***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.*

1. Introduction […./2]

Waarom is programmeren slechts een klein deel in de kost voor een "echt" software project?

De "echte" activiteiten in een software project zijn:

Requirements Collection, Analysis, Design, Implementation, Testing, Maintenance, Quality Assurance

De enige stappen waar er programmeren aan te pas komt is: Implementation, Testing en bij Maintenance.

Er komt dus veel meer bij kijken dan enkel programmeren.

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

Wat kan je doen om een vertraging op het kritische pad terug in te halen?

- ADDING MORE PROGRAMMERS = NOT GOOD (-1)
- Adding senior staff for well specified tasks (outside critical path to avoid communication overhead)
- Prioritize requirements and deliver incrementally (+1)
  - Deliver most important functionality on time
  - testing remains a priority (even if customer disagrees)
- EXTEND DEADLINE (+1)

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.

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].

5. Testing [..../2]

a) Wat is Testen?

programma uitvoeren met de boeling defects te vinden

b) Wat is een Test Techniek

technieken met een hoge kans om nieuwe fouten te vinden

c) Wat is een Test Strategie

plan dat zegt wanneer je welke techniek moet toepassen

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

Wat is het *Liskov substitutie principe*?

Replacing an instance of a superclass by each of its subclasses is allowed. .....

a class must be replaceable by any of its subclasses

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

It defines the rules for inheritance and therefore subcontracts. .....

precondition myst be weaker or equal and postcondition stronger

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 syntax 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,....

8. Software Architecture […./2]

Geef drie afwegingen bij het gebruik van het Adapter pattern?

1) How much adapting is required?

- For one class
- For the whole hierarcy

2) How will the separately developed classes evolve?

3) Does the merging work in one direction or in both directions?

4) How much overhead in performance and maintenance can you afford?

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

Benoem en definieer de vijf niveaus van het Capability Maturity Model.

Level 5: Optimizing

Improvement is fed back into QA process

Level 4: Qualitatively Managed

QA Process + quantitative data collection

Level 3:Defined

QA Process defined and institutionalized

Level 2:Managed (Repeatable)

Formal QA Procedures in place (reactive)

Level 1:Initial (Ad Hoc)

No effective QA procedures, quality is luck

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

Waarom zijn koppeling (*coupling*) en cohesie (*cohesion*) metrieken belangrijk?

.....

.....

.....

Waarom worden ze dan zelden gebruikt?

.....

.....

.....

11. Refactoring […./2]

Geef vier symptomen voor code die kan "genezen" worden met het toepassen van refactoring. Welke refactoring zou je hier dan voor toepassen?

1) Duplicated code —> Extract method, Pull Up Method (extract code to new method)

2) Nested conditionals —> Extract method (add extract code in method using inheritance and polymorphism)

3) Large classes/methods —> Extract method, Extract Class, Pull Up Method, Move Method, ….(split/add class/method)

4) Abusive inheritance —> add (sub)class to hierarchy

Andere mogelijk!!!

12. Conclusion [..../2]

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

Waarom is "program verification" 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:

Waarom was het ontwerp van de user interface gebrekkig?

Enkel toetsenbord / geen muis.

Too many menu items (+ no natural ordering of menu items)

There were too many colors in too small a space.

Error messages could appear in almost any color and could be accompanied by almost any kind of musical effect. Error messages could appear almost anywhere at the screen.

No shortcuts of any kind in the entire interface design.

No informative feedback / design dialogues to yield closure (. For example, there was a fairly complicated dialogue necessary to remove a widget from the acid bath. However, upon completion of this dialogue, the robot operator was led into a new, unrelated dialogue, without being informed that the widget removal dialogue had been completed.).

…. see killer robot article for more!!!

.....

13. Oefening - Planning [..../5]

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) en leg hieronder kort uit hoe je die berekent.

For each task n: compute earliest start-date

= Latest of all incoming paths

ESD (n) := latest of (ESD (preceding) + estimated time (preceding))

b) Vul voor elke node in de *Pert-chart* de *latest end date* in (uitgedrukt in weeknummers) en leg hieronder kort uit hoe je die berekent.

