

SOFTWARE DESIGN ESSENTIALS

SOFTWARE DESIGN ESSENTIALS



Contact

Johan.van.den.Broek@ehb.be



Canvas

Cursusmateriaal

Exameninfo

Aankondigingen

Oefeningen & oplossingen

OVERZICHT

SOFTWARE DESIGN ESSENTIALS

 Software ontwikkelproces

 UML: use cases

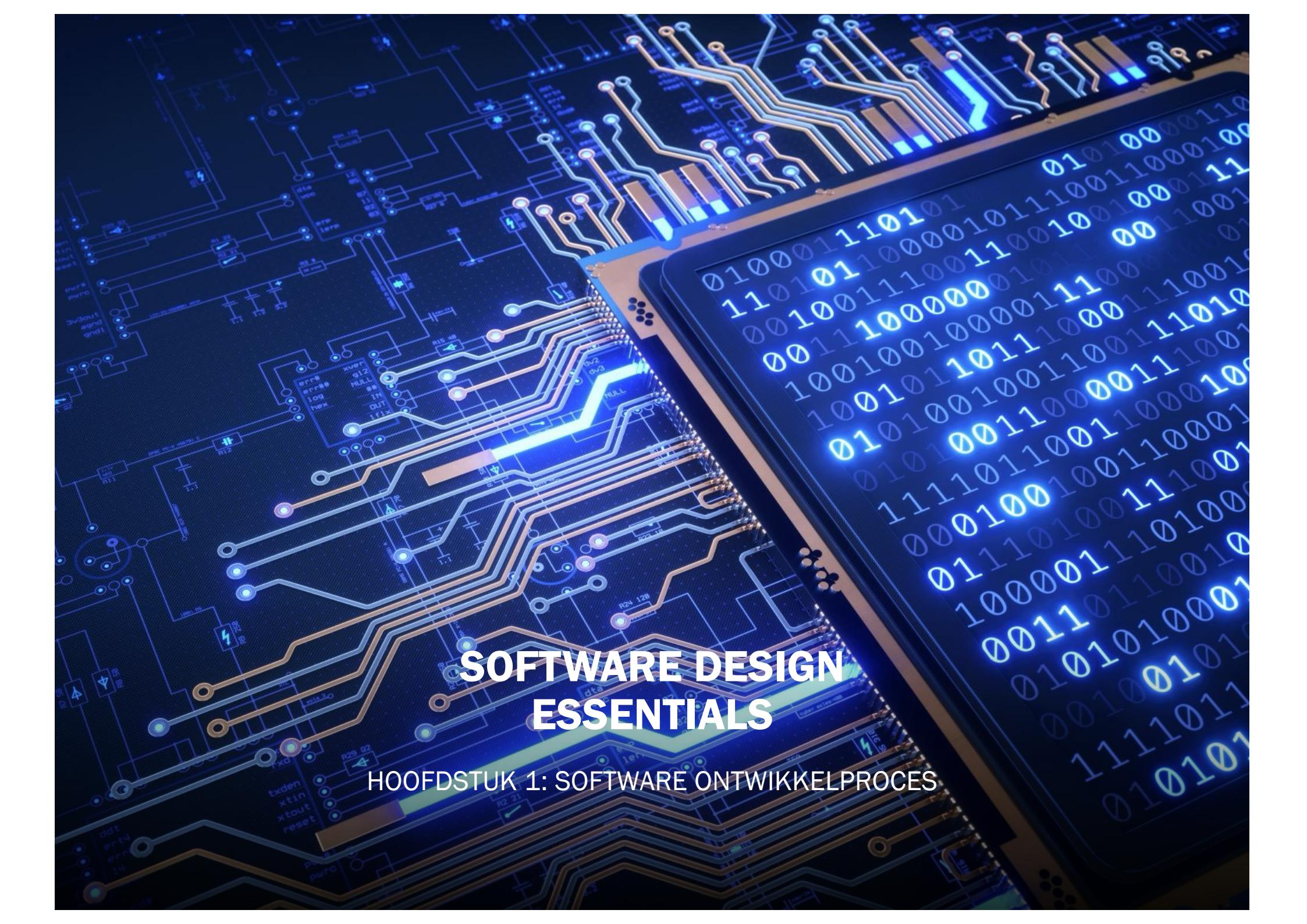
