15   mergeSort(b);
16
17   int ai = 0;
18   int bi = 0;
19   while(ai + bi < data.length) {
20      if(bi >= b.length||(ai < a.length && a[ai] < b[bi])) {
21         data[ai + bi] = a[ai];
22         ai++;
23      } else {
24         data[ai + bi] = b[bi];
25         bi++;
26      }
27   }
28 }
```

a) Teken de *flow graph* voor bovenstaande functie.



b) Bereken de cyclomatische complexiteit en geef kort aan hoe je hiertoe gekomen bent.

#edges - #nodes + 2 = 19 – 13 + 2 = 8
= number of binary conditions + 1 = 7 + 1
= #regions = 8

c) Bepaal een volledige verzameling onafhankelijke paden. (Nummer ze)

(a) {1, 13}
(b) {1, 2, 7, 13}
(c) {1, 2, 3, 4, 6, 2, 7, 13}
(d) {1, 2, 3, 5, 6, 2, 7, 13}
(e) {1, 2, 7, 8, 9, 12, 7 13}
(f) {1, 2, 7, 8, 10, 9, 12, 7 13}
(g) {1, 2, 7, 8, 10, 11, 9, 12, 7 13}
(h) {1, 2, 7, 8, 10, 11, 12, 7 13}

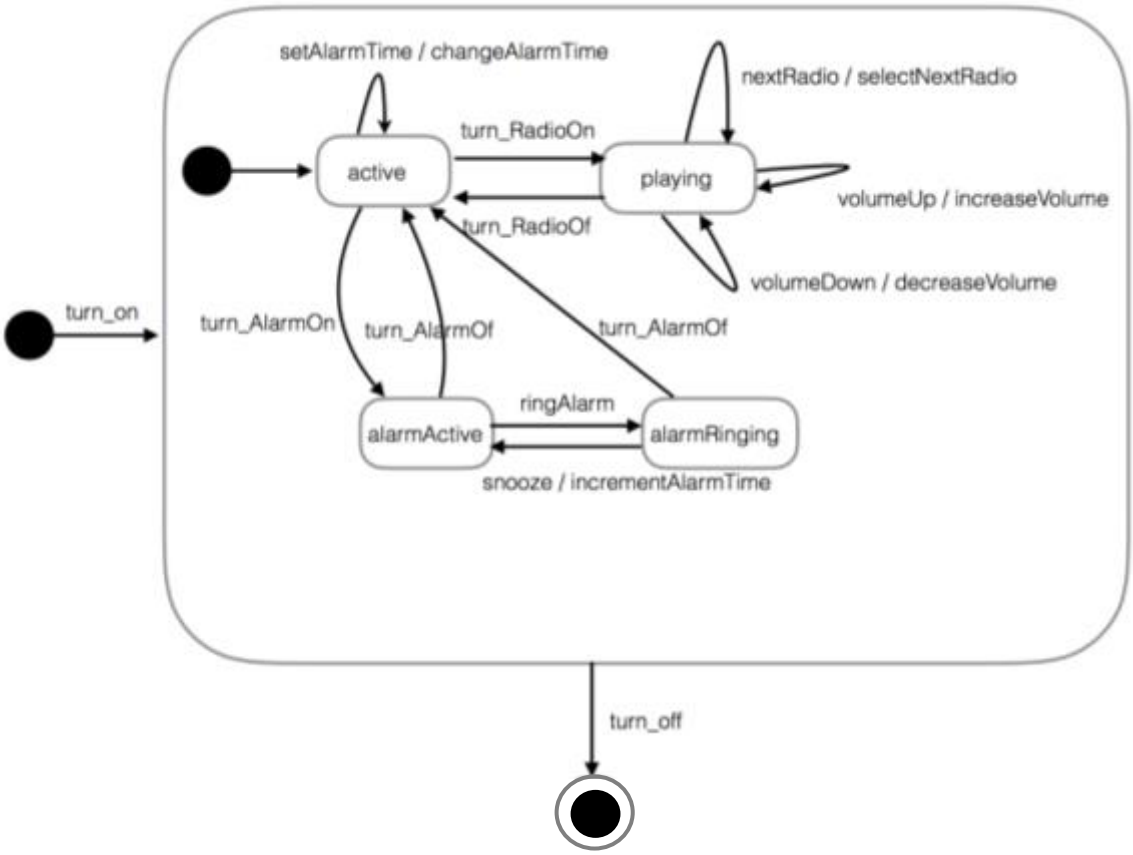
d) Bepaal 3 test cases (input en verwachte output) die een uitvoering van alle onafhankelijke paden garanderen.

Input: [1] Output: [1]
Input: [1,2] Output: [1,2]
Input: [2,1] Output: [1,2]

15. Oefeningen – Formal Specifications [... / 6]

Modelleer met behulp van *statecharts* de werking van een wekkerradio. Teken je oplossing op de volgende pagina. De wekkerradio moet onderstaand gedrag vertonen:

a) De gebruiker kan de radio op- en afzetten.
b) De radio heeft 3 ingebouwde zenders waartussen de gebruiker kan kiezen, hiervoor heeft de gebruiker een knop "*next_radio*", welke ervoor zorgt dat de volgende radio gekozen wordt. Willekeurig verspringen van zenders is niet mogelijk.
c) Wanneer de radio begint te spelen zal automatisch de laatst gekozen zender gebruikt worden.
d) De gebruiker kan met 2 knoppen "*volume_up*" en "*volume_down*" het volume wijzigen.
e) De gebruiker kan de wekker instellen.
f) De gebruiker kan de wekker aan of uitschakelen.
g) Indien de radio reeds aan het spelen is zal de wekker niet aflopen.
h) Wanneer de wekker afloopt:
    a. Begint de radio te spelen.
    b. Kan de gebruiker door de knop "*snooze*" de radio laten snoozen. Deze stopt met spelen en de wekker zal opnieuw aflopen na enkele minuten.
    c. De gebruiker kan ervoor kiezen de wekker uit te schakelen.


setAlarmTime / changeAlarmTime
nextRadio / selectNextRadio
turn_RadioOn
active
playing
volumeUp / increaseVolume
turn_RadioOf
volumeDown / decreaseVolume
turn_on
turn_AlarmOn
turn_AlarmOf
turn_AlarmOf
ringAlarm
alarmActive
alarmRinging
snooze / incrementAlarmTime
turn_off