For each task n: compute latest end-date

= Earliest of all outgoing paths

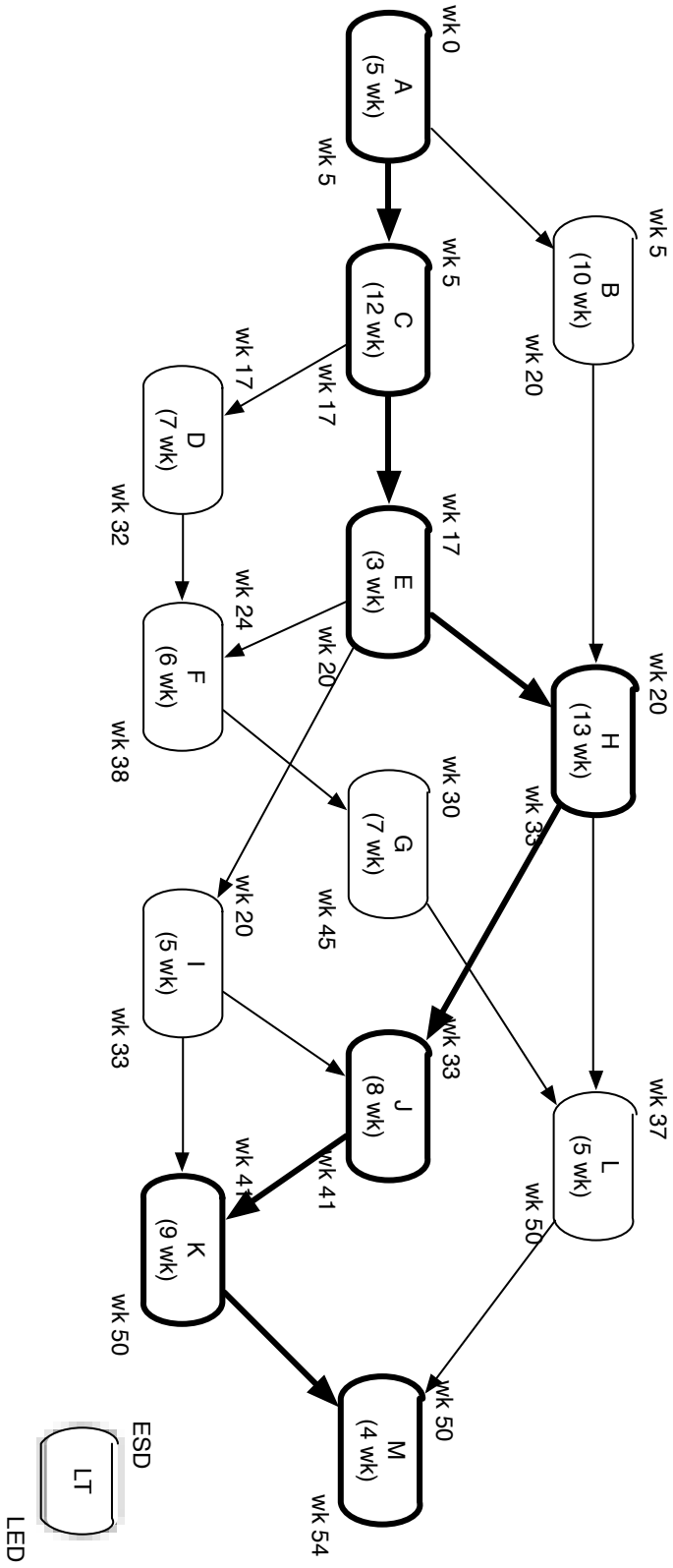
LED (n):= earliest of (LED (subsequent) - estimated time (subsequent))

c) Duid op de figuur de *critical paths* aan.