 UML: class diagrams

 UML: communication diagrams

 UML: activity diagrams en state machine diagrams

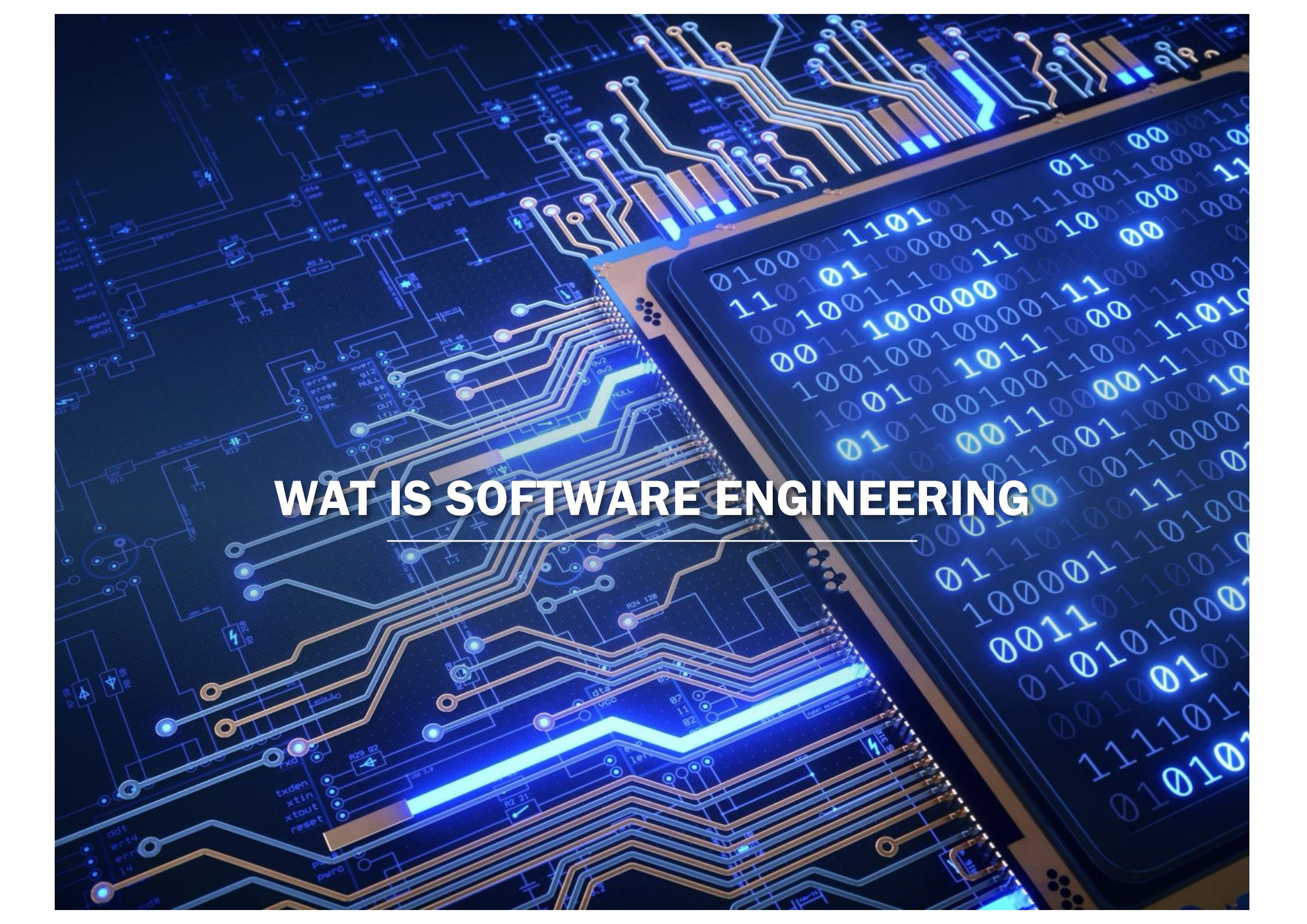
 Software testing

 Application lifecycle management



SOFTWARE DESIGN ESSENTIALS

HOOFDSTUK 1: SOFTWARE ONTWIKKELPROCES



WAT IS SOFTWARE ENGINEