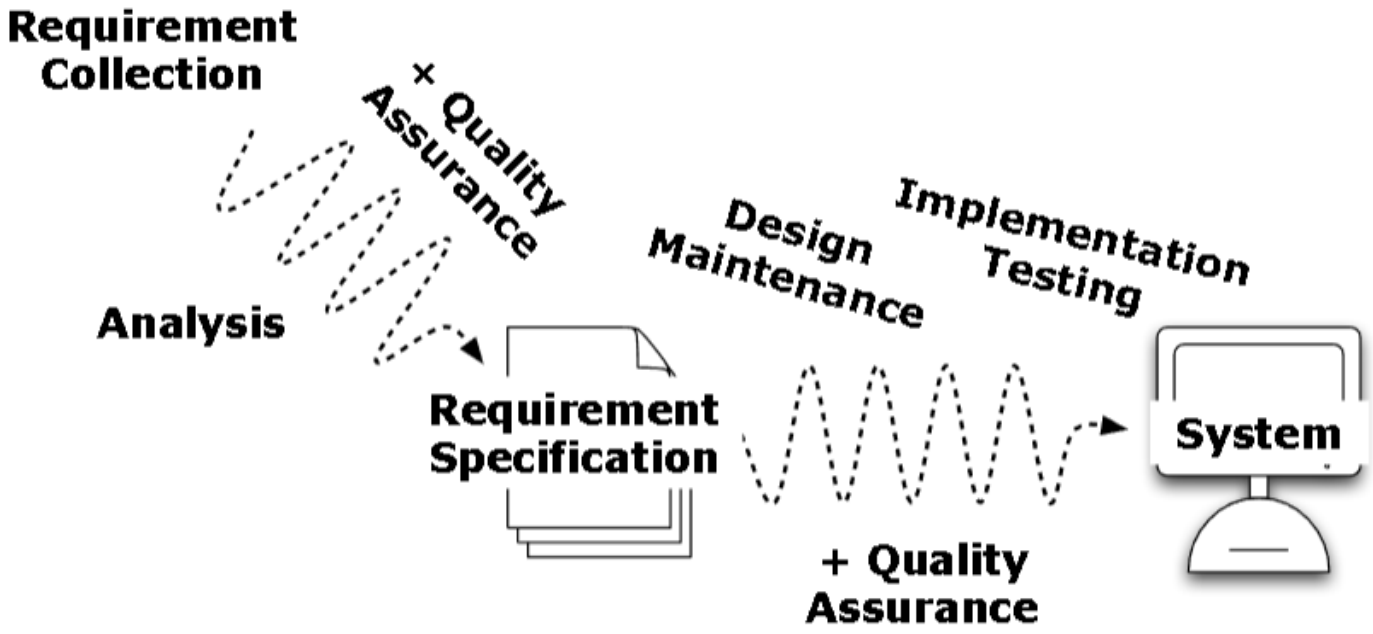
SE Questions for every chapter

> Questions of Software Engineering course, also answered in Dutch

Intro



Summary i

- How does Software Engineering differ from programming?
  - Programmeren is de taak dat we uitvoeren om een product te kunnen verkrijgen. Terwijl Software Engineering het gehele process is om tot aan het product te komen.
    - Bijvoorbeeld. het consulteren van wat het product moet omvatten (Requirements) met de klant, de verschillende risico's dat kunnen voorkomen bij het maken van een bepaalde feature, de kwaliteit van het product bij een tussentijdse en/of eindevaluatie, enz.
  - "*Software Engineering is a state-of-the-art profession dedicated to designing, implementing and modifying software on time and within budget so that it is of higher quality, more affordable, maintainable and faster to build.*"
  - "*Multi-person construction of multi-version programs* "
  - "*Software engineering is different from other engineering disciplines*"
- Why is programming only a small part of the cost of a “real” software project ?
  - Er is veel tijd nodig om te weten wat de klant wilt verkrijgen van het product. Dit moet duidelijk zijn voor zowel klant als voor ons.
  - We moeten nagaan of sommige features risico's omvatten dat we moeten voor nakijken of oppassen. (Na elke tussentijdse bespreking)
  - Daarnaast moet er tijdens elke tussentijdse bespreking ook een validatie/verificatie gedaan worden, m.a.w. we gaan na of we het juiste maken en kijken na of we dit op een juiste manier

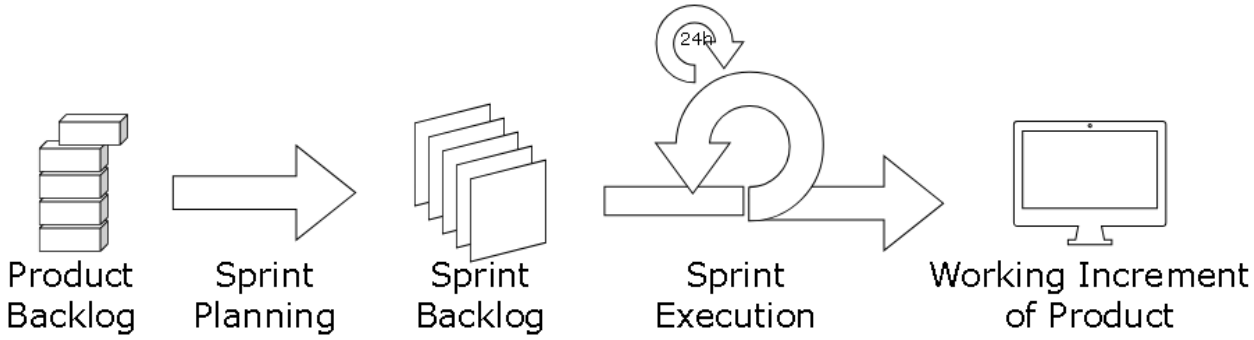
doen. Indien dit niet zo is, moeten we dit weer opnieuw doen.
    - Er is daarnaast ook bij elke bespreking een check nodig of het nog nuttig is om verder aan het project te werken, dit heeft ook een grondige bespreking nodig en kan ook wat tijd kosten indien dit niet duidelijk is.
    - We hebben dan ook nog een plan nodig om aan te geven hoe we te werk gaan (development plan).
    - Naast het documenteren, hebben we ook testing, dat apart gezien wordt van de "implementatie".
    - En het programmeren komt dan na al deze stappen dat in slechte gevallen meerdere iteraties moet overlopen worden.
        - Het programmeren kan ook lang duren maar dit is vergeleken het "plan" opstellen een fractie van tijd.
- Give a definition for “traceability”.
    - Traceability in context van software engineering kan in verschillende vlakken besproken:
        - Kunnen we nagaan wat de impact zou zijn van een bepaalde aanpassing op verschillende components van het product?
        - Hoe kunnen we dit doen?
            - Relaties bijhouden:
                - Van component -> requirements dat de aanwezigheid veroorzaakte.
                - Van requirements volgens de impact van een component die is aangepast.
- What is the difference between analysis and design?
    - Analysis: is de "What?", dus het modelleren en specificeren van de requirements .
    - Design:
        - Is de "How?", modelleren en specificeren van de oplossing.
        - System design (architecture) + detailed design (object design, formal spec).
    - Dus het verschil ligt dat de analyse specificeert wat we gaan maken en dat design zegt ons vertelt hoe we dit gaan doen.
- Explain verification and validation in simple terms.
    - Verification: maken we het juist?
    - Validation: maken we het juiste?
- Why is the “waterfall” model unrealistic? Why is it still used?
    - Why is the “waterfall” model unrealistic?
        - Complete: realistisch kan een klant niet alle specificaties direct specificeren.
        - Idealistic: realistisch gaan we door meerdere iteraties (tools en organisatie belemmeren dit)
        - Time: Een werkend product is alleen zichtbaar op het einde van de process, wat problemen kan zorgen indien er iets fout is gelopen
        - Change: Onhandig en duur indien er aanpassingen moeten gemaakt worden aan de requirements. Bijvoorbeeld process loopt fout op het einde of een requirement is fout gedefinieerd en dit is in 1 van de latere stages van het process dan kan dit zorgen voor veel problemen.
    - Why is it still used?

    - Het watervalmodel is populair bij het hogere management, omdat het zichtbaar is: het is eenvoudig om de voortgang van projecten te controleren.
        - Het is een linear traject dus alles bouwt op elkaar dus het is makkelijk te weten in welke stage we zitten van het project --> visibility
        - Omdat het opbouwend is, is er veel controle over het project, namelijk indien er geen voortgang is in een bepaalde stage kunnen we niet verder naar het volgende stage.
- Can you explain the difference between iterative development and incremental development?
    - Iterative development:
        - We itereren door een volgorde van stappen opnieuw en opnieuw om uiteindelijk een eindresultaat te bekomen. We gaan gecontroleerd stukjes van het product opbouwen om verbetering te verkrijgen.
          + "*we get things wrong before we get them right.*"

    - Incremental development:
        - Hier splitsen we ons product in verschillende stukjes en gaan we stapgewijs te werk. We streven naar product dat op elk moment bruikbaar is. We streven naar vroege en kleine resultaten.
            - " *always having a running version.*"
- How do you decide to stop in the spiral model?
    - Extra uitleg in verband met de 4 fases van spiral model:
        - Determine objectives: waar we de requirements opbouwen en verschillende oplossingen bekijken indien plan A faalt.
        - Daarna hebben we risk analyse, waarbij we de risico bekijken op de componenten op basis van verschillende aspecten. Dit kan zowel risico's op basis van de requirements zijn of risico's op basis van de budget.
        - Daarna maken we gebruik van de plan die we gemaakt hebben in de vorige twee stages om iets te ontwerpen en te testen.
        - Als laatste doen we een evaluatie en kijken we of het een succes was of een faal. En plannen we de volgende iteratie.
    - Het stoppen wordt bepaald door een risk analyse, we checken of het nog waard is of nog verder te werken aan het project.

- How do you identify risk? How do you asses a risk? Which risks require action?
  - How do you identify risk?
    - Op basis van een "risk item checklist". Dit checklist is uniek voor bepaalde gevallen, bijvoorbeeld andere programmeertalen.
  - How do you asses a risk?
    - Voor elke risk factor bepalen we een likelihood en de impact van de risk
      - 3 point scale
        - low - medium - high
      - 5 point scale
        - [impact] insignificant - minor - moderate - major - catastrophic
        - [likelihood] almost certain - likely - possible - unlikely - rare
    - Met deze projectie gaan we dan bepalen hoe belangrijk een risk is, dus hoe hoog zijn impact is met zijn likelihood.
  - Which risks require action?
    - important risk factors: dus risks met hoge impact en medium tot hoge likelihood
- What is Failure Mode and Effects Analysis (FMEA)?
  - Een stap voor stap aanpak om alle mogelijke fouten te detecteren in een bepaalt design, een fabricageproces of assemblageproces, of een product of een service.
    - Failure mode: alle mogelijke fouten die voor kunnen komen die mogelijk de klant kan storen. Deze fouten kunnen toekomstige fouten zijn of momentele fouten.
    - Effect Analysis: Is de analyse van de consequenties van deze fouten.
- List the 6 principles of extreme programming.
  - Fine scale feedback
    - Dit is gerelateerd tot de feedback, regelmatige feedback en gedetailleerde feedback wordt onder verwisseld tussen elkaar.
  - Continuous process
    - Heeft te maken met de constante progress. Dit zorgt voor "Continuous Integration" waarbij elk probleem dat te gemoed komt, zo spoedig nodig opgelost wordt.
  - Shared understanding
    - Dit is verband dat de groep een zelfde visie hebben over het product en werken op een gelijkaardige manier zodat er geen verwarring komt in notaties.
  - Programmer welfare
    - Dit heeft te maken met de snelheid dat een programmeur kan werken, we willen namelijk dat hij geen burnout op loopt en dat de code nog steeds kwaliteit vol blijft. Dit heeft ook te maken met de netheid van de code, waardoor het verstaan van de code beter wordt.
  - Coding
    - Dit heeft te maken met modelling van onze product dat dan later omgezet wordt naar code. Daarnaast zoeken we hier ook naar alternatieve oplossingen.
  - Testing
    - Dit heeft te maken met het testen van de code:
      - Voor alle code bestaat er een Unit test (automated test om te checken of het juist is).
      - Alle code moeten deze Unit test slagen.

        - Wanneer bugs gevonden worden, worden er eerst testen gemaakt en daarna worden deze aangehaald.
        - Acceptance test (klant checked of het product nog voldoet aan zijn wensen) worden regelmatig uitgevoerd en worden publiekelijk gepubliceerd.
- What is a “sprint” in the SCRUM process?
    - 2 - 4 week periode
    - Team ontwerpt een werkende incremental product
    - Features die gekozen worden, worden gekozen op basis van de backlog van de sprint.



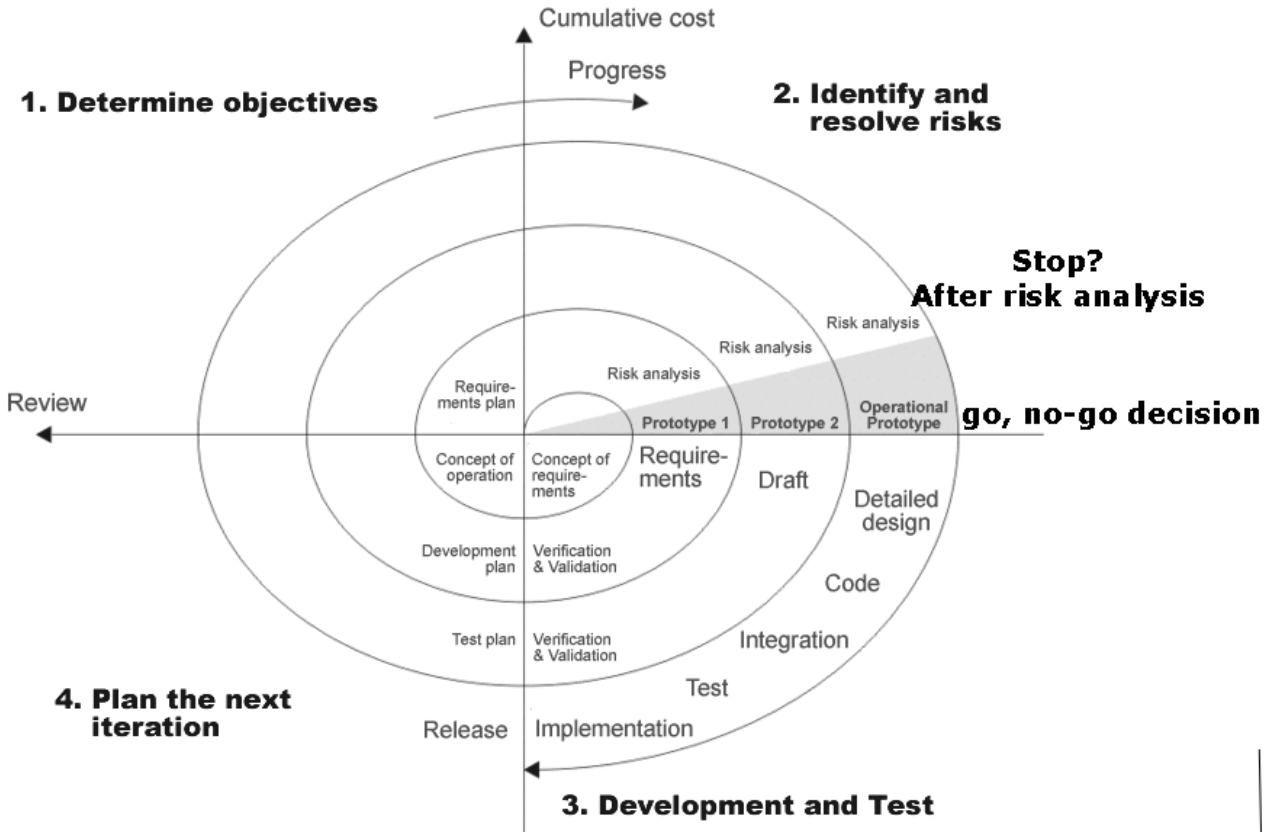
- Give the three principal roles in a scrum team. Explain their main responsibilities.
    - Product owner
        - Prioritizing backlog
            - sets weights on features in the backlog based on how important it is or how crucial
    - Scrum master
        - A facilitator
            - plans, guides and manages a group event to meet its goals
    - Development team
        - Responsible for increment to be added to the product
            - 5 - 9 individuals
            - self organizing
- Draw a UML class diagram modelling marriages in cultures with monogamy (1 wife marries 1 husband), polygamy (persons can be married with more than one other person), polyandry (1 woman can be married to more than one man) and polygyny (1 man can be married to more than one woman).
    - To do

Summary ii

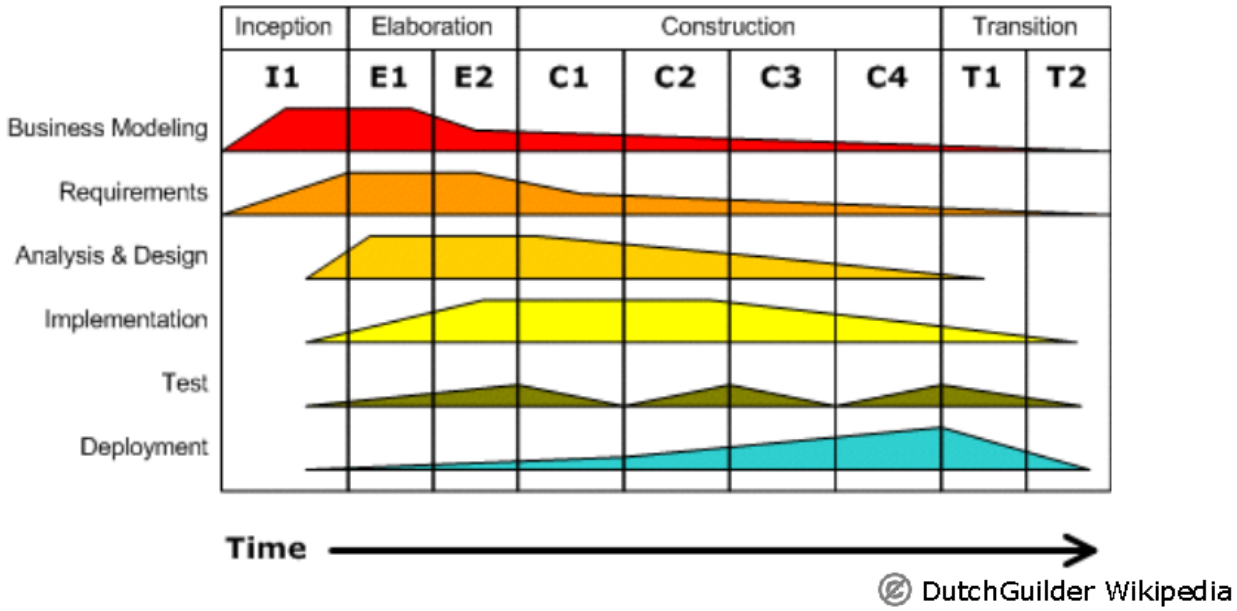
- Draw a UML diagram that represents an object “o” which creates an account (balance initially zero), deposits 100$ and then checks whether the balance is correct.
    - To do
- What is your preferred definition of Software Engineering? Why?
    - We hebben 3 definities gezien van "Software Engineering":
        - "state of the art of developing quality software on time and within budget"
        - "multi-person construction of multi-version programs"
        - "software engineering is different from other engineering disciplines"