Gegeven het schema op pagina 8, bereken:

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

e) de '*worst case delay*' van het project (ten opzichte van de '*estimated time*'): .....


wk 0
A
(5 wk)
wk 5
wk 5
B
(10 wk)
wk 5
C
(12 wk)
wk 20
wk 17
wk 17
D
(7 wk)
wk 32
wk 17
E
(3 wk)
wk 24
wk 20
F
(6 wk)
wk 38
wk 20
H
(13 wk)
wk 30
G
(7 wk)
wk 45
wk 33
wk 20
I
(5 wk)
wk 33
wk 33
J
(8 wk)
wk 41
wk 41
K
(9 wk)
wk 50
wk 37
L
(5 wk)
wk 50
wk 50
M
(4 wk)
wk 54
ESD
LT
LED

Naam: .....

	OT	LT	PT	ET	S_nominator	S_denominator	S	Path	SP	Worst Case Delay
A								A		
B								A - B		
C								A - C		
D								A - C - D		
E								A - C - E		
F								A - C - D - F		
								A - C - E - F		
G								A - C - D - F - G		
								A - C - E - F - G		
H								A - B - H		
								A - C - E - H		
I								A - C - E - I		
J								A - B - H - J		
								A - C - E - H - J		
								A - C - E - I - J		
K								A - C - E - I - K		
								A - B - H - J - K		
								A - C - E - H - J - K		
								A - C - E - I - J - K		
L								A - B - H - L		
								A - C - E - H - L		
								A - C - D - F - G - L		
								A - C - E - F - G - L		
M								A - C - E - I - K - M		
								A - B - H - J - K - M		
								A - C - E - H - J - K - M		
								A - C - E - I - J - K - M		
								A - B - H - L - M		
								A - C - E - H - L - M		
								A - C - D - F - G - L - M		
								A - C - E - F - G - L - M		

Risky path
Worst case delay

weeks

14. Oefening -Testing[..../5]

Beschouw de volgende functie om de maximum waarde uit een Array van integers te bepalen:

```
int maximum(int numbers[], int array_size)
{
  int i;
  int max = numbers[0];

  for (i = 0; i < array_size-1; i++)
  {
    if (numbers[i] > max)
      max = numbers[i];
  }
  return max;
}
```

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

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

CC = 3

#e - #n + 2 =

aantal binaire condities + 1 = 3

aantal grafregios = 4

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

.....

.....

d) Geef aan welke *input* vereist is voor elk van de *onafhankelijke paden* die je in c) hebt opgesomd. (Gebruik de nummering voor verwijzingen naar elk onafhankelijk pad.)

.....

.....

e) Hoe verhoudt het aantal paden in deze verzameling zich tot de *cyclomatische complexiteit*? Waarom?

aantal paden dat nodig is om alle knopen te doorlopen is maximaal de CC.

Anderzijds kan je door het kiezen van de input data sommige paden combineren.

f) Wordt hiermee deze functie goed getest? Hoe zou je het eventueel nog beter kunnen testen?

construction is a heuristic: does not necessarily result in set of independent paths

it is possible to get the same coverage with less paths

it is sometimes not feasible to exercise all required paths

it does not necessarily cover all entry-exit paths

--> Uitbreiden met Condition Testing en of Loop Testing

15. Oefening - Formele Specificaties […./6]

In deze oefening ga je het gedrag van een automatisch autoraam (de power window) modelleren aan de hand van een ***StateChart***. De power window wordt bediend met 1 knop die een “up” of “down” signaal kan geven.

Teken je state chart op de volgende pagina.

We kunnen het gedrag van de power window als volgt beschrijven:

- Indien de knop kort een “up” signaal geeft, zal het raam kort omhoog bewegen.
- Indien de knop langer dan 3 seconden een “up” signaal geeft zal het raam helemaal sluiten.
- Indien de knop kort een “down” signaal geeft, zal het raam kort omlaag bewegen.
- Indien de knop langer dan 3 seconden een “down” signaal geeft zal het raam helemaal openen.
- Indien de autosleutels van het contact zijn en de auto wordt afgesloten, zal automatisch het raam volledig sluiten.
- Indien er bij het omhoog bewegen van het raam een obstakel gedetecteerd wordt, houdt hij onmiddelijk op met bewegen en zal hij gedurende 2 seconden terug omlaag bewegen.