***Belangrijk**: Schrijf je antwoorden kort en bondig in de daartoe voorziene velden. Indien nodig gebruik je de achterkant van het blad (met duidelijke vermelding van de vraag). Het gebruik van een eenvoudig rekenmachine is toegestaan. Andere hulpmiddelen (smartphone, laptop, cursusnota's, ...) zijn niet toegestaan tijdens het examen.*
*Elke theorie-vraag staat op 2 punten (totaal op 24). De oefeningen staan in totaal op 6 punten. Het geheel staat op 30 punten.*

1. Introduction [... / 2]

   Geef de 6 principes van *Extreme Programming*.

   - *Fine scale feedback*
   - *Continuous 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 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 drie criteria voor goede use cases wat betreft het beschrijven ven requirements en bespreek waarom deze criteria belangrijk zijn.

   *1) begrijpbaar: actoren geven een end-user perspectief weer*
   *2) nauwkeurig: scenario's zijn voldoende gedetailleerd om als test-input te dienen*
   *3) open: het perspectief van de actor benadrukt het wat (en veel minder het hoe)*

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*

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 possibilities were done by the previous technique(s) => complementary*

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

Als een subclasse een operatie override, wat mag het dan doen met de pre- en postcondities? En wat met de klasse invariant?

*Precondition: weaker or equal*

*Postcondition: stronger or equal*

*Invariant: equal*

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]

Wat is *coupling*?

*Een maat voor hoe sterk een component gekoppeld is met een andere component via een connector*

Wat is *cohesion*?

*de mate waarin de onderdelen van een component samenhangen*

Hoe kunnen we ze gebruiken als criterium voor een goed design?

*minimise coupling and maximise cohesion*

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]

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

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

11. Refactoring [... / 2]

Welke vier activiteiten zouden door tools moeten ondersteund worden als men aan refactoring doet?

*1) Refactoring - Source-to-source program transformation behaviour preserving*
*⇒ improve the program structure*
*2) Regression Testing - Repeating past tests*
*⇒ improvements do not break anything*
*3) Programming Environment - Fast edit-compile-run cycles Support small-scale reverse engineering activities*
*⇒ convenient for "local" ameliorations*
*4) Configuration & Version Management - keep track of versions that represent project milestones*
*⇒ go back to previous version*

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:

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.*

13. Oefeningen – Refactoring [... / 6]

Jouw ontwikkelingsteam wordt gevraagd een recommendation system te schrijven voor een bestaande mediaplayer. Deze recommendations maken gebruik van de titel, artiest, … (eventueel nog nieuwe informatie die toegevoegd kan worden op een later tijdstip) van de nummers in verschillende playlists die een gebruiker kan aanmaken om nieuwe voorstellen te doen. Zoals je kan zien in de code op pagina 7 is er nog wat voorbereidend werk nodig aan de code die een nummer en een afspeellijst implementeert vooraleer je team kan starten aan de gevraagde nieuwe functionaliteit.

Ga na welke verschillende refactoring-technieken je zou moeten toepassen op de code op pagina 13 om deze te verbeteren. Geef voor elke refactoring de volgende informatie:

- De lijnnummer(s)
- De naam van de techniek die je wilt toepassen
- De reden waarom je deze techniek wilt toepassen
- Beschrijf kort hoe je deze refactoring in de praktijk wilt uitvoeren

```
1   private class Audiobestand {
2
3           private String m_naam;
4           private String m_uitvoerder;
5           private Tijd m_tijdsduur;
6
7           Audiobestand(String naam, String uitvoerder, Tijd tijd) {
8                   m_naam = naam;
9                   m_uitvoerder = uitvoerder;
10                  m_tijdsduur = tijd;
11          }
12
13          public void speelAf() {
14                  System.out.println("Start afspelen: " + m_naam + " - " + m_uitvoerder);
15          }
16
17          public void stop() {
18                  System.out.println("Stop afspelen: " + m_naam + " - " + m_uitvoerder);
19          }
20
21  }
22
23  private class Tijd {
24          public long m_seconds;
25
26          Tijd() {
27                  m_seconds = 0;
28          }
29
30          public void add(long sec) {
31                  m_seconds += sec;
32          }
33  }
34
35  private class Afspeellijst {
36          private List<Audiobestand> m_files;
37
38          public Afspeellijst() {
39                  m_files = new ArrayList<Audiobestand>();
40          }
41
42          public void add(Audiobestand file) {
43                  m_files.add(file);
44          }
45
46          public void printInfo() {
47                  Tijd total_time = new Tijd();
48                  for (Audiobestand file : m_files) {
49                          System.out.println("> " + file.m_naam + " - " + file.m_uitvoerder);
50                          total_time.add(file.m_tijdsduur.m_seconds);
51                  }
52                  long secs = total_time.m_seconds;
53                  long hours = secs / 3600;
54                  secs -= hours * 3600;
55                  long mins = secs / 60;
56                  secs -= mins * 60;
57                  System.out.println("Total duration: " + hours + ":" + mins + ":" + secs);
58          }
59
60  }
```