    - Mijn preferred definitie is de eerste, dit is omdat we spreken over de kwaliteit van het product. Persoonlijk vind ik dat het belangrijkste dat we een kwaliteit vol product verkrijgen, binnen een budget en tijdspannen.
    - De tweede definitie gaat dan meer over team verband, dat ook belangrijk is maar niet de essentie toont van software engineering.
    - De derde definitie zegt enkel dat het anders is dan andere engineering praktijken, vertelt niks over wat het is en klinkt ook negatief tegen over de andere praktijken.
- Why do we choose “Correctness” & “Traceability” as evaluation criteria? Can you imagine some others?
    - *Correctness: Validation / Verification.* Dit zorgt voor de kwaliteit van het product
        - Zijn we het juiste aan het maken?
        - Maken we het juist?
    - *Traceability: Deduce what things will be affected by change.* Dit zorgt ervoor dat we kwaliteit behouden van het product indien er aanpassingen zijn.
    - *They ensure quality.*
    - *Others: reusability, readability, cost, completeness, ...*
- Why is “Maintenance” a strange word for what is done during the activity?
    - Maintenance: Het definieert de aanpassingen dat gebeuren na dat de product is gepubliceerd.
    - Bijvoorbeeld: reparaties bij defecten, updates na de deployment, ...
- Why is risk analysis necessary during incremental development?
    - Dit zorgt ervoor dat we weten of we wel nog nuttig bezig zijn.
        - Go, No-go decision
    - Het zorgt er ook voor dat we eventuele problemen op tijd kunnen ontwijken of oplossen.
- How can you validate that an analysis model captures users’ real needs?
    - Een kleine prototype opbouwen zodat de klant kan zien of de gestelde gevalideerd wordt.
        - Exploratory prototype
            - UI prototype: to validate user requirements
- When does analysis stop and design start?
    - Analyse stopt nooit, dit is omdat er altijd nieuwe requirements bij kunnen komen.
    - Design start wanneer we voldoende aantal requirements hebben om een product te kunnen voorstellen.
- When can implementation start?
    - Implementatie kan technisch gezien direct gestart worden, maar dit is natuurlijk na dat we weten waar we kunnen starten.
- Can you compare the Unified Process and the Spiral Model?

- Spiral model: de spiral model van Boehm is een model dat meer gebaseerd is op risk analyse.

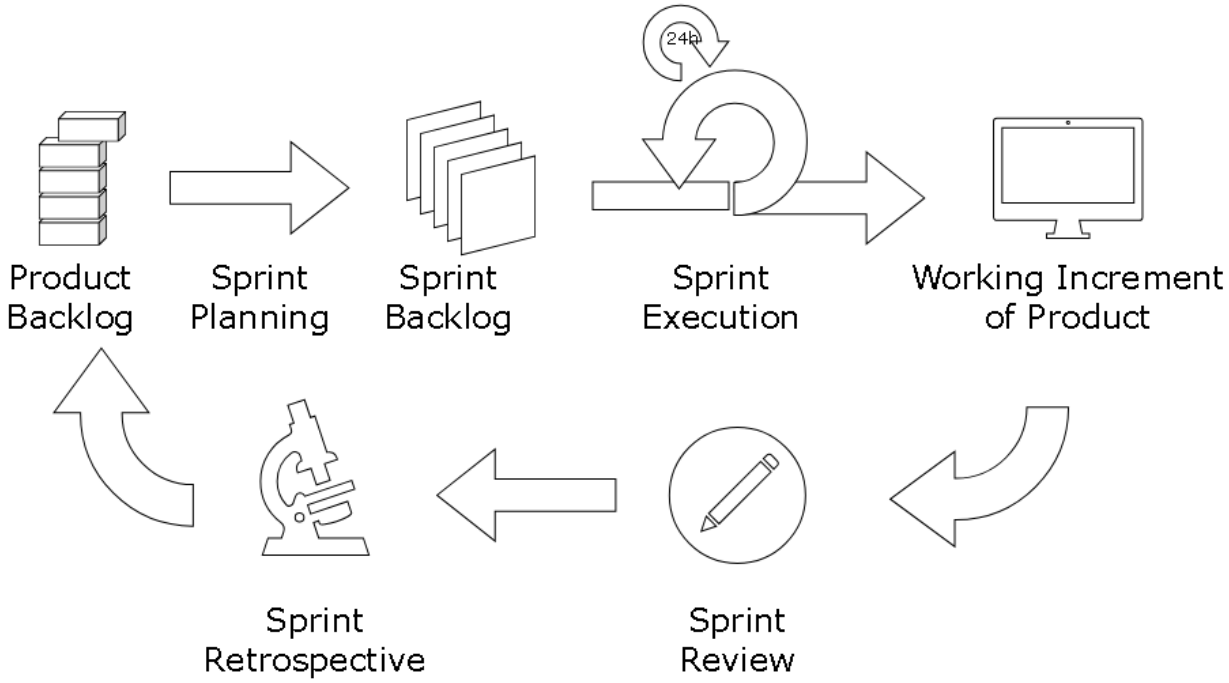


- Unified Process model: Is een model dat meer werkt op basis van tijd, het maakt gebruik van aantal iteraties en is meer incrementeel model van werken.



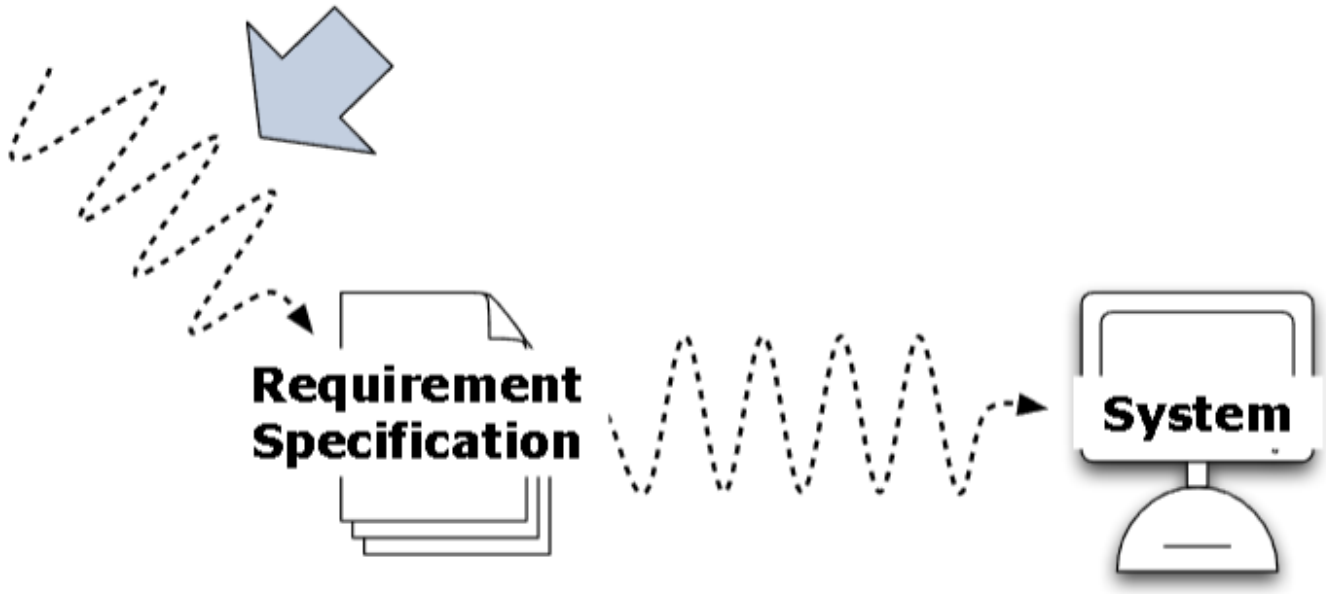
- Can you explain the values behind the Agile Manifesto?
  - Agile Manifesto streeft meer naar kwaliteit van het product in verband met de klant.
    - **Individuals and interactions** over processes and tools
    - **Working software** over comprehensive documentation
    - **Customer collaboration** over contract negotiation
    - **Responding to change** over following a plan
- Can you identify some synergies between the techniques used during extreme programming?
  - Fine scale feedback - Coding
    - Waarbij we werken met pair programming, dit zorgt ervoor dat we coderen met feedback van een andere.

        - Daarnaast coding is niet enkel de code gedeelte maar ook weten wat er gecodeerd moet worden en daarom is de klant ook een deel van de team (Whole team).
    - Coding - Programmer Welfare
        - No overtime. We willen natuurlijk niet dat programmeren resulteren in code quality loss, dus moeten we nagaan dat we geen overtime en burnout oplopen.
    - Coding - Testing
        - We maken eerst de unit test voor dat we beginnen aan de werkelijke code. Dit zorgt ervoor dat we kunnen nagaan of we de juiste resultaten verkrijgen.
    - Shared understanding - Coding
        - Simple code maakt het makkelijker voor iedereen om te weten wat er gemaakt wordt.
        - Daarnaast gebruik maken van system metaphors, maakt coderen veel overzichtelijker.
- Can you explain how the different steps in the scrum process create a positive feedback loop?
    - Product backlog --> Sprint planning --> Sprint backlog --> Sprint execution (24h) --> Working increment of product --> Sprint review --> Sprint retrospective --> Product backlog



- How does scrum reduce risk?
    - Omdat we een sprint meeting om de 2 - 4 weken doen, kunnen we elke risico's identificeren indien er zijn. Dit zorgt ervoor dat we de "Severe risks" op tijd kunnen benaderen en van boven aan in de sprint backlog kunnen plaatsen.
    - Dit geeft ons meer een visie wat momenteel belangrijk is om aan te werken, waardoor grote problemen sneller benaderd worden.
- Is it possible to apply Agile Principles with the Unified Process?
    - Technisch gezien ja, we kunnen de principes dat we willen benaderen in de iteraties toepassen.
- Did the UML succeed in becoming the Universal Modelling Language? Motivate your answer.
    - Ja
        - Er is een uniforme notatie
        - Complete lifecycle process
        - adaptable: je kan de notatie uitbreiden, je kan zelf je process kiezen

Requirements



Summary i

- Why should the requirements specification be understandable, precise and open?
  - Understandable: Zodat we weten wat in en uit de scope zit van ons product.
  - Precise: Zodat we weten wat binnen en buiten de systeem hoort.
  - Open: Zodat de developers genoeg vrijheid hebben om een optimaal oplossing te vinden.
- What's the relationship between a use case and a scenario?
  - Een use-case beschrijft opmerkelijke requirements van het systeem.
  - Een scenario: Het is een instantie van een use-case, vertelt ons een typisch voorbeeld van deze use-case indien we het uitvoeren.
- Can you give 3 criteria to evaluate a system scope description? Why do you select these 3?
  - Should be short
    - lange beschrijvingen zijn niet overtuigend.
  - Should be written down
    - wordt later naar gerefereerd wanneer we spreken over de prioriteiten van de use-cases.
  - Should have end-user commitment
    - end-user wordt betrokken in het beschrijven van het product.
- Why should there be at least one actor who benefits from a use case?
  - De correspondeerde stakeholder (actor) zal dan eisen om deze use-case in de requirements houden.
- Can you supply 3 questions that may help you identifying actors? And use cases?
  - Actors
    - Who uses the system?
    - Who installs the system?
    - Who starts up/shuts down the system?
    - Who maintains the system?

        - What other systems use this system?
        - Who provides information to this system?
        - Does anything happen automatically at a preset time?
    - Use cases
        - What functions will the actor want from the system?
        - What actors will create, read, update, or delete information stored inside the system?
        - Does the system need to notify actors about changes in its internal state?
        - Who gets information from this system?
        - Are there any external events the system must know about?
        - What actor informs the system about those events?
- What's the difference between a primary scenario and a secondary scenario?
    - Primary scenario = "succes" scenario
        - We veronderstellen dat alles juist gaat.
    - Secondary scenario = "alternative" scenario
        - We veronderstellen dat er een andere pad is dat een bepaalde use case kan doorlopen. Meestal special cases. Bijvoorbeeld indien we een error hebben.
- What's the direction of the <<extends>> and <<includes>> dependencies?
    - <<extends>>
        - Dit zal wijzen naar onze scenario. Scenario <-- use-case.
    - <<includes>>
        - Dit zal wijzen weg van onze scenario. Scenario --> use-case.
- What is the purpose of technical stories in scrum?
    - Technical stories drukken onze non-functional requirements uit in story form.
- List and explain briefly the INVEST criteria for user stories.
    - Independent
        - Stories moeten onafhankelijk zijn van elkaar, er zijn geen onderlingen verbanden.
    - Negotiable
        - Te veel detail maakt het overleggen moeilijker met de klant.
    - Value
        - Elke story moet waardevol zijn voor de klant.
    - Estimate
        - Stories moeten klein genoeg zijn om voorgesteld te kunnen worden.
    - Small enough
        - Stories moeten klein genoeg zijn om in 1 iteratie voltooid te kunnen worden.
    - Testable
        - Acceptance criteria moet beschikbaar zijn
            - "*Definition of done*"
- Explain briefly the three levels of detail for Product Backlog Items (Epic, Features, Stories).
    - Epic
        - Een maanden
        - Groter dan een release
    - Feature

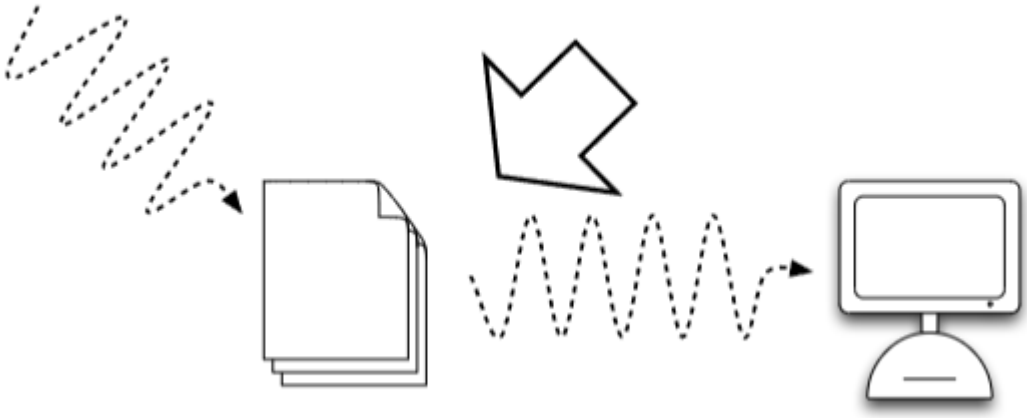
        - Een weken
        - Groter dan een sprint
    - Sprintable features
        - Enkele dagen
        - Sprint ready
- What is a minimum viable product?
    - Is een minimum bruikbaar versie van het product met het juiste aantal features, zodat users het kunnen gebruiken om feedback te geven voor future development.
- Define a misuse case.
    - Een use-case waarbij de actor hostile is tegen de systeem.
        - Bijvoorbeeld in de context van een auto hebben, zou een andere actor als use case hebben "auto stelen".
- Define a safety story.
    - Indien het satisfied is, zou het een probleem vermijden of de impact van het probleem verminderen indien dit te voorschijn komt.

Summary ii

- Why do use cases fit well in an iterative/incremental development process?
    - Dit is omdat use-cases helpen definiëren welke features nodig zijn om een vroege werkende product te hebben.
    - Geeft ons ook een mogelijkheid om aanpassingen toe te voegen na elke iteratie, indien er problemen zijn tijdens de momentele iteratie.
- Why do we distinguish between primary and secondary scenarios?
    - We willen het verschil weten tussen de algemene scenario en alternatieve scenarios, dit zorgt er voor dat we een brede overzicht hebben over de use-case.
    - We maken vooral een verschil omdat de secondary scenarios, scenarios zijn die niet veel voorkomen maar nog steeds belangrijk zijn om te weten dat die bestaan. Zodat eventuele maatregelingen kunnen nemen indien nodig.
- What would you think would be the main advantages and disadvantages of use-cases?
    - Voordelen
        - Uitzondering identificeren
            - Ze helpen ons uitzondering te identificeren dat we origineel misschien niet gezien zouden hebben.
        - Simpler
            - Use cases maken ook het simpler om de goal te begrijpen
        - Primary en secondary scenarios geven ons meer een "future sight"
            - Ze geven ons een blik op de toekomst, zodat we eventuele problemen kunnen vermijden.
    - Nadelen
        - Complex
            - Bij complexe systemen kan het beheren van use cases complex worden.
        - Beperkte focus

        - Use case zijn gefocust op functionele requirements en niet non-functional requirements.
- How would you combine use-cases to calculate the risky path in a project plan?
    - We identificeren use-cases die op volgorde van andere use-cases gemaakt moeten worden, en kijken of ze bepaalde risico's hebben. Indien ze risico's hebben, kijken we hoe groot deze impact kan zijn en bepalen op basis van deze info de risico path.
- Do use-cases work well with agile methods? Explain why or why not.
    - Er zijn vlakken waarmee het bots en waarmee het perfect samen zou kunnen werken
        - Good
            - We kunnen gebruik maken van onze use-cases om een sprint concept uit te voeren.
            - Met use case kunnen we ook vroeg risico's identificeren.
        - Bad
            - Agile maakt gebruik van weinig documentatie, use-cases in tegendeel kunnen soms redelijk gedocumenteerd zijn wat tegen de Agile method gaat.
            - Daarnaast kan het zijn dat use-case veel tijd rooft indien er aanpassingen zijn aan requirements. Wat bij Agile omarmt wordt.
- Can you explain the use of a product roadmap in scrum?
    - Het vertelt ons hoe we onze product moeten opbouwen na elke increment. Het vertelt ons elk belangrijk feature dat toegevoegd wordt bij elke grote release.
- Choose the three most important items in your “Definition of Ready” checklist. Why are these most important to you?
    - De taak is realistisch, dus het moet klein/groot genoeg zijn om voltooid te kunnen worden in 1 sprint.
    - Testbaar, de taak moet testbaar zijn. Er moet een succes criteria zijn voor de taak.
    - Onafhankelijk, dit is niet altijd mogelijk maar liefst zijn de taken onafhankelijk van elkaar.
- Can you relate scrum user stories to some of the principles in the Agile Manifesto?
    - **Individuen en interacties boven processen en tools**: User Stories benadrukken de behoeften van de gebruiker en de interactie tussen het systeem en de gebruiker.
    - **Werkende software boven uitgebreide documentatie**: User Stories zijn beknopt en richten zich op het leveren van waardevolle, werkende software.
    - **Klantensamenwerking boven contractonderhandeling**: User Stories bevorderen samenwerking met klanten door te focussen op hun behoeften en feedback.
    - **Reageren op verandering boven het volgen van een plan**: User Stories zijn flexibel en kunnen gemakkelijk worden aangepast als de behoeften van de gebruiker veranderen.
- How would you turn an FMEA analysis into a misuse case diagram?
    - Onze potential failure modes, zijn onze verschillende user stories. We kunnen dit dan visualiseren als een node in onze misuse case diagram.
- Elaborate on the relationship between an FMEA analysis and the variants of safety stories.
    - Safety stories zijn de oplossingen die we kunnen toepassen bij een failure mode.

Architecture



Summary i

- What's the role of a software architecture?
  - Een beschrijving van components en hun verbindingen tussen elkaar.
- What is a component? And what's a connector?
  - Component
    - Een ingekapseld onderdeel van een softwaresysteem met een designated interface
  - Connector
    - Een verbinding tussen componenten
- What is coupling? What is cohesion? What should a good design do with them?
  - Coupling
    - Een manier om de sterkte van verbinding aan te geven tussen connectors
      - High coupling betekent dat modules nauw met elkaar verbonden zijn en dat veranderingen in één module andere modules kunnen beïnvloeden.
  - Cohesion
    - Verwijst naar de mate waarin elementen binnen een module samenwerken om één enkel, goed gedefinieerd doel te vervullen.
      - High cohesion betekent dat elementen nauw met elkaar verbonden zijn en gericht zijn op één enkel doel.
  - Good design
    - Minimize coupling and maximize cohesion
- What is a pattern? Why is it useful for describing architecture?
  - De essentie van een oplossing voor een terugkerend probleem in een bepaalde context.
  - Hergebruik van oplossingen bij gelijkaardige patronen.
- Can you name the components in a 3-tiered architecture? And what about the connectors?
  - Application layer
  - Domain layer
  - Database layer
  - Alle verbindingen worden gedaan via de "domain layer". De "application layer" en "database layer" hebben geen communicatie met elkaar.

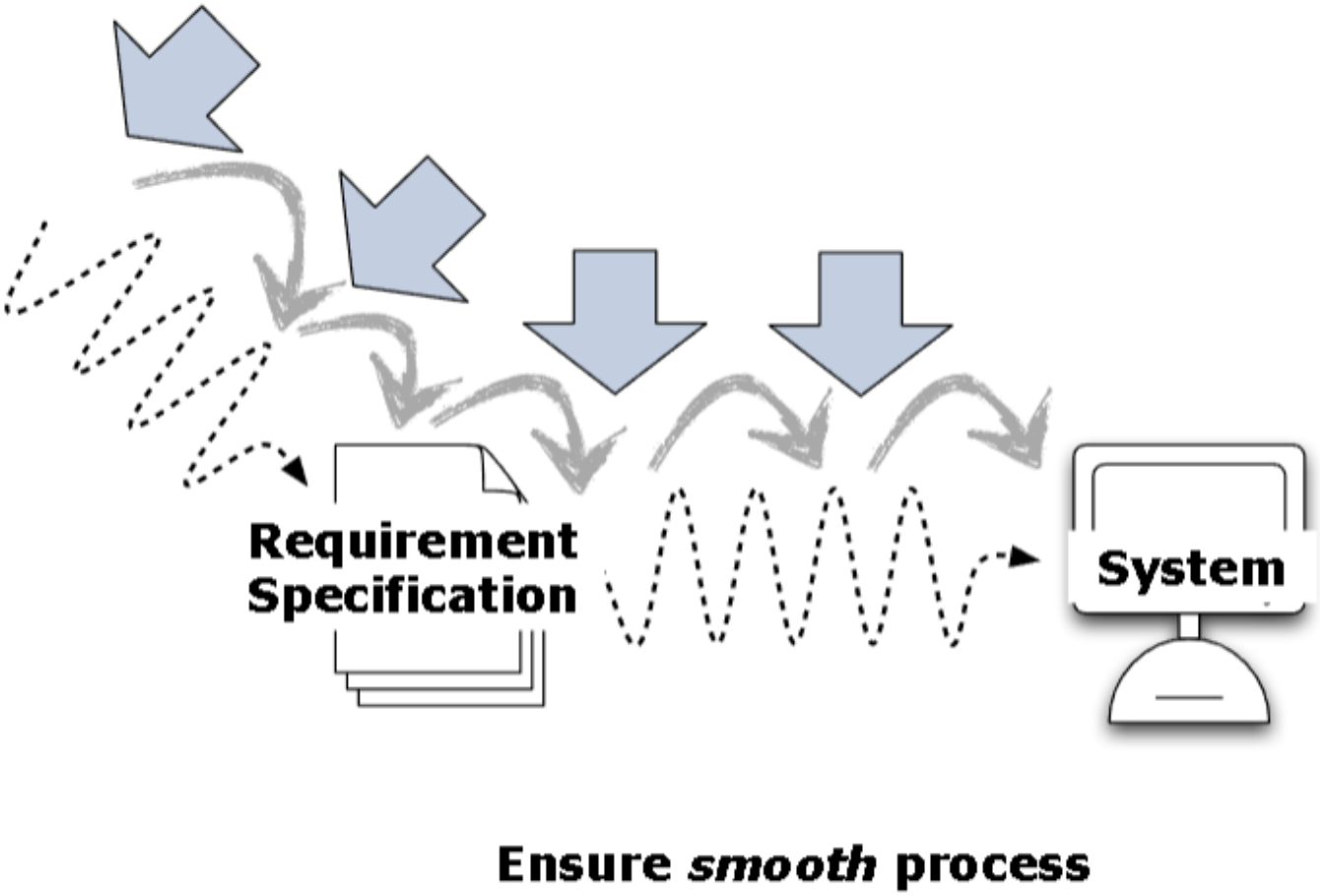
- Why is a repository better suited for a compiler than pipes and filters?
  - Bij "pipes and filters" moeten we hele tijd heel de data sturen, dit is zeer inefficient.
  - Terwijl we bij repository architecture stukjes van de data kunnen aanpassen.
  - Daarnaast kan veel filters veel computationele overhead creëren.
- What's the motivation to introduce an abstract factory?
  - Klassen hiërarchie met abstracte wortels die een familie van objecten vertegenwoordigen.
  - Abstract Factory is een patroon waarmee je families van gerelateerde objecten kunt produceren zonder hun concrete klassen te specificeren
- Can you give two reasons not to introduce an Adapter (Wrapper)?
  - Indien we aanpassingen maken aan de component dat we wrappen, moeten we ook de adapter aanpassen.
  - Daarnaast maakt het ook de systeem complexer, want we hebben een extra laag of abstractie.
- What problem does an abstract factory solve?
  - Het aanroepen van constructeurs impliceert een nauwe koppeling met betonnen bladeren in plaats van met abstracte wortel. (High coupling)
- List three trade-offs for the Adapter pattern.
  - How much adapting is required?
    - For one class
    - For the whole hierarchy
  - How will the separately developed classes evolve?
  - Does the merging work in one direction or in both directions?
  - How much overhead in performance and maintenance can you afford?
- How do you decide on two architectural alternatives in scrum?
  - We maken gebruik van spike stories en we maken daarna dan gebruik van condition of satisfactions of te checken welke architecturen geschikt zijn.
- What's the distinction between a package diagram and a deployment diagram?
  - Kortom, een **Package Diagram** is gericht op het organiseren van de softwarecomponenten, terwijl een **Deployment Diagram** zich richt op hoe deze componenten daadwerkelijk worden ingezet op de hardware gedeelte.
- Define a sensitivity point and a trade-off point from the ATAM terminology.
  - Een **sensitivity point** is een eigenschap van een of meer componenten (en/of component relaties) die van cruciaal belang is voor het bereiken van een bepaalde kwaliteit attribuut respons.
  - Een **trade-off point** omvat twee (of meer) conflicterende gevoeligheidspunten

Summary ii

- What do architects mean when they say "architecture maps function onto form"? And what would the inverse "map form into function" mean?
  - Simpel weg is het gewoon het mappen van onze requirements op de structuur van de systeem.
  - Het inverse resulteert dat we de structuur van de systeem op de requirements mappen.
- How does building architecture relate to software architecture? What's the impact on the corresponding production processes?
  - Ze hebben veel gemeenschappelijke eigenschappen:

        - Zowel bij gebouwen en software moeten veel plannen en ontwerpen voordat we kunnen beginnen "bouwen".
        - Beide zijn opgebouwd in componenten, als we kijken naar complexe infrastructuren zien we dat gebouwen kunnen opgebouwd worden in veel verschillende componenten. Gelijkaardig als software.
    - Impact:
        - Efficiëntie: Een goed ontwerp resulteert in een efficiëntere opbouw.
        - Kosten: Keuze van de architecture zal ook de kosten bepalen, bepaalde architecturen kosten meer werk dan andere.
        - Kwaliteit: Keuze van architectuur past ook de kwaliteit aan.
- Why are pipes and filters often applied in CGI-scripts?
    - Om datastromen te verwerken waarbij flexibiliteit (en parallellisme) vereist is.
- Why do views and controllers always act in pairs?
    - De views zullen de aanpassingen detecteren en om deze aanpassingen te verwerken maken we gebruik van controllers.
    - Ook omgekeerd zullen controllers iets doen en de views notificeren om andere controllers te laten reageren.
- Explain the sentence “Restricts communication between subject and observer” in the Observer pattern
    - Observers kunnen de objecten niet aanpassen maar enkel toekijken.
- Can you explain the difference between an architecture and a pattern?
    - Architectuur opereert op een hoger abstractieniveau en biedt het de basis voor het systeem, terwijl patronen oplossingen bieden voor algemene problemen op een lager abstractieniveau. Ze voegen allebei duidelijkheid en begrip toe, maar opereren op verschillende abstractieniveaus
- Explain the key steps of the ATAM method?
    - Analyse architecture + scenarios, check their risks, sensitivity points and trade-offs, and reanalyse/reform.
- How can you balance emergent design with intentional architecture?
    - Het balanceren van opkomend ontwerp met opzettelijke architectuur impliceert het begrijpen van hun rollen, het gebruiken van architecturale start- en landingsbanen, balanceren met enablers, samenwerken tussen teams, evolueren met Agile Architecture en investeren in legacy-modernisering.
    - link: https://scaledagileframework.com/agile-architecture/#:~:text=capabilities%20of%20teams.-,Balancing%20Intentional%20and%20Emergent%20Design,-Traditional%20architecture%20approaches
- What happens when your team goes outside the boundaries of the guardrail?
    - Indien de team buiten de guardrails gaan kunnen er verschillende dingen fout lopen. Bijvoorbeeld een standaard van coderen wordt niet nageleefd, dit kan resulteren in verwarrende code of misverstanden van gebruik van code, wat dan kan resulteren in foute resultaten.
- How would you organize an architecture assessment in your team?
    - Architecture assessment omvat het identificeren van de behoefte, het definiëren van de scope, het selecteren van het team, het kiezen van de methode, het uitvoeren van de beoordeling en het beoordelen van de uitkomst.

Project Management



Summary i

- Name the five activities covered by project management.
  - Planning
    - Breakdown tasks + planning resources
  - Organization
    - Who does what?
  - Staffing
    - Recruiting and motivating personnel
  - Directing
    - Ensure team acts as a whole
  - Monitoring
    - Detect plan deviations + take corrective actions
- What is a milestone? What can you use them for?
  - Een milestone is een goal dat verifieert kan worden door de klant wanneer de taak voltooid wordt. Duidelijke milestones zijn belangrijk, een taak dat 80% klaar is, is niet duidelijk.
  - Wordt gebruikt om de progress van een taak de monitoren.
- What is a critical path? Why is it important to know the critical path?
  - Critical path: Is een pad waarbij een taak de hele pad kan vertragen indien dit taak een vertraging oploopt.

    - Het is belangrijk om te weten welke taken critical zijn omdat we dan de juiste recourses kunnen plaatsten op de juiste taken.
- What can you do to recover from delays on the critical path?
    - We kunnen dit oplossen in die 3 verschillende manieren
        - Senior/specialized staff voor specifieke taken
            - Buiten de scope van de kritische pad
        - Priorteer requirements en voltooi incrementally
            - Focus eerst op de belangrijkste taken
            - Testen blijft nog steeds een must
        - Extend deadline
- How can you use Gantt-charts to optimize the allocation of resources to a project?
    - Door gebruik te maken van de timeline, kunnen we de tijd van de werknemers onder de tijdlijn tekenen, zodat we de juiste taken op de juiste personen alloceren.
    - Shuffle, split en extend taken op basis van de tijdsloten. Verdeel taken evenly.
- What is a “Known known”, and “Unknown known” and an “Unknown Unknown”?
    - Known known
        - Dit zijn de dingen dat we weten en we kunnen daarvoor makkelijk vooruit voor plannen.
    - Unknown known
        - Dingen dat we niet weten maar wel voor kunnen vooruit voor plannen.
    - Unknown unknown
        - Dingen dat we niet weten en niet verwachten. We kunnen alleen er voor op de hoede zijn, dus we moeten een risk analyse doen hiervoor.
- How do you use PERT to calculate the risk of delays to a project?
    - Bepaal eerst likely time, optimistic time en pessimistic time.
    - Daarna bepalen we de estimated time = (OT + 4 * LT + PT) / 6
    - Daarna bepalen we de standaard deviatie tussen de taken: S(n) = (PT(n) - OT(n)) / 6
    - De taak met de hoogste standaard deviatie, heeft een hogere kans om een delay op te lopen, dus de meest risky.
- Why does replacing a person imply a negative productivity?
    - De nieuw personeel moet notities maken voor de informatie die zij gemist hebben, daarnaast hebben een lagere motivatie dat ook resulteert in lagere productiviteit.
    - Communicatie zorgt voor overhead, dat motivatie van andere personeel beïnvloed en resulteert in lagere productiviteit.
- What’s the difference between the 0/100; the 50/50 and the milestone technique for calculating the earned value?
    - 0/100
        - 0 wanneer we starten aan de taak
        - 100 wanneer het voltooid is
            - Dit is een zeer pessimistisch blik.
            - Meer voor kleiner taken
    - 50/50
        - 50% klaar indien we er aan starten
        - en 100% klaar wanneer we de taak voltooien.

        - Geeft een betere blik op maar nog steeds niet perfect (20/80 geeft een betere blik)
        - Geschikt voor grotere taken
    - milestone technique
        - We splitsen onze taak in verschillende milestones en berekenen onze progress op basis van #milestones done / #milestones.
            - Geeft een goed blik van onze progress
            - Geschikt voor grotere taken, maar taken bevatten veel milestones
            - Indien onze taken te groot zijn, kunnen we die milestones gebruiken om kleiner taken te maken en terug gaan naar 1 van de twee andere manieren.
- Why shouldn't managers take on tasks in the critical path?
    - Ze moeten managen en tegelijkertijd taken voltooien, dit is moeilijk om te doen.
    - Ze hebben algemeen minder tijd, ze moeten extra tijd open houden indien er iets misloopt.
- What is the "definition of done" in a Scrum project?
    - Het is een checklist van taken dat we moeten voltooien voor dat we een taak kunnen definiëren als klaar (done).
- Give a definition for a Squad, Tribe, Chapter and Guild in the spotify scrum model.
    - Squad
        - Is een scrum team
    - Tribe
        - Losjes gekoppelde Scrum Teams die werken aan gerelateerde features/componenten
    - Chapter
        - Teamleden met vergelijkbare expertise binnen een stam.
    - Guild
        - Teamleden met vergelijkbare interesses binnen verschillende tribes.

Summary ii

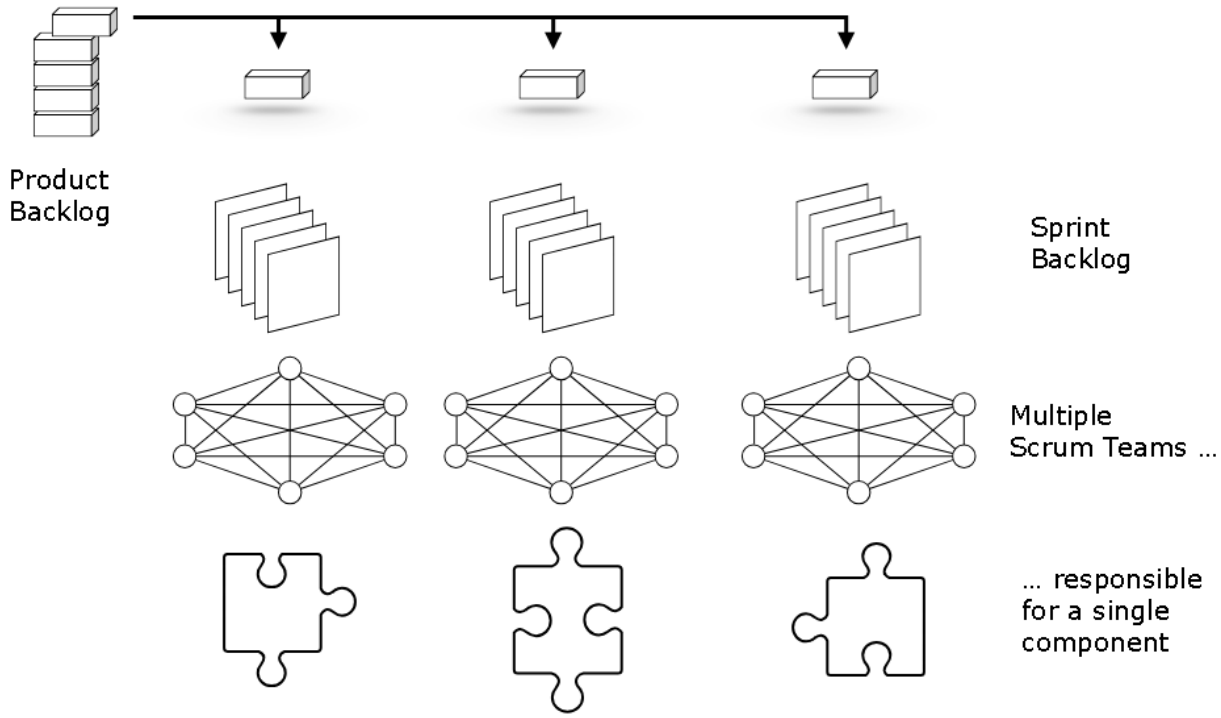
- Name the various activities covered by project management. Which ones do you consider most important? Why?
    - Activiteiten
        - Planning
        - Organization
        - Staffing
        - Directing
        - Monitoring
    - Planning & Monitoring zijn de belangrijkste in mijn mening
        - Planning heeft invloed op de andere 4 activiteiten, indien er goede planning is, is er ook minder werk voor de rest van de activiteiten.
        - Monitoring is daarnaast ook zeer belangrijk, dit is omdat we moeten nagaan of er deviaties zijn en dan gaan we terug naar het plannen indien nodig.
- How can you ensure traceability between the plan and the requirements/system?
    - Dit kan alleen als het goed gedaan wordt
        - Kleine taken.

        - Milestones moeten verifieerbaar zijn door de klant.
- Compare PERT-charts with Gantt charts for project planning and monitoring.
    - PERT
        - Good for: Task interdependencies
            - We kunnen zien welke taken afhankelijk zijn van elkaar.
            - Optimaliseer taak parallellisme.
            - We kunnen complexe afhankelijkheden monitoren.
            - Laat zien hoe lang elke taak duurt.
        - Good for: Critical path analyse
            - Bereken: ESD LED
            - Optimaliseren van recourses die gealloceerd wordt op de kritische pad
            - We kunnen de kritische pad analyseren.
        - Not for: Time management
    - Gantt
        - Good for: Time management
            - Taken in tijdformaat
            - Optimalisering van taken op basis van slack time
            - monitoren van kritische taken zonder slack time
        - Good for: Recourse and staff allocation
            - Shows recourse/staff allocatie
            - Optimalisering van "free time"
            - monitoren van bottlenecks (= fully occupied recourses / staff)
        - Not for: Task independencies
- How can you deal with “Unknown Unknowns” during project planning?
    - Risk analyse
    - Buffer toevoegen zodat de deviaties kunnen opvangen
- Choose between managing a project that is expected to deliver soon but with a large risk for delays, or managing a project with the same result delivered late but with almost no risk for delays. Can you argue your choice?
    - We kiezen liever voor de tweede
        - De eerste is verwacht dat het vroeg tijdig klaar zou zijn maar de risk kan zorgen voor de delays dat erger resulteert dan de tweede keuzen.
        - Terwijl we bij de tweede weten dat het al te laat zou zijn en van te voren de klant kunnen verwittigen. Er is weinig risico dus weinig kans op extra vertraging.
- Describe how earned-value analysis can help you for project monitoring.
    - Hiermee kunt u de bestede tijd vergelijken met de geplande tijd. Ook kunt u aan mensen vragen of er vertraging wordt verwacht.
- Would you consider bending slip lines as a good sign or a bad sign? Why?
    - bending slip lines naar links is een bad sign, want dit geeft als aanwijzing dat er een delay is. Indien we bending slip lines hebben naar rechts dan is het good sign, want dan zijn de taken eerder klaar dan verwacht.
- You’re a project leader and one of your best team members announces that she is pregnant. You’re going to your boss, asking for a replacement and for an extension of the project deadline. How would

you argue the latter request?

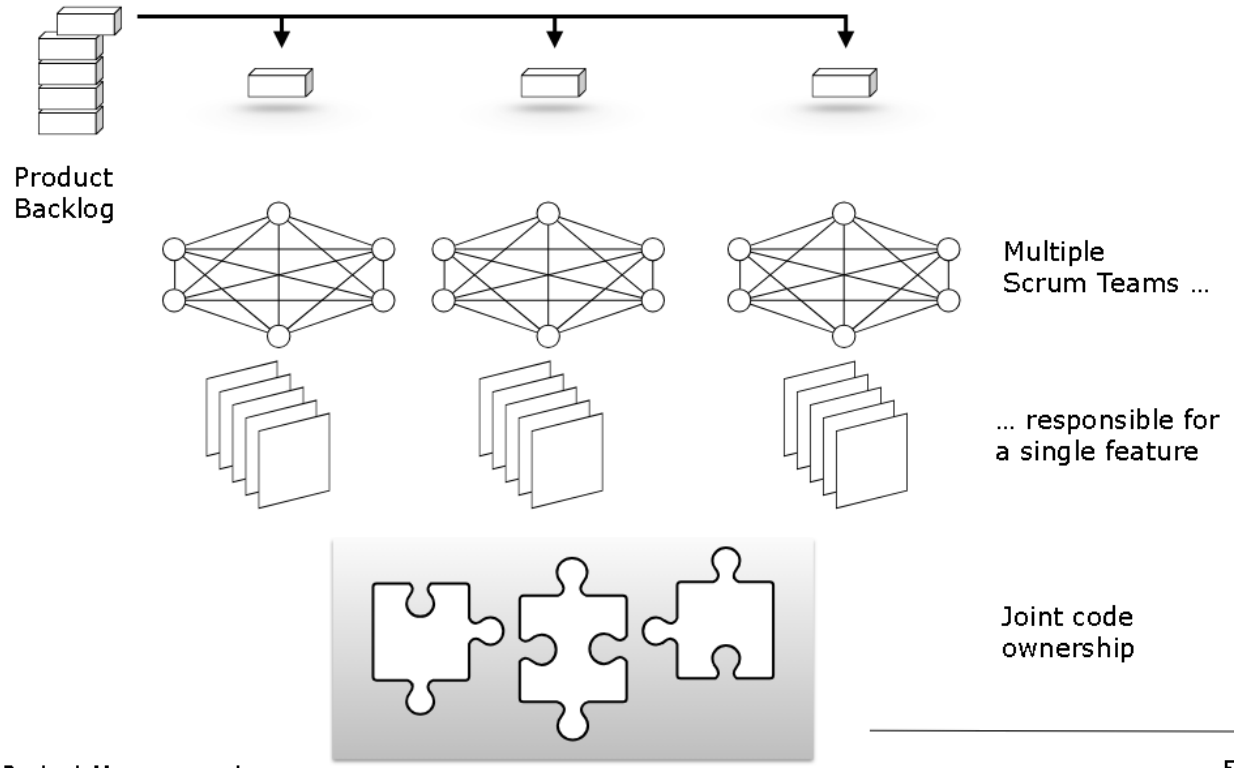
- Replacement zorgt voor veel problemen.
  - Verduidelijking van het project aan de replacement.
    - Zorgt voor communicatie overhead --> verlaagt productiviteit van andere leden.
    - Verlaagt motivatie van replacement --> minder motivatie resulteert dan ook in minder productiviteit.
  - Start met minder motivatie --> dus minder productiviteit.

- You have to manage a project team of 5 persons for building a C++ compiler. Which team structure and member roles would you choose? Why?
  - Consensus: Fast knowledge transfer
- Can you give some advantages and disadvantages of scrum component teams and scrum feature teams.
  - Component team
    - Waarbij elk team responsible is voor een component
      + Dit kan resulteren in componenten die mismatchen, hiervoor is er dan extra complexiteit nodig om ze te combineren.

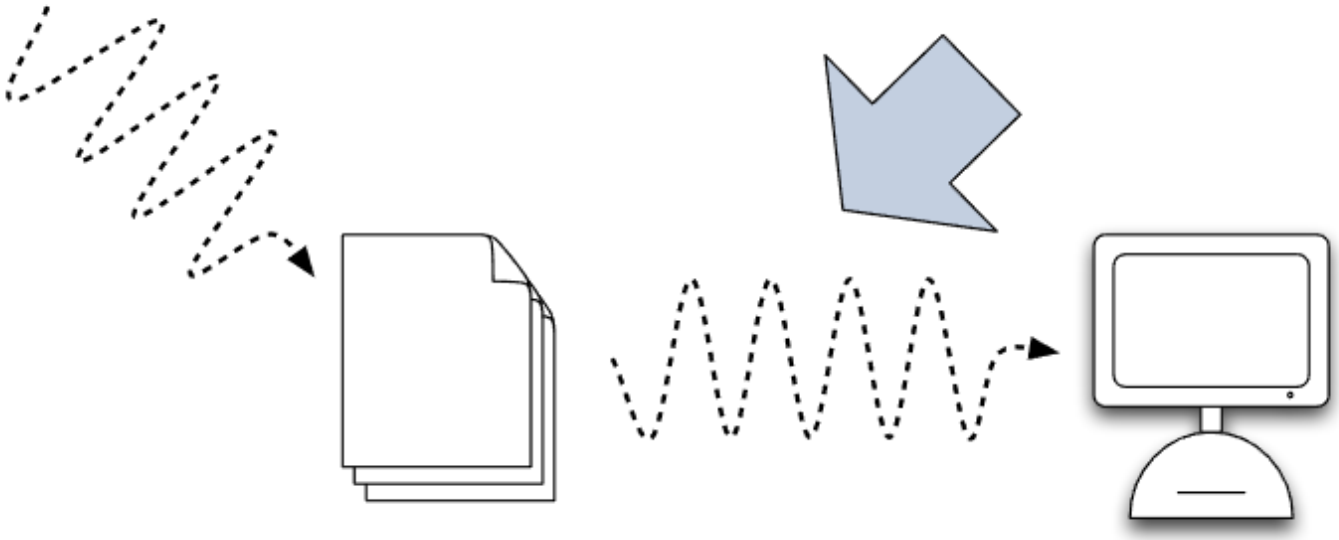


  - Feature team
    - Hier is elke team responsible for een single feature
      + Hier heeft iedereen toegang op de code maar omdat we feature werken, is dit veel

trager dan per component.



Design Contract



Summary i

- What is the distinction between Testing and Design by Contract? Why are they complementary techniques?
  - Testen probeert defecten na de feiten te diagnosticeren (en te genezen).
  - Design by Contract probeert defecten te vermijden.
  - Ze zijn complementary omdat we niet alle defecten kunnen vermijden als diagnosticeren.
- What's the weakest possible condition in logic terms? And the strongest?
  - Zwakste: True
  - Sterkste: False

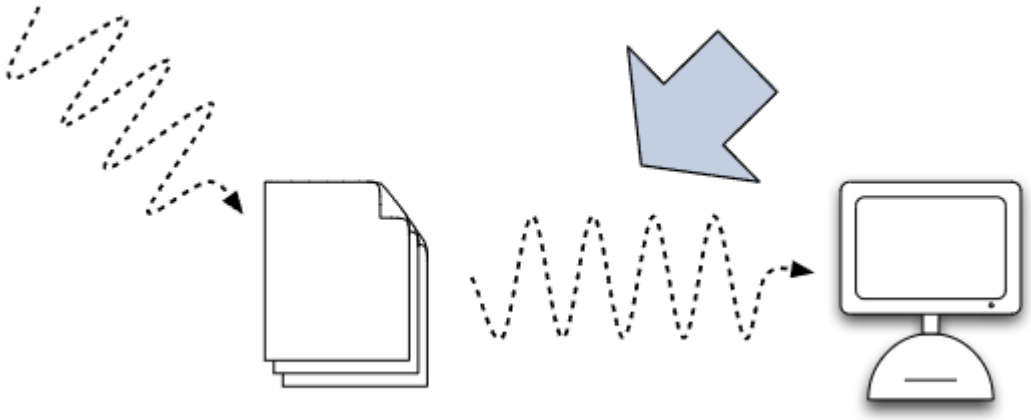
- If you have to implement an operation on a class, would you prefer weak or strong conditions for pre- and postcondition? And what about the class invariant?
  - Pre-conditie
    - Hier willen we een sterkere conditie omdat dit implementatie versimpeld.
  - Post-conditie
    - Hier willen we een zwakkere conditie omdat we dan makkelijker een einduitkomst kunnen bekomen.
  - Invariant
    - Maakt niet uit
- If a subclass overrides an operation, what is it allowed to do with the pre- and postcondition? And what about the class invariant?
  - Pre-conditie
    - Hier willen we een sterkere of gelijkwaardige conditie.
  - Post-conditie
    - Hier willen we een zwakkere of gelijkwaardige conditie.
  - Invariant
    - Gelijkwaardig
- Compare Testing and Design by contract using the criteria “Correctness” and “Traceability”.
  - Traceability
    - Beide garanderen traceability
      - Assertions is een manier om de requirements te mappen op de code
      - Via bepaalde testen kunnen we assertions terug op de requirements mappen. (Regression test)
  - Correctness
    - Beide garanderen geen correctness want ze zijn complementair, maar de som is meer waard dan enkel een deel.
      - We kunnen niet alle defecten vinden hiermee.
- What’s the Liskov substitution principle? Why is it important in OO development?
  - "*You may substitute an instance of a subclass for any of its superclasses, the program should not break*"
  - Het naleven van het LSP leidt tot robuustere, onderhoudbare en herbruikbare code, waardoor het een essentieel principe wordt in objectgeoriënteerd ontwerp.
- What is behavioural subtyping?
  - Het is een subtype relatie waarbij elk subtype object voldoet aan de specificatie van supertype.
- When is a pre-condition reasonable?
  - Het is reasonable wanneer de pre-conditie geëist wordt in de requirements.
  - Klanten moeten het kunnen bevredigen en controleren.

Summary ii

- Why are redundant checks not a good way to support Design by Contract?
  - Extra complexiteit
    - Extra code dat na gecheckt moet worden.

    - Performance penalty
        - Extra executie tijd nodig.
    - Wrong context
        - Een dienstverlener kan de situatie niet inschatten, alleen de consument kan dat.
- You’re a project manager for a weather forecasting system, where performance is a real issue. Set-up some guidelines concerning assertion monitoring and argue your choice.
    - Rule of thumb: *at least monitor pre-conditions*
        - Pre-condities moeten snel zijn
        - Hang niet af aan het uitzetten van "monitor pre-conditions" om efficiëntie te verbeteren.
        - Maak een profiel van de prestaties om te zien waar je efficiëntie verliest.
- If you have to buy a class from an outsourcer in India, would you prefer a strong precondition over a weak one? And what about the postcondition?
    - Een combinatie van beide is het best
        - Sterk --> Code is reusable maar te sterk maakt het te constraint misschien voor ons.
        - Zwak --> Minder moeite gedaan en meer kans op onverwachte resultaten.
    - Waarbij voor post-conditie we liever een zwakke post conditie hebben
        - Omdat we van een outsourcer code vragen en eventueel willen we een resultaat maar niet volledig op de manier die zij ons geven.
            - Bijvoorbeeld: een functie dat een gesorteerd lijst terug geeft, maar wij willen een niet gemanipuleerde lijst.
        - Indien de constraints zwakker zijn, kunnen wij onze eigen post-conditie erop plakken die wij denken dat nodig zijn.
- Do you feel that design by contract yields software systems that are defect free? If you do, argue why. If you don’t, argue why it is still useful.
    - Nee, ook al maken we heel ons code op basis van "design by contracts", er zullen altijd nog defecten zijn.
        - Dit is omdat er altijd de "Unkwown unkwowns" zijn, sommige dingen die we niet verwacht hebben.
- How can you ensure the quality of the pre- and postconditions?
    - Make ze redelijk en duidelijk.
- Why is (consumer-driven) contract testing so relevant in the context of microservices?
    - Contract testing is een cruciaal onderdeel om ervoor te zorgen dat microservices correct kunnen samenwerken, waardoor het een onmisbare strategie is voor applicaties met een microservices-architectuur.
- Assume you have an existing software system and you are a software quality engineer assigned to apply design by contract. How would you start? What would you do?
    - **Identificeer Contracten**
    - **Definieer Pre-condities en Post-condities**
    - **Pas Contracttesten toe**
        - Schrijf tests die controleren of aan de pre-condities, postcondities en invarianten wordt voldaan.
    - **Verfijn en check of de condities snel zijn**

Testing



Summary i

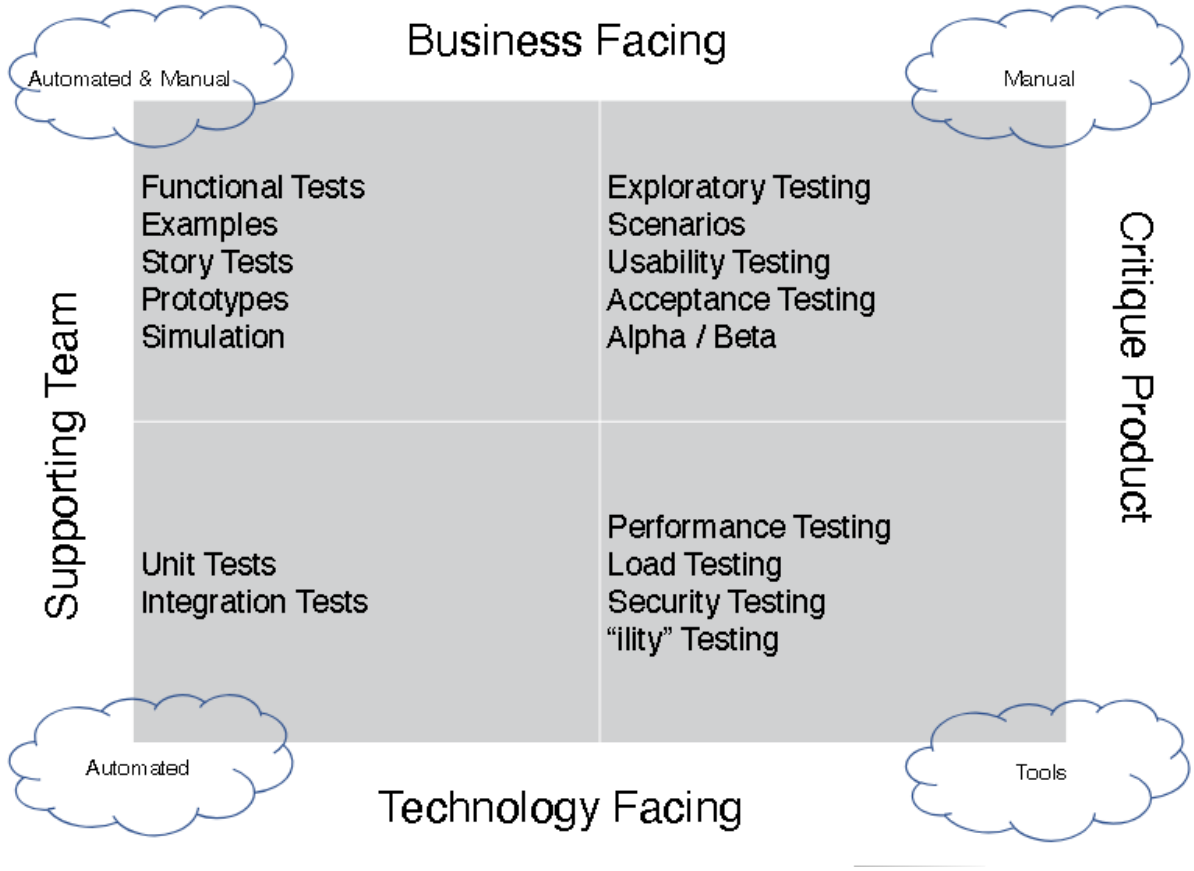
- What is (a) Testing, (b) a Testing Technique, (c) a Testing Strategy?
  - Testing
    - Het is een activiteit met de intentie of defecten te vinden.
  - Testing Technique
    - Dit zijn technieken om deze defecten te vinden met hoge succes.
  - Testing Strategy
    - Vertelt ons wanneer we welke techniek moeten toepassen om de juiste defecten te vinden.
- What is the difference between an error, a failure and a defect?
  - Defect
    - Een design of codering fout dat onverwachte gedrag vertoont.
  - Failure
    - Een afwijking tussen de specificatie en het draaiende systeem.
  - Error
    - De input dat een failure resulteert.
- What is a test case? A test stub? A test driver? A test fixture?
  - Test case
    - Een set van inputs en verwachte resultaten dat we toepassen op een component met de bedoeling van failures tegen te komen.
      - Als de component de verwachte resultaten brengt --> True
      - Else False
  - Test stub
    - Gedeeltelijke implementatie van componenten waarvan het geteste component afhankelijk is.
  - Test driver
    - Gedeeltelijke implementatie van een component die afhankelijk is van de geteste component.
  - Test fixture

        - Het doel van een test fixture is ervoor te zorgen dat er een bekende en vaste omgeving is waarin tests worden uitgevoerd, zodat de resultaten herhaalbaar zijn.
        - "*fixed state of software under test, baseline for running test*"
- What are the differences and similarities between basis path testing, condition testing and loop testing?
    - Basis path testing
        - Het is een soort white box testing die alle mogelijke onafhankelijke paden in de control-flow-graph van een programma test.
    - Condition testing
        - Het doel is het grondig testen van elke conditie of test die in de source code voorkomt.
    - Loop testing
        - Het is een type software testing dat wordt uitgevoerd om de loops te valideren.
    - Ze testen samen paden, maar controleren elk een ander aspect ervan. (Pad, conditions en loops)
- How many tests should you write to achieve MC/DC coverage? And multiple condition coverage?
    - Voor n condities
        - MC/DC (Modified condition / Decision coverage)
            - $n + 1$ test cases

Modified Condition/Decision Coverage					
a==1	b==1	c==1	d==1	decision	
TRUE	FALSE	TRUE	FALSE	return 1	+ row 4 shows effect of c
TRUE	FALSE	FALSE	TRUE	return 1	+ row 5 shows effect of a
FALSE	TRUE	FALSE	TRUE	return 1	+ row 5 shows effect of b
TRUE	FALSE	FALSE	FALSE	return 0	+ row 2 shows effect of d
FALSE	FALSE	FALSE	TRUE	return 0	

        - Multiple condition coverage
            - 2^n
- Where do you situate alpha/beta testing in the four quadrants model?
    - Manual
        - Business facing

- Critique facing

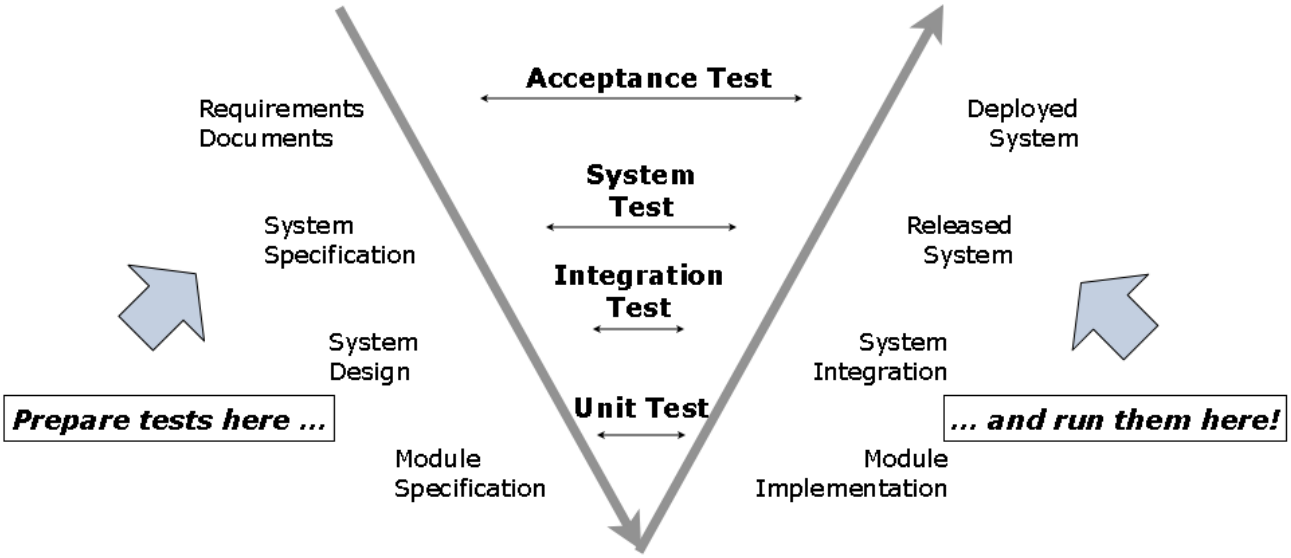


- What are the differences and similarities between unit testing and regression testing?
  - Het verschil is dat ze voor andere redenen testen:
    - Unit test: test dat de component juist werkt en dat ze juist gemaakt worden
    - Regression test: test dat alles nog werkt als vroeger, dus de originele werkende functionaliteiten zijn niet veranderd.
  - Gelijkenis:
    - Ze hebben als gelijkenis dat ze allebei nakijken dat het werkt als wat ervan gevraagd werd.
- How do you know when you tested enough?
  - Er is "geen" einde aan testen, elke run is een nieuwe test.
    - Elke bug-fix zal normaal gezien ook vergezeld worden met zijn eigen test.
  - Indien we geen geld/tijd meer hebben dan kunnen we dat wel als een einde zien.
- What is Alpha-testing and Beta-Testing? When is it used?
  - Alpha-testing
    - Een selectieve aantal mensen worden gekozen om het product te testen in een gecontroleerde omgeving.
  - Beta-testing
    - Het product staat publiek voor een selectieve doelgroep zonder enige invloed van de makers van het product.
- What is the difference between stress-testing and performance testing?
  - Stress-testing
    - We gaan het systemen overloaden en zien hoe het daarmee omgaat.
  - Performance testing
    - Hier kijken we gewoon naar performantie van het systeem.

- Bijvoorbeeld: Time consumption, memory consumption

Summary ii

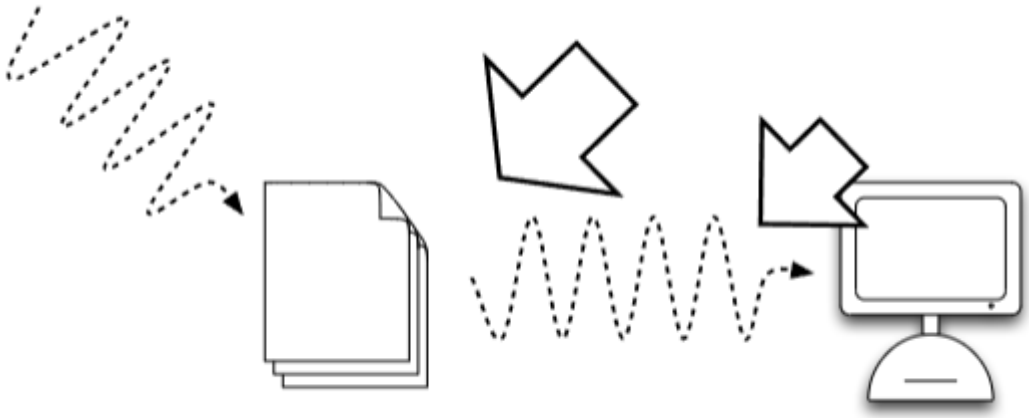
- You're responsible for setting up a test program. To whom will you assign the responsibility to write tests? Why?
  - De programmeur van de component.
  - Of een specialized test team.
  - Programmeurs zijn niet altijd de beste testers, dus daarom een specialized team.
- Why do we distinguish between several levels of testing in the V-model?
  - "*SPECIFY & DESIGN WITH TESTABILITY IN MIND*"



- Explain why basis path testing, condition testing and loop testing complement each other.
  - Ze testen allemaal verschillende aspecten van de pad.
- Why is mutation coverage a better criterion for assessing the strength of a test suite?
  - De mutation coverage biedt een uitgebreidere en rigoureuzere beoordeling van het vermogen van de testsuite om fouten te detecteren, waardoor het een waardevol hulpmiddel is voor het verbeteren van software test praktijken.
- Explain fuzzing (fuzz testing) in your own words.
  - Het is een techniek of security vulnerabilities te vinden, door grote random data door het systeem te sturen.
- Explain what FIT tables are.
  - FIT = Framework for Integrated Testing
    - Het is een tabel dat door klanten worden opgesteld, klanten geven een input van wat zij verwachten van het systeem. Daarna wanneer we de systeem testen, worden de tabellen ingekleurd op basis van of het systeem voldoet aan de beschrijvingen dat de klanten gaven.
- When would you combine top-down testing with bottom-up testing? Why?
  - **Grote softwareprojecten:** deze aanpak is handig als het gaat om grote softwareprojecten die meerdere subsystemen bevatten. Het maakt het mogelijk om zowel functionaliteiten op hoog niveau als componenten op laag niveau tegelijkertijd te testen
- When would you combine black-box testing with white-box testing? Why?

    - Het combineren van black-box- en white-box-testen kan een evenwichtige aanpak opleveren, waarbij zowel de functionaliteit als de implementatie van de software grondig worden getest.
- Is it worthwhile to apply white-box testing in an OO context?
    - White-box-testen is een waardevol hulpmiddel in een OO-context, omdat het vroege detectie van bugs mogelijk maakt, inzicht geeft in de interne werking van de software, grondigheid garandeert en helpt bij code-optimalisatie
- What makes regression testing important?
    - Het is belangrijk om na te gaan of vorige functionaliteiten nog werken of werken zoals origineel verwacht werd.
- Is it acceptable to deliver a system that is not 100% reliable? Why (not)?
    - Ja een systeem nooit 100% reliable, er kunnen altijd nog bugs inzitten dat we niet verwacht hadden. Het enige dat we wel kunnen doen is dat het consistent de juiste resultaten levert maar dit betekent niet dat het 100% reliable is.
- Explain the subtle difference between code coverage and test coverage.
    - Hoewel zowel code coverage als test coverage gericht zijn op het verbeteren van de code-effectiviteit, verschillen ze qua focus: code coverage richt zich op code-uitvoering, terwijl test coverage zich richt op functionele prestaties.

Formal Specifications

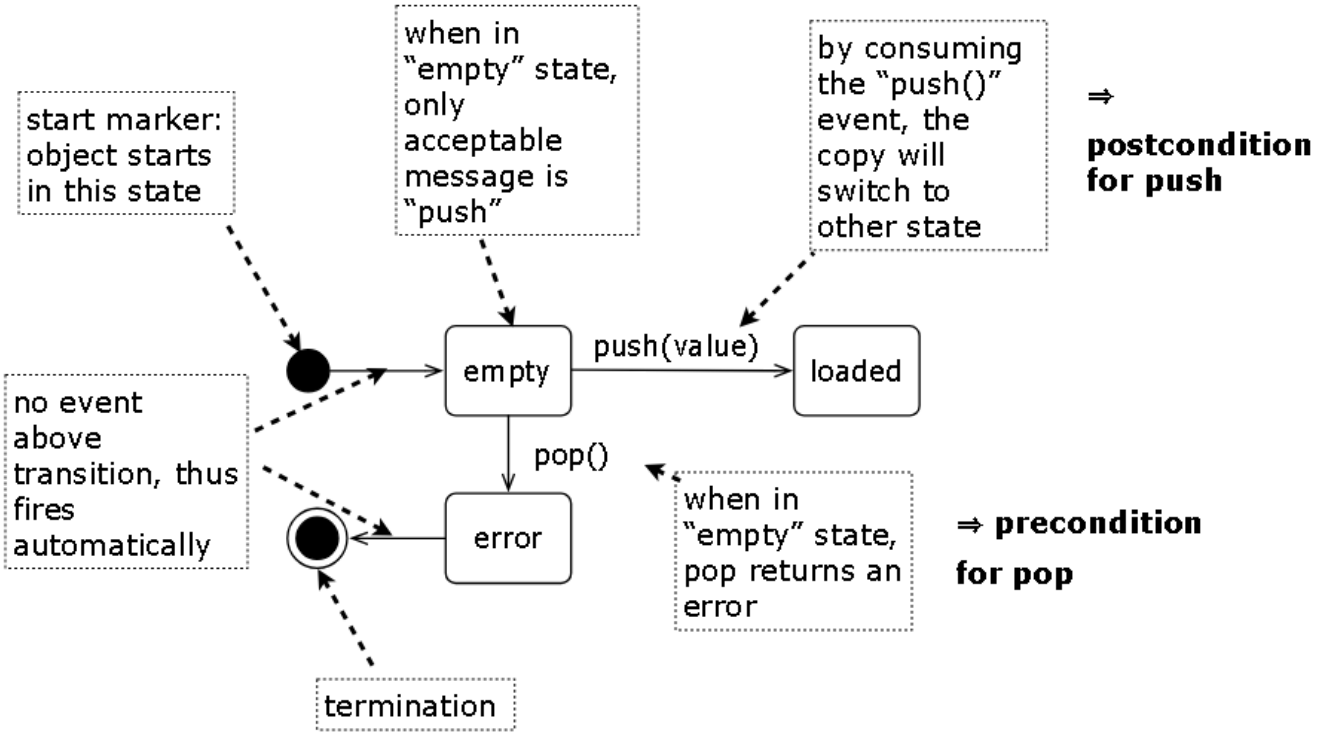


Summary i

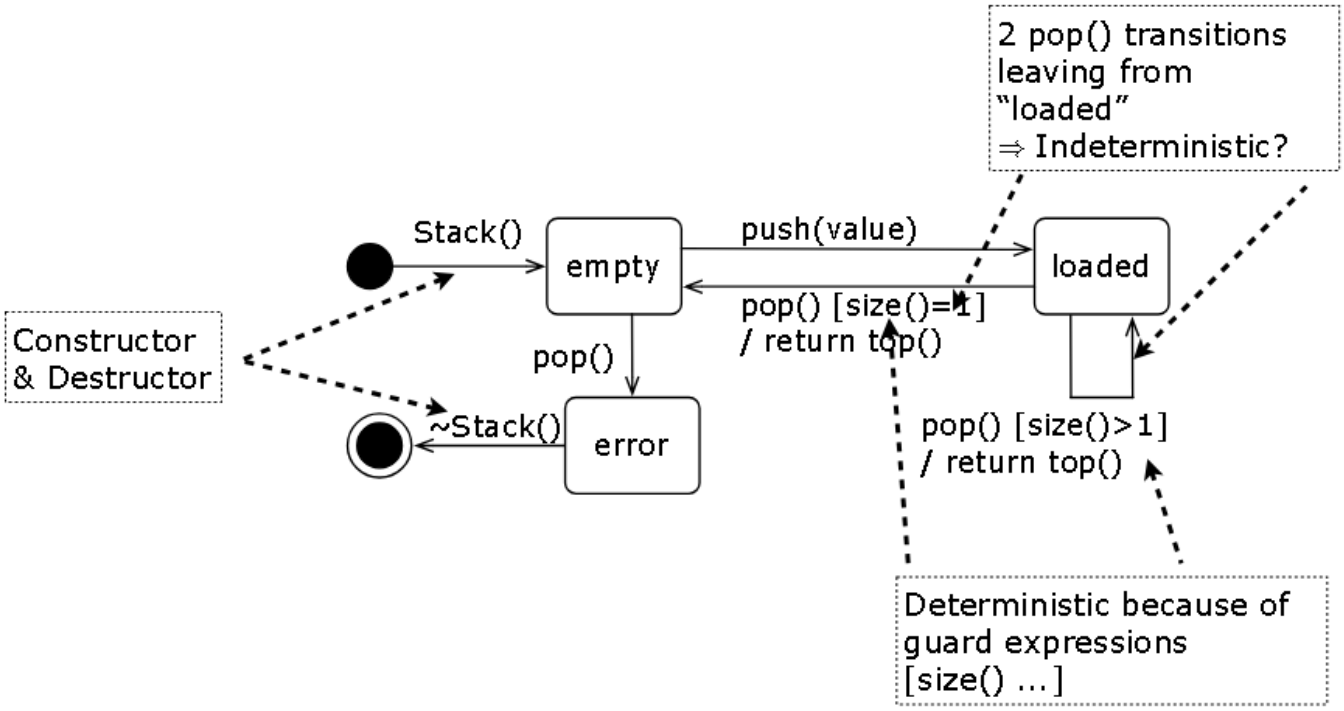
- Why is an UML class diagram a semi-formal specification?
    - **Informaliteit:** hoewel UML-klassediagrammen bepaalde regels en conventies volgen, zijn ze niet zo strikt gedefinieerd als formele specificaties. Dit betekent dat verschillende mensen hetzelfde diagram op enigszins verschillende manieren kunnen interpreteren.
- What is an automated theorem prover?
    - Het is een compiler support dat zegt "wanneer de pre-conditie waar is, dat de post-conditie normaal gezien moet voldaan worden".
- What is the distinction between “partially correct” and “totally correct”?
    - Partially correct
        - Assuming the precondition is true just before the function executes, then *if the function terminates*, the postcondition is true.
            - Infinite loops, exceptions, ...

    - Totally correct
        - Again assuming the precondition is true before function executes, t*he function is guaranteed to terminate* and when it does, the post-condition is true.
    - Het verschil is dat partially correct nog steeds kan lijden tot foute terminatie van de functie.
- Give the mathematical definition for the weakest precondition of Hoare triple {P} S {Q}
    - If ∀ P' such that {P'} S {Q}, P' ⇒ P, then P is the weakest precondition of S with respect to Q.
        - Denoted with wp(S, Q)
- Why is it necessary to complement sequence diagrams with statecharts?
    - Een statechart maakt het mogelijk om alle geldige (ook alle ongeldige) scenario's te specificeren.
- What is the notation for the start and termination state on a state-chart? What is the notation for a guard expression on an event?

Statechart for a Stack



Guarded Transition

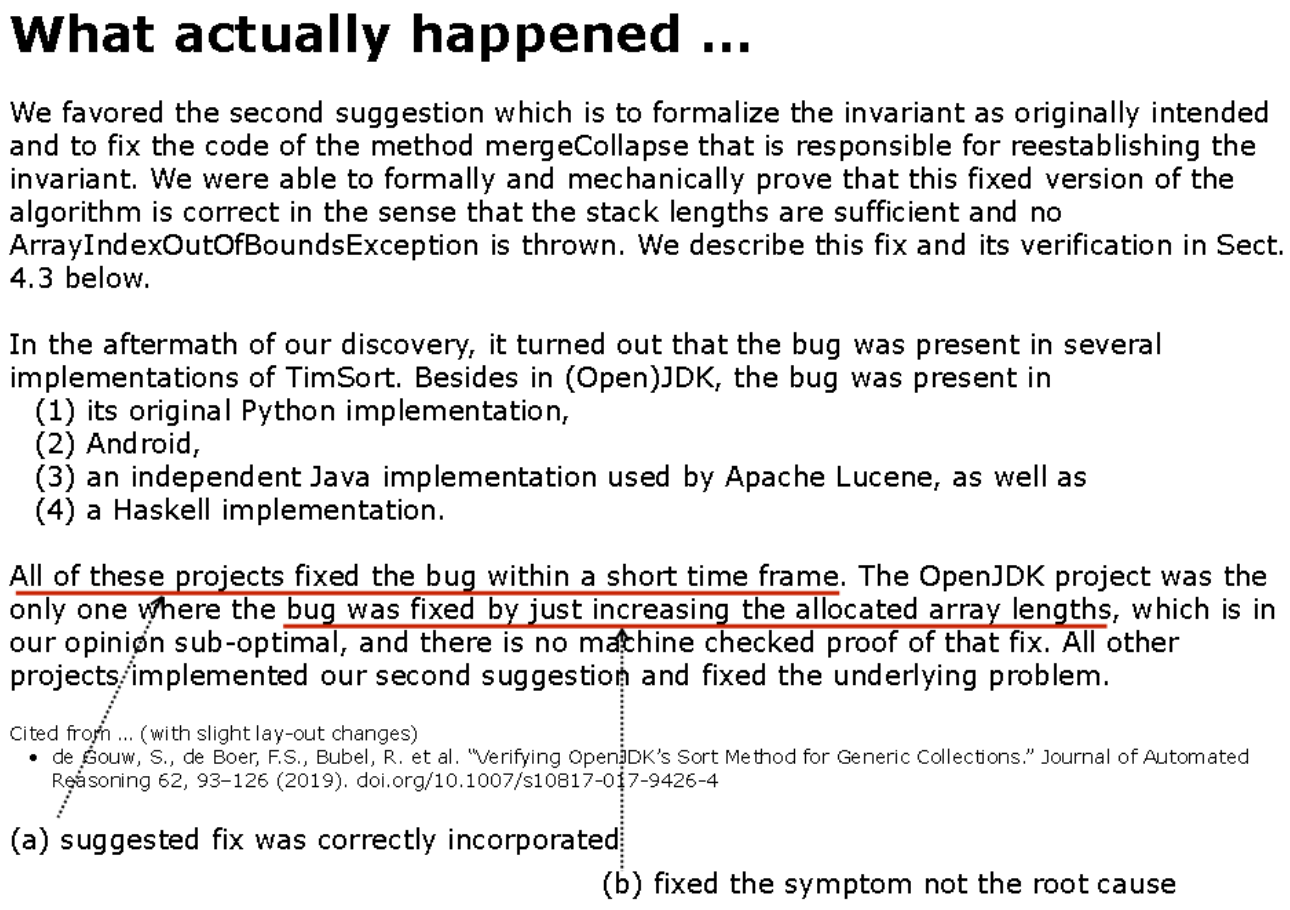


- What does it mean for a statechart to be (a) consistent, (b) complete, and (c) unambiguous?
  - Consistent
    - Elke staat is bereikbaar vanuit de oorspronkelijke staat.
  - Complete
    - Elk gebeurtenis/toestand-paar heeft een overgang.
  - Unambigious
    - Dezelfde gebeurtenis (incl. guard) komt niet op meer dan één overgang voor.
- How does a formal specification contribute to the correctness of a given system?
  - Verification

        - "*Formal specifications allowto verify presence of desired properties*"
    - Validation
        - "*Formal specifications can be simulated / animated*"

Summary ii

- Why is it likely that you will encounter formal specifications?
    - Softwareprojecten zijn meer afhankelijk van "Buy" dan van "Build", dus moeten de zaken formeel worden gespecificeerd.
- Explain why we need both the loop variant and the loop invariant for proving total correctness of a loop?
    - De loop invariant wordt gebruikt om de gedeeltelijke correctness te bewijzen (dat wil zeggen: als de lus eindigt, geeft deze het juiste resultaat), terwijl de loop variant wordt gebruikt om de beëindiging te bewijzen. Samen worden ze gebruikt om de totale correctness te bewijzen (dat wil zeggen, de lus eindigt en geeft het juiste resultaat).
- What do you think happened with the bug report on the broken Java.utils.Collection.sort()? Why do you think this happened?
    - "*They fixed the symptom and not the root cause; the risk is reduced but still not correct*."



What actually happened ...

We favored the second suggestion which is to formalize the invariant as originally intended and to fix the code of the method mergeCollapse that is responsible for reestablishing the invariant. We were able to formally and mechanically prove that this fixed version of the algorithm is correct in the sense that the stack lengths are sufficient and no ArrayIndexOutOfBoundsException is thrown. We describe this fix and its verification in Sect. 4.3 below.

In the aftermath of our discovery, it turned out that the bug was present in several implementations of TimSort. Besides in (Open)JDK, the bug was present in
(1) its original Python implementation,
(2) Android,
(3) an independent Java implementation used by Apache Lucene, as well as
(4) a Haskell implementation.

All of these projects fixed the bug within a short time frame. The OpenJDK project was the only one where the bug was fixed by just increasing the allocated array lengths, which is in our opinion sub-optimal, and there is no machine checked proof of that fix. All other projects implemented our second suggestion and fixed the underlying problem.

Cited from ... (with slight lay-out changes)
- de Gouw, S., de Boer, F.S., Bubel, R. et al. "Verifying OpenJDK's Sort Method for Generic Collections." Journal of Automated Reasoning 62, 93–126 (2019). doi.org/10.1007/s10817-017-9426-4

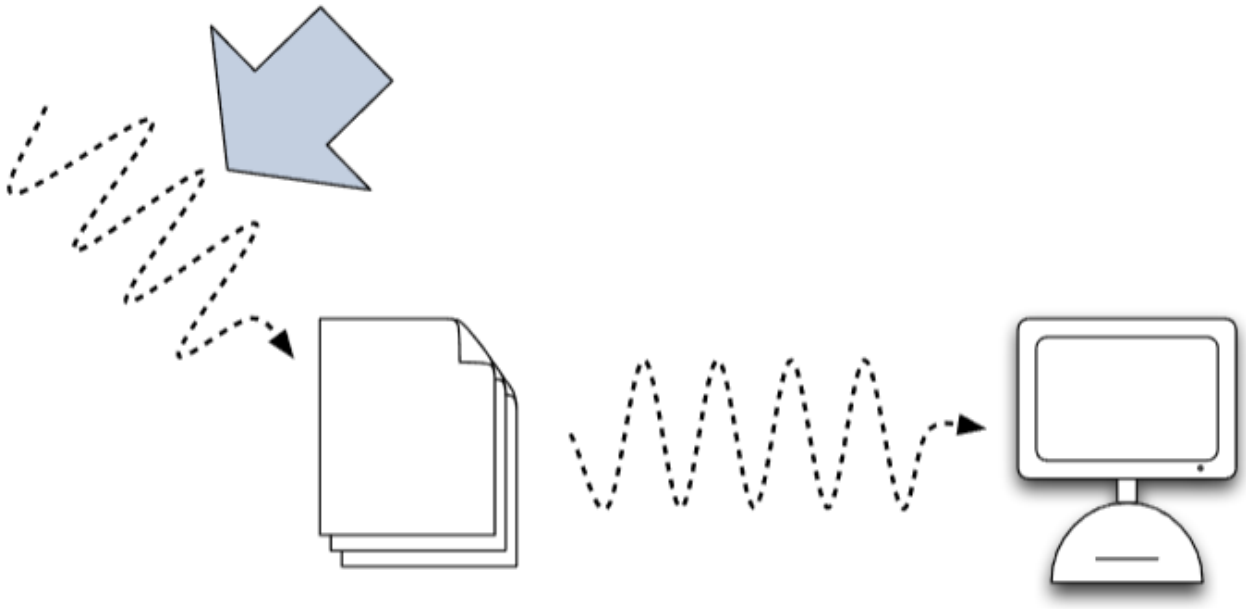
(a) suggested fix was correctly incorporated

(b) fixed the symptom not the root cause

- Explain the relationship between “Design By Contract” on the one hand “State based specifications” on the other hand.
    - Voor formele specificaties kunnen natuurlijke pre- en postcondities worden gebruikt.
- Explain the relationship between “Testing” on the one hand and “State based specifications” on the other hand.
    - Formele specificatie is black box-testen, maar met complete coverage (met de hoogste kans op het vinden van fouten).

- You are part of a team, build a fleet management system for drones transporting medical goods between hospitals. You must secure the system against cyber-attacks. Your boss asks you to look into formal specs; which ones would you advise and why?
    - High risk systems: human lives depend on the system.

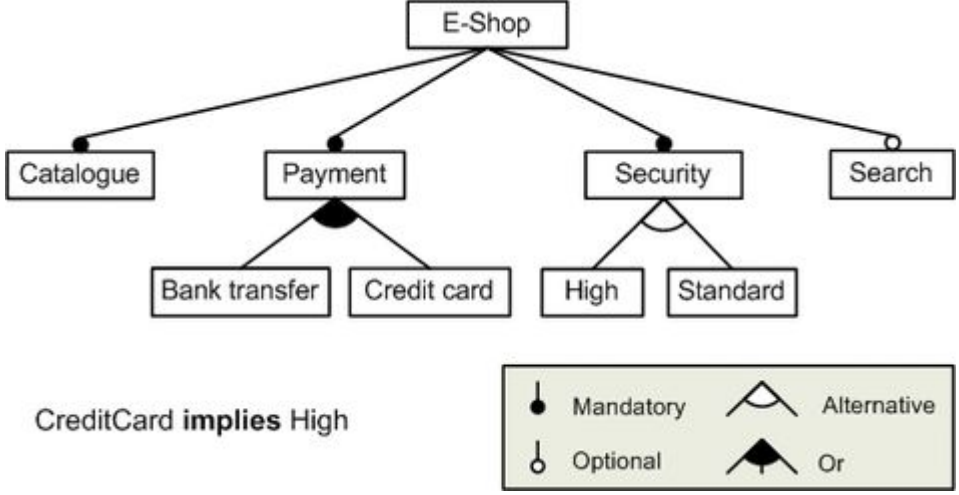
Domain Modelling



Summary i

- Why is it necessary to validate and analyse the requirements?
    - Validate:
        - Maken we de systeem juist?
    - Analyse:
        - Hebben we het probleem juist begrepen?
        - Maken we het juiste?
    - Cliënten weten niet wat ze willen en als ze dat wel weten dan gaan ze zeker van gedachten veranderen.
- What's the decomposition principle for functional and object-oriented decomposition?
    - Functional
        - Ontleed volgens de functies die een systeem moet uitvoeren.
            - "⇒ *single "subfunction-of" hierarchy*"
    - Object-oriented
        - Ontleden op basis van de objecten die een systeem moet manipuleren.
            - "⇒ *several coupled "is-a" hierarchies*"
- Can you give the advantages and disadvantages for functional decomposition? What about object-oriented decomposition?
    - Functional
        - Advantages
            - Goed met stabiele requirements of enkele functie
            - Duidelijke probleem decompositie strategie

        - Disadvantages
            - Naive: Moderne systemen zijn complexer dan 1 functie.
            - Maintainability: Functies evolueren door de tijd heen, dit moet dan aangepast worden door heel de systeem.
            - Interoperability: Samenwerking met andere systemen wordt zeer complex of onmogelijk.
    - Object-oriented
        - Advantages
            - Beter voor complexe en evoluerende systemen.
        - Disadvantages
            - Encapsulatie biedt robuustheid tegen typische veranderingen.
- How can you recognize “god classes”?
    - Veel kleine “provider”-klassen, die voornamelijk accessor-bewerkingen bieden.
        - Bijvoorbeeld: getters of setters
    - Inheritance hierarchy is gericht op hergebruik van gegevens en code.
    - Er zijn maar weinig grote ‘god’-klassen die het grootste deel van het werk doen.
- What is a responsibility? What is a collaboration?
    - Responsibility
        - De publieke diensten die een object kan leveren aan andere objecten, niet de methode van hoe dit gedaan wordt.
    - Collaboration
        - Andere objecten die nodig zijn om een verantwoordelijkheid te vervullen, lege samenwerkingen zijn mogelijk.
- Name 3 techniques to identify responsibilities.
    - Scenarios en Role Play
        - Voer scenario's van het systeem uit waarbij verschillende personen de klassen 'spelen' en hardop nadenken over hoe ze aan andere objecten zullen delegeren.
    - Identificatie van werkwoorduitdrukkingen
    - Class Enumeration
        - Som alle candidate classes op en bedenk een eerste reeks verantwoordelijkheden.
    - Hierarchy Enumeration
        - Noem alle klassen in een hiërarchie en vergelijk hoe zij verantwoordelijkheden vervullen.
- What do feature models define?
    - Definieert een reeks herbruikbare en configureerbare vereisten voor het specificeren van de systemen in een domain.
    - een model dat kenmerken en hun afhankelijkheden definieert, meestal in de vorm van een feature diagram.

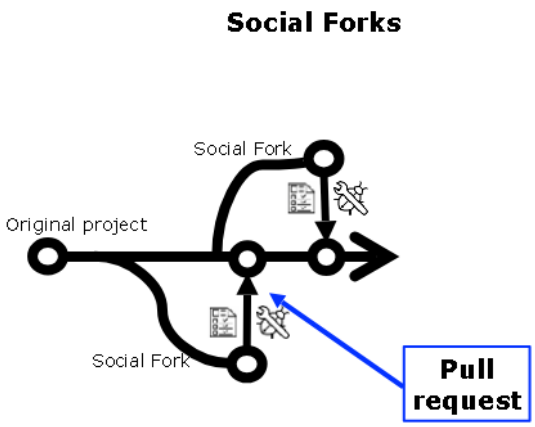


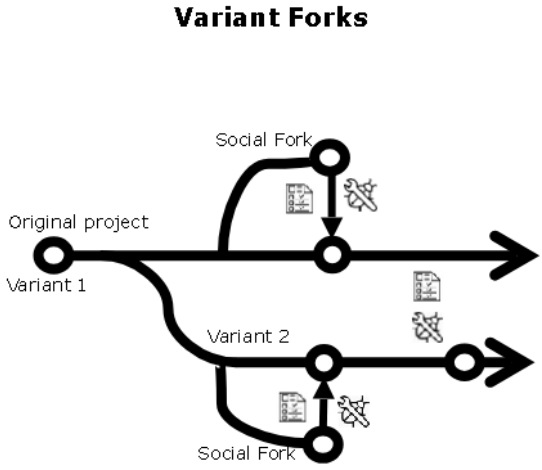
- Definieert de overeenkomsten en variaties tussen de leden van een softwareproductlijn.

- Give two advantages and disadvantages of a “clone and own” approach
  - Clone and own: "*Cloning an existing product variant, then modifying it to add and/or remove some functionalities, in order to obtain a newproduct variant.*"
    - Advantages
      - Efficiency
        - Saves time and reduces costs
        - Provides independence
        - Readily available
    - Disadvantages
      - Overhead
        - Het propageren van veranderingen.
        - Adapting the clone is difficult
        - Repetitive tasks are common
          - Bug fixes
        - "*Which variant to clone from?*"

- Explain the main difference between a social fork and a variant fork
  - Social fork
    - We creëren een aparte branche van de main fork, en indien de aanpassingen gemaakt zijn, mergen we die met de main branch.
  - Variant fork
    - Hier wordt iets gelijkaardigs gedaan maar in plaats te mergen met de main branch, kan de branch onafhankelijk blijven van de main branch. Dit kan resulteren in herhalende werk

dat al gedaan is in andere forks.





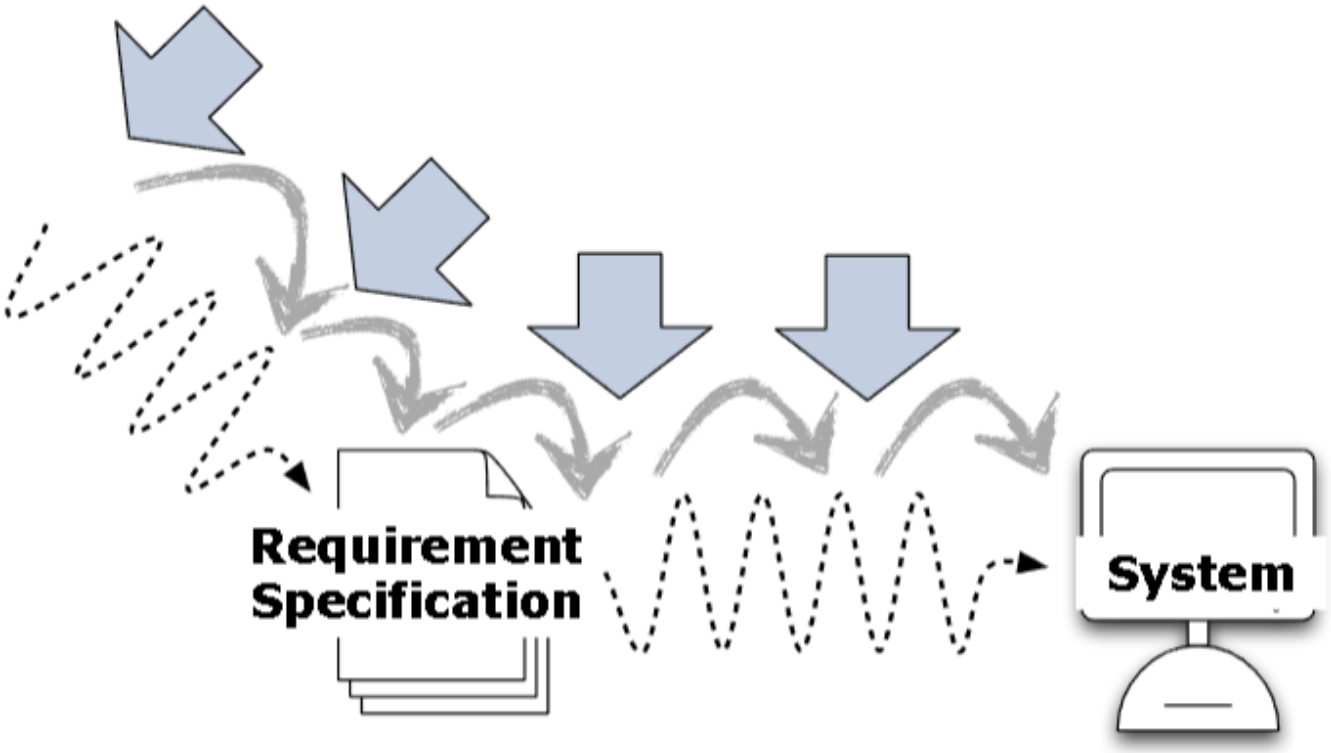
- How does domain modelling help to achieve correctness? Traceability?
  - Correctness
    - Verification: maken we het juist?
      - Goede onderhoudbaarheid via een robuust model van het probleemdomein.
        - We specificeren de "what" niet de "how".
    - Validation: maken we het juiste?
      - Modelleer het probleemdomein vanuit het perspectief van de klant.
      - Role-play helpt bij het valideren van use-cases
      - Functiediagrammen richten zich op overeenkomsten/variaties
  - Traceability
    - Requirements en systeem
      - Via propere naming conventies
      - Vooral de namen van classes en operaties

Summary ii

- How does domain modelling help to validate and analyse the requirements?
  - Role-playing helpt het valideren van de use-cases
  - Het modelleren van het probleem via de perspectief van de klant, helpt het analyseren van requirements.
  - CRC kaarten is een techniek dat gebruikt wordt om requirements te valideren (wordt gebruikt in role play).
- What's the problem with "god classes"?
  - Maakt onderhoud moeilijker
  - "*Knows to much, does to much*"
- Why are many responsibilities, many collaborators and deep inheritance hierarchies suspicious?
  - Responsibilities
    - Classes dat te veel responsibilities hebben worden te complex, waardoor maintainability moeilijker wordt.
  - Collaborators
    - Meer collaborators betekent een hogere coupling, hogere coupling resulteert in complexere maintainability en het evolueren wordt ook complexer.
  - Deep inheritance

        - Maintainability wordt nogmaals complexer en readability kan ook onduidelijk worden. Ook kan het zijn dat we unexpected behaviours tonen indien we aanpassingen maken in 1 van de superclasses.
- Can you explain how role-playing works? Do you think it helps in creative thinking?
    - Role-playing
        - Een groep van mensen spelen de rollen uit van het systeem.
        - Ze krijgen een handvol CRC kaarten met hun responsibilities.
        - Ze spelen hun rollen uit en kijken hoe of er aanpassingen moeten gedaan worden.
        - Herhaal deze stap tot dat de scenarios zo goed als perfect zijn.
    - Ja, het helpt met creative thinking. Dit is omdat we naar alle scenarios kijken en zien wat er gebeurd indien we een aanpassingen maken.
- Can you compare Use Cases and CRC Cards in terms of the requirements specification process?
    - Use cases
        - Een techniek om requirements te specificeren.
    - CRC kaarten
        - Een techniek om requirements te valideren.
- Do CRC cards yield the best possible class design? Why not?
    - Nee, het resulteert tot een goed oplossing maar niet de beste oplossing. Dit is omdat CRC kaarten een techniek is om de communicatie tussen de objecten te versoepelen.
    - Daarnaast ligt het denken van oplossing in de hand van de mens, dus het kan altijd zijn dat we oplossingen overzien, ook al doen we dit techniek 1000 keer.
- Why are CRC cards maintained with paper and pencil instead of electronically?
    - Niet iedereen is computer smart, het zijn niet alleen programmeurs die meedenken aan een oplossing.
    - Beter voor de klant
    - Makkelijk
    - Minder officieel
- What would be the main benefits for thinking in terms of “system families” instead of “one-of-a-kind development? What would be the main disadvantages?
    - Reusability van code is de main advantage
    - Een disadvantage is dan dat we niet goed om kunnen gaan met aanpassingen.
- Can you apply scrum to develop a product line? Argue your case.
    - Ja, omdat scrum werkt op basis van iteratief en incrementeel productie. Dit betekent dat we constant kunnen aanpassen indien nodig.
    - Daarnaast is scrum een goed techniek voor communicatie tussen leden.

Software Quality



Summary i

- Why is software quality more important than it was a decade ago?
  - Software is meer in de dagelijks leven dan vroeger. Niet alleen in algemene toepassingen maar ook in toepassingen waar leven op het spel staat of producten waar veel geld in geïnvesteerd wordt. Dit zorgt ervoor dat fouten grote problemen kunnen resulteren.
- Can a correctly functioning piece of software still have poor quality? Why?
  - Ja, ook al werkt het systeem, er kunnen verschillende functionele of non-functional requirements zijn dat niet kwalitatief zijn.
- If quality control can't guarantee results, why do we bother?
  - "*A quality process leads to a quality product*"
  - Daarnaast hoe meer we doen om fouten te vermijden, hoe hoger de kans is dat we wel resultaat krijgen.
- What's the difference between an external and an internal quality attribute? And between a product and a process attribute?
  - Internal vs External
    - External
      - Afgeleid van de relatie tussen omgeving en systeem/proces.
    - Internal
      - Direct afgeleid van de product- of procesbeschrijving.
  - Product vs Process
    - Product
      - Is wat geleverd wordt aan de klant.
    - Process
      - Process is wat de software product levert.
- What's the distinction between correctness, reliability and robustness?
  - Correctness

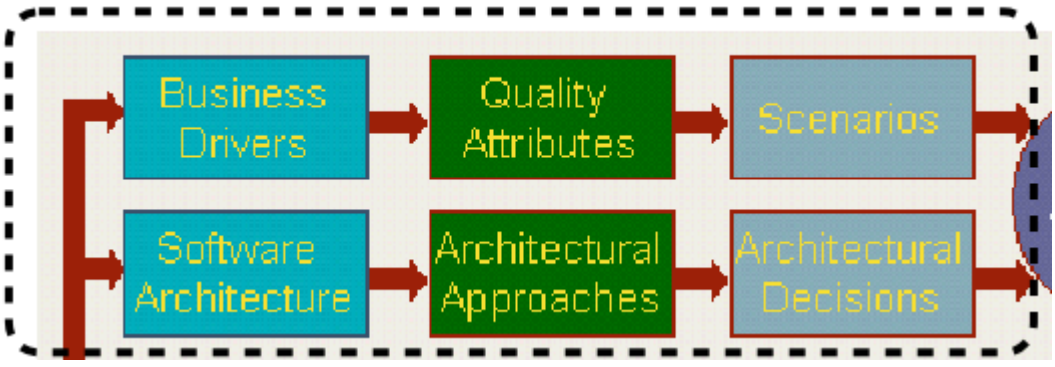
        - Een systeem is correct als het zich gedraagt volgens zijn specificaties.
    - Reliability
        - Indien een klant gebruik maakt van het systeem dan zou het de meeste keren werken zoals het moet werken.
        - De klant kan zonder enige concern gebruik maken van het systeem.
    - Robustness
        - Een systeem is robuust als het zich redelijk gedraagt, zelfs onder niet-gespecificeerde omstandigheden.
- How can you express the “user friendliness” of a system?
    - De mate waarin de menselijke gebruikers het systeem (proces) gemakkelijk te gebruiken vinden.
- Can you name three distinct refinements of “maintainability”? What do each of these names mean?
    - Repairability
        - Hoeveelheid werk dat nodig is om fouten te corrigeren.
    - Adaptability
        - Hoeveelheid werk dat nodig is om aanpassingen te maken volgens nieuwe requirements.
    - Portability
        - Hoeveelheid werk dat nodig is om naar een nieuwe environment of platform te gaan.
- What is meant with “short time to market”? Can you name 3 related quality attributes and provide definitions for each of them?
    - Time to market = de tijd tussen de overeenkomstige requirements en wanneer het deployed wordt.
    - => Short time to market = dit wilt zeggen dat we de gevraagde functies snel kunnen vrijgeven
    - 3 related quality attributes
        - Productivity
            - De hoeveelheid producten die we produceren tijdens een process met de gegeven recourses.
        - Timeliness
            - De mogelijkheid om het product op tijd vrij te geven.
        - Visibility
            - Huidige processtappen en projectstatus zijn toegankelijk.
- Name four things which should be recorded in the review minutes.
    - What was reviewed?
    - Who reviewed it?
    - What were the findings and what was the conclusion.
    - Decision
        - Accepted zonder enige nood aan aanpassingen
        - Provisionally accepted, er zijn nog aanpassingen nodig (geen extra review meer nodig)
        - Rejected, er zijn aanpassingen nodig en het moet daarna terug herbekeken worden.
- Explain briefly the three items that should be included in a quality plan.
    - Geef de gewenste productkwaliteiten weer en hoe deze worden beoordeeld.
        - Definieer de belangrijkste kwaliteitskenmerken.
    - Bepalen welke organisatorische normen moeten worden toegepast.

        - Meestal door middel van checklists en standaarden.
    - Het kwaliteitsbeoordelingsproces definiëren.
        - Meestal gedaan via kwaliteitsbeoordelingen na interne vrijgave.
- What's the relationship between ISO9001, CMMI standards and an organization's quality system? How do you get certified?
    - De kwaliteitsnormen (ISO9001 en CMMI) beïnvloeden het kwaliteitssysteem van de organisatie. Zo'n kwaliteitsorganisatie moet u vragen zich te laten certificeren, zij zullen dan het kwaliteitssysteem auditeren.
- Can you name and define the 5 levels of CMMI?
    - Level 1 Initial
        - "*No effective Quality Assurance procedures, quality is luck*"
    - Level 2 Managed
        - "*Formal Quality Assurance procedures in place (reactive*"
    - Level 3 Defined
        - "*Quality Assurance process defined and institutionalized*"
    - Level 4 Quantitatively managed
        - "*Quality Assurance Process + quantitative data collection*"
    - Level 5 Optimized
        - "*Improvement is fed back into Quality Assurance process*"
- Where would "use-cases" as defined in chapter 3 fit in the table of core process areas (p. 38)? Motivate your answer shortly.
    - Requirements management
        - De requirements bepalen welke use cases worden bepaald.

Summary ii

- Given the Quality Attributes Overview table, argue why the crosses and blanks occur at the given positions.
    - Er zijn vier elementen dat we bespreken:
        - Product: Wat geleverd wordt aan de klant
        - Process: Wat de software product oplevert.
        - External: Is wat we afleiden van de omgeving en systeem/process.
        - Internal: Is wat we direct kunnen afleiden van de product- of process beschrijving.
    - Indien er een kruis staat onder 1 van deze elementen dan weten we dat die een correlatie hebben hiermee.
    - Bijvoorbeeld maintainability: Het heeft een correlatie met het product, want deze attribute wordt pas toegepast na dat het de product geleverd is aan de klant. Daarnaast is het ook external, want we kunnen dit enkel afleiden van het systeem/process. Het is daarnaast niet internal want we kunnen niks afleiden van de beschrijvingen om te weten of er een variant van maintenance toegepast wordt. Het kan daarnaast wel process, indien we spreken over adaptability, want er kan integratie zijn in de software dat hier voor zorgt.
- Why do quality standards focus on process and internal attributes instead of the desired external product attributes?
    - "*a quality process leads to a quality product*"

- "*internal quality leads to external quality*"
- Why do you need a quality plan? Which topics should be covered in such a plan?
  - Een quality plan is belangrijk omdat we zo ons product op tijd en binnen budget kunnen houden.
  - Topics
    - Geef de gewenste productkwaliteiten weer en hoe deze worden beoordeeld.
      - Definieer de belangrijkste kwaliteitskenmerken.
    - Bepalen welke organisatorische normen moeten worden toegepast.
      - Meestal door middel van checklists en standaarden.
    - Definieer kwaliteitsbeoordelingsproces.
      - Meestal gedaan via kwaliteitsbeoordelingen na interne vrijgave.
- How should you organize and run a review meeting?
  - Organize
    - 3 - 5 mensen
    - 2 uur voorbereiding van te voren
      - Reviewers bereiden hun checklist voor
    - meeting duurt max 2 uur
  - How?
    - Who is the reviewer?
    - What is reviewed
    - What were the findings and conclusion
    - Decision
      - Accepted -> no further review
      - Provisionally accepted, correct defects (no further review)
      - Rejected, correct defects and follow-up review
- Why are coding standards important?
  - Coding standaarden helpen met leesbaarheid van code --> meer kwaliteit.
- What would you include in a documentation review checklist?
  - Project plan, Design, Testing, Requirements specification, code, ...
- How often should reviews be scheduled?
  - minstens 1 per week
- Could you create a review check-list for ATAM?
  - Ja we creëren onze vragen op basis van de volgende onderwerpen



    - Is there high coupling?
    - Is there low cohesion?
    - ...

- Would you trust software from an ISO 9000 certified company? And if it were CMMI?
  - Nee, we verwachten minstens een standaard van ISO 9001.
  - Voor CMMI willen we minsten een level 3 maturity level.
    - "***Proactive, rather than reactive.** Organization-wide standards provide guidance across projects, programs, and portfolios.*"
    - https://cmmiinstitute.com/learning/appraisals/levels
- You are supposed to develop a quality system for your organization. What would you include?
  - Indien we systeem willen bouwen kunnen we best volgens de normen werken. Is dat ISO 9001 of CMMI (minstens level 3), maakt niet zo veel uit.
  - Daarnaast voegen we code conventies toe, zodat de code leesbaar blijft en de kwaliteit verbetert.
  - Daarnaast hebben we quality plan nodig:
    - We definiëren hier dan onze set van kwaliteiten die we wensen en hoe we ze beoordelen.
    - Organisatie normen.
    - Kwaliteitsbeoordelingsproces.
      - Dit kan dan gedaan worden vie quality reviews. (Formal technical reviews, process reviews)
- Where would “testing” fit in the table of core process areas (p. 38). Does it cover a single row or not? Argue why (not)?
  - Process and Product Quality Assurance
  - Het hoort in deze "area", we willen na gaan of alles nog werkt (regression test en unit tests), dit hoort bij het "process" want we willen na gaan of het geleverde software nog werkt. Daarnaast maken we gebruik van tests om na te kijken of onze product die we afleveren aan de klant, werkt volgens de specificaties die geëist werden. Natuurlijk is dit geen perfecte manier en moeten we nog gebruik maken van andere technieken om dit verder te verifiëren.

Metrics

Summary i

- Can you give three possible problems of metrics usage in software engineering? How does the measurement theory address them?
  - Preciseness
    - Do we use the same units to compare?
    - Is the context the same?
      - familiar with language?
  - Representation Condition
    - Is “code size” really what we want to have?
      - What about code quality?
  - Scale and Scale Type
    - How do we want to interpret results?
  - GQM-paradigm
    - What do we want to do with the results?
- What’s the distinction between a measure and a metric?

- Measure
  - is een functie mapping
    - attribute van de echte wereld
  - op een symbool in een verzameling met bekende wiskundige relaties (= range)
- Metric
  - is een measure met als range de reële getallen en die voldoet
    - m(x,x) = 0
    - m(x,y) = m(y,x)
    - m(x,z) <= m(x,y) + m(y,z)

- Can you give an example of a direct and an indirect measure?
  - Direct
    - Length of code
    - Duration of process
    - Number of defects discovered
  - Indirect
    - Module Defect Density = Number of defects discovered / Length of source
    - Temperature is usually derived from the length of a liquid or metal
- What kind of measurement scale would you need to say “A specification error is worse than a design error”? And what if we want to say “A specification error is twice as bad as a design error?”
  - Ordinaal
  - Ratio
- Explain the need for a calibration factor in Putnam’s model.

$$Size = \mathrm{Pr}ocess\mathrm{Pr}oductivity \times \sqrt[3]{\frac{Effort}{\beta}} \times \sqrt[3]{Time^4}$$

  - De calibratie getal β is ongeveer gelijk aan 1
    - Indien we grote complexe projecten met een grote team hebben dan "> 1".
    - Indien we kleine simpele projecten met een kleine team hebben dan "< 1".
- Fill in the blanks in the following sentence. Explain briefly, based on the Putnam’s model. + If you want to finish earlier (= decrease scheduled time), you should ... the effort ... .
  - Increase
  - a lot
- Give three metrics for measuring size of a software product.
  - Lines of code
  - Function point
  - Use case points
- Discuss the main advantages and disadvantages of Function Points.
  - Advantages
    - Kan gemeten worden na de design
    - Is onafhankelijk van de implementation language
    - Meet functionaliteit

        - Werkt goed voor data processing
    - Disadvantages
        - Vereist subjectief experts oordeel
        - Kan niet automatisch berekent worden
- What does it mean for a coupling metric not to satisfy the representation condition?
    - Coupling is niet accuraat om een representatie te geven of er high coupling is.
- Can you give 3 examples of impreciseness in Lines of Code measurements?
    - "*Ignores software reuse, code duplication, benefits of redesign*"
    - "*The lower level the language, the more productive the programmer*"
        - Dan heeft LOC niet direct een indicatie want lower level language kan vaak veel lijnen of code hebben.
    - "*The more verbose the programmer, the higher the productivity*"
        - Duidelijker code, documentatie in de code.

Summary ii

- During which phases in a software project would you use metrics?
    - process, system, documentation, effort estimation, quality assurance
- Why is it so important to have “good” product size metrics?
    - Om de modellen goed te kalibreren (om een goede schatting te krijgen) zijn deze nodig voor algorithmic cost modelling.
- Can you explain the two levels of calibration in COCOMO (i.e. C & S vs. M)? How can you derive actual values for these parameters?
    - C & S: $C \cdot PM^S$
        - C is de complexity factor en S is de exponent dat bijna 1 is. Indien we complexere groter projecten hebben, dan is S groter.
        - C en S worden bepaald via regressie analyse met een database van 60 andere projecten.
    - M:
        - Met M zijn er meerdere formules
        - M is de calibratie factor dat tussen 0.7 en 1.66 zit.
            - houd rekening met quality attributes (reliability, performance)
            - en project constraints (tool usage, fast to market)
- Can you motivate why in software engineering, productivity depends on the scheduled time? Do you have an explanation for it?
    - Indien we eerder willen klaar zijn, dan kunnen we via Putnam's model zien dat we VEEL meer man maanden (een persoon die werkt voor een maand) nodig hebben.
    - En om de kost te verminderen, hebben we VEEL meer tijd nodig.
    - Indien we minder tijd hebben, is er meer onzekerheid
- Can you explain the cone of uncertainty? And why is it so relevant to cost estimation in software projects?
    - In het begin is er weinig informatie. Nadat we meer gedaan hebben en meer weten over product, wordt de cone kleiner. Dit is omdat er meer zekerheid is van hoeveel resources we nog moeten alloceren.

- Het is belangrijk omdat we de juiste hoeveelheid resources dan kunnen alloceren zodat we de best mogelijk situatie hebben.

- How can you decrease the uncertainty of a project bid using Putnam's model?
  - Meer tijd geven.
  - Meer effort.
  - We maken ook gebruik van de calibratie factor β om dit te calibreren.
- Why do we prefer measuring Internal Product Attributes instead of External Product Attributes during Quality Control? What is the main disadvantage of doing that?
  - External heeft een finished product nodig om, daarnaast is de nauwkeurigheid niet goed.
  - Internal is goedkoop en goed focus
    - Maar het is niet betrouwbaar
      - "*false positives: "bad" measurements, yet good quality*"
      - "*false negatives: "good" measurements, yet poor quality*"
    - Heavy weight approach
      - "*Requires team to develop/customize a quantitative quality model*"
      - "*Requires definition of thresholds (trial and error)*"
    - Moeilijk om het te interpreteren
      - "*Requires complex combinations of simple metrics*"
- You are a project manager and you want to convince your project team to apply algorithmic cost modelling. How would you explain the technique?
  - Choose system model
    - Formula consisting of product and process attributes + parameters
  - Calibrate system model
    - Choose values for parameters based on historical costing data
  - Measure (or estimate) attributes
    - Some attributes are fixed, others may vary

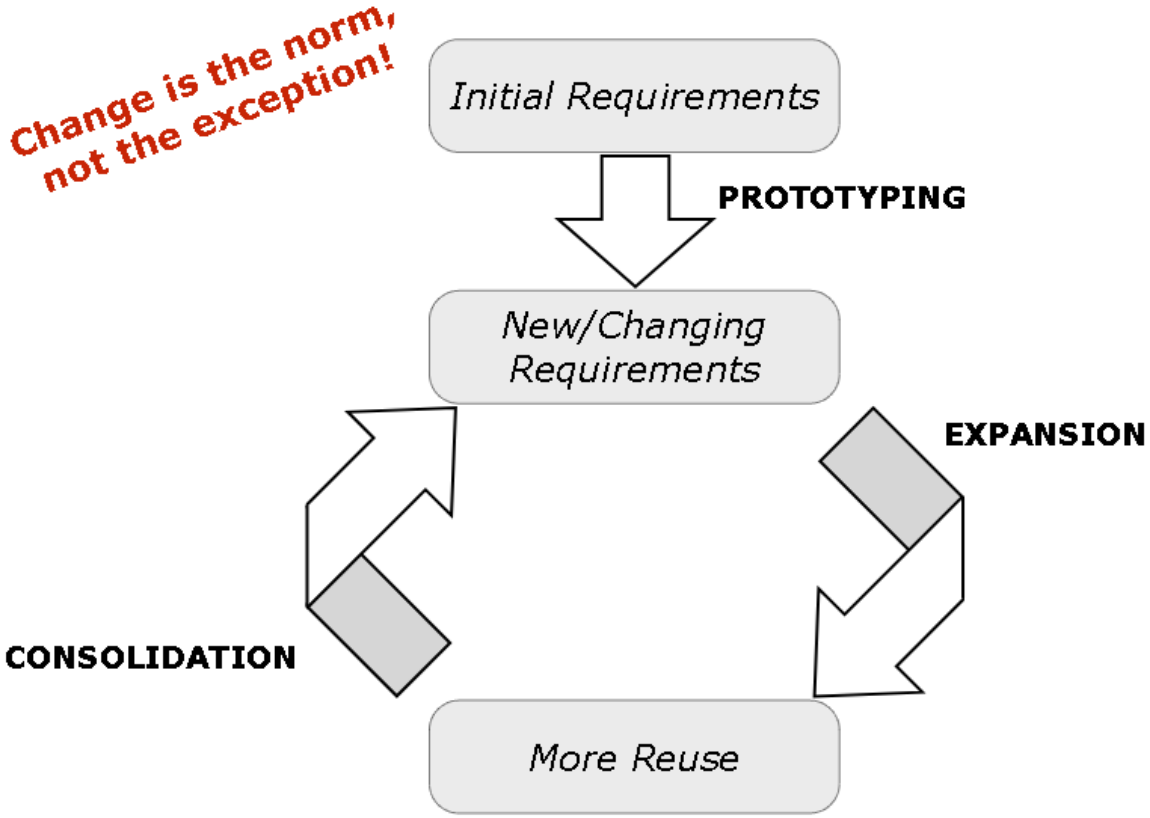
        - Calculate Effort
            - Iterate until satisfied
- Where would you fit coupling/cohesion metrics in a hierarchical quality model like ISO 9126?
    - Maintainability - Simplicity/modularity
- Why are coupling/cohesion metrics important? Why then are they so rarely used?
    - Omdat lage coupling en hoge cohesion willen, dit resulteert in veel voordelen.
    - Helaas is dit heel lastig om te bepalen omdat ze een slechte representatie weergeven.
        - Bijvoorbeeld: het kan zijn dat we lage coupling hebben maar dat we een hoge coupling waarde hebben.
- Do you believe that “defect density” says something about the correctness of a program? Motivate your answer?
    - Defect density
        - Is de hoeveelheid defects vergeleken de hoeveelheid lijnen code, meestal berekent om de 1000 lijnen code.
    - Indien we hoge defect density hebben, dan hebben we meer fouten per 1000 lijnen code. Dus correctness verlaagd.

Refactoring (Java)

Summary i

- Can you explain how refactoring differs from plain coding?
    - We veranderen onze originele code niet qua werking maar alleen qua interne structuur.
- Can you tell the difference between Corrective, Adaptive and Perfective maintenance? And how about preventive maintenance?
    - Corrective
        - De defecten oplossen.
    - Adaptive
        - Aanpassen tot nieuwe environments
    - Perfective
        - Extra features toevoegen, en onnodige features weg doen.
    - Preventive
        - Aanpassen zodat we defecten vermijden in de toekomst.
- Can you name the three phases of the iterative development life-cycle? Which of the three does refactoring support the best? Why do you say so?
    - Prototyping
    - Expansion

- Consolidation



- Can you give 4 symptoms for code that can be “cured” via refactoring?
  - Duplicate code
  - Nested conditionals
  - Large classes/methods
  - abusive inheritance
- Can you explain why add class/add method/add attribute are behaviour preserving?
  - Add class
    - Zolang de andere classes geen coupling hebben met deze nieuwe class, is er geen verandering aan behaviour.
  - Add method
    - Een methode toevoegen zal enkel extra functionaliteit toevoegen.
  - Add attribute
    - Zelfde met attributen, we geven enkel extra informatie.
  - Dus de main reason is dat de originele code hetzelfde blijft omdat we geen aanpassingen brengen in de originele code. We voegen enkel extra functionaliteiten toe.
- Can you give the pre-conditions for a “rename method” refactoring?
  - Does the name already exist?
- Which 4 activities should be supported by tools when refactoring?
  - rapid edit-compile-run cycles
  - reverse engineering facilities
  - regression testing
  - version & configuration management
- Why can’t we apply a “push up” to a method “x()” which accesses an attribute in the class the method is defined upon (see Refactoring Sequence on page 24)?

- omdat we de attribute niet kunnen accessen in de andere class. Dus we moeten de attribute ook "push up" refactorren.

Summary ii

- Why would you use refactoring in combination with Design by Contract and Regression Testing?
  - Design by contract zorgt er voor dat de aanpassingen defect loos zijn en regression testing zorgt er voor dat er geen verandering is van functionaliteiten, dus alles werkt nog als vroeger.
- Can you give an example of a sequence of refactorings that would improve a piece of code with deeply nested conditionals?
  - Extract method, indien de condities types zijn, kunnen we eventueel nog polymorphism gebruiken.
- How would you refactor a large method? And a large class?
  - Large method
    - Opsplitsen in kleinere methodes.
  - Large class
    - Indien we veel functionaliteiten hebben met gerelateerde doelen, omzetten naar eigen klasse.
- Consider an inheritance relationship between a superclass “Square” and a subclass “Rectangle”. How would you refactor these classes to end up with a true “is-a” relationship? Can you generalise this procedure to any abusive inheritance relationship?
  - Als we kijken naar beide klassen, dan zien we dat beide niet echt een correlatie hebben met elkaar om een "is-a" relatie te creëren. Daarom maken we een extra klassen hier, dat hun gemeenschappelijk maakt, bijvoorbeeld klasse "Shape".
  - Identify the Inappropriate Inheritance: Determine if the subclass is not a more specific form of the superclass.
  - Procedure
    1. **Find a Common Superclass:** Identify a more general concept that both the current superclass and subclass can fall under.
    2. **Create the New Superclass:** Create a new superclass based on the more general concept identified in step 2.
    3. **Move Common Attributes and Methods:** Move any attributes or methods that are common to both the original superclass and subclass to the new superclass.
    4. **Refactor the Original Classes:** Refactor the original superclass and subclass to inherit from the new superclass.
    5. **Test the Refactored Classes:** Ensure that the refactored classes work as expected and that the system behaviour is preserved.

Conclusion

Summary i

- Name 3 items from the code of ethics and provide a one-line explanation.
  - Public
    - Software developers onderhandelen met als doel van de publieke interesse.

    - Client and employer
        - Software engineers zullen onderhandelen in de beste interesse van de client en de werkgever onderhandelt in de doel van de publieke interesse
    - Product
        - Software engineers zullen ervoor zorgen dat hun product en de modificaties die ze maken aan de hoogste standaarden voldoet.
    - Judgment
        - Software engineers behouden hun integrity en independence in hun professionele beoordeling.
    - Management
        - Managers en leiders op het gebied van software-engineering moeten een ethische benadering van het beheer van de ontwikkeling en het onderhoud van software onderschrijven en bevorderen.
    - Profession
        - Software engineers moeten de integriteit en reputatie van het beroep bevorderen in overeenstemming met als belang de publieke interesse.
    - Colleagues
        - Software engineers zullen eerlijk en ondersteunend zijn tegen hun collega's
    - Self
        - Software engineers nemen deel aan een leven lang leren over de uitoefening van hun beroep en bevorderen een ethische benadering van de beroepsuitoefening.
- If you are an independent consultant, how can you ensure that you will not have to act against the code of ethics?
    - Voeg een extra clause toe in de contract, indien het tegen de "*code of ethics*" gaat, dat je de recht hebt om het contract te beëindigen.
- What would be a possible metric for measuring the amount of innovation of a manufacturing company?
    - Algemeen
        - Technologie evolueert om de 20 jaar.
        - Business model blijft hetzelfde
    - ICT
        - Technologie evolueert om de 5 jaar
        - Business model veranderd vaak
- Explain the 2 main steps of test amplification: input amplification and assertion amplification
    - Assert amplificatie

- Extra testen toevoegen om de staat van het object na te kijken tijdens de run van de code.


```
1  def testDeposit_amplified (self) :
2    self.b.set_owner('Iwena Kroka')
3    self.b.deposit(10)
4    self.assertEqual(self.b.
5            get_transactions(), [10])
6    self.assertFalse(self.b.is_empty () )
7    self.assertEqual(self.b.owner, 'Iwena Kroka')
8    self.assertEqual(self.b.get_balance(), 10)
     …
```

- Input amplificatie
  - Extra input testen toevoegen om edge cases te testen. We veranderen of voegen extra testen toe, om paden die we ervoor nog niet afgelopen hebben nog eens te testen.
    - Brute force maar optimaliseert door code coverage te vergroten.

When you chose the “No Silver Bullet” paper

- What’s the distinction between essential and accidental complexity?
- Name 3 reasons why the building of software is essentially a hard task? Provide a one-line explanation.
- Why is “object-oriented programming” no silver bullet?
- Why is “program verification” no silver bullet?
- Why are “components” a potential silver bullet?

When you chose the “Killer Robot” paper

Case: https://gelzaragoza.medium.com/comprehensive-analysis-of-the-case-of-the-killer-robot-b4f2146aeb21

- Which regression tests would you have written to prevent the “killer robot”?
  - Tests dat de robot nog steeds passive blijft tegen de mens. Dus dat het niet mensen dood.
- Was code reviewing applied as part of the QA process? Why (not)?
  - Ja, maar was niet afgemaakt. "*One co-worker told of an incident in which Samuels stormed out of a quality-assurance meeting.*".
- Why was the waterfall process disastrous in this particular case?
  - Hierdoor waren er fouten niet op tijd opgemerkt, waardoor er formule fouten waren die uiteindelijk resulteerde in iemands leven.
- Why was the user-interface design flawed?
  - "*Because it violated every Shneiderman's rule, not to mention the console and the comfort for the operator.*"
  - Waar Shneiderman's rule zegt: "***For every user action, there should be an interface feedback**. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial.*"

Summary ii

- You are an experienced designer and you heard that the sales people earn more money than you do. You want to ask your boss for a salary-increase; how would you argue your case?
  - Omdat het een goed product is, dat het makkelijker te verkopen is.
  - Omdat iedereen gelijkwaardig moet behandeld worden, jij hebt evenveel moeite gedaan in het product als iemand die het verkoopt. Code of ethics: Colleagues
- Software products are usually released with a disclaimer like “Company X is not responsible for errors resulting from the use of this program”. Does this mean that you shouldn’t test your software? Motivate your answer.
  - Indien we 100% zeker weten dat het niet de standaarden overschrijft, dan weten we dat alles volgens de publieke interesse werd gedaan.
  - Algemeen is het altijd noodzakelijk om testen uit te voeren, uiteindelijk is de software gemaakt door iemand anders en het kan altijd zijn dat er defecten zijn dat overlooked zijn. Zelfs als het software gemaakt is door jezelf, zijn testen noodzakelijk, er kunnen altijd verbetering zijn (refactoring).
  - De disclaimer dient enkel om niet liable te zijn indien je voor de rechter komt.
- Your are a QA manager and are requested to produce a monthly report about the quality of the test process. How would you do that?
  - We willen uiteindelijk nakijken of de kwaliteit gegarandeerd is of dat er aanpassingen nodig zijn.
  - We kunnen simpel, en goedkoop nakijken of software voldoet aan kwaliteit door gebruik te maken van FTR (formal technical reviews).
  - We kunnen daarnaast ook gebruik maken van standaarden, om na te gaan of het gecertificeerd kan worden.
  - Daarnaast maken we ook gebruik van testen, om na te gaan of de systeem zo juist was als ervoor, om na te gaan of de kwaliteit niet gedaald is vergeleken vorige maanden.
  - Daarnaast kunnen we gebruik maken van metrics, om verschillende aspecten in maatwaardes uit te drukken, bijvoorbeeld defect density.
- Why is “explainable Artificial Intelligence” so important when creating bots for software engineering tasks?
  - Dit heeft deels te maken met de ethische waardes van software. Indien AI niet explainable, kan het zijn dat het onverwachte resultaten vertoont, dat ethisch en niet ethisch kan zijn.
  - Indien het explainable is, is er een hogere kans dat het moet werken zoals verwacht. Dit verhoogt het vertrouwen en zekerheid dat de AI niks verkeerd doet.

When you chose the “No Silver Bullet” paper

- Explain why incremental development is a promising attack on conceptual essence. Give examples from the different topics addressed in the course.
- “Software components” are said to be a promising attack on conceptual essence. Which techniques in the course are applicable? Which techniques aren’t?

When you chose the “Killer Robot” paper

- Recount the story of the Killer Robot case. List the three most important causes for the failure and argue why you think these are the most important.
  - "*Bad interface design (because of the death), the waterfall model (because of the approach) and faked tests (because of prevention).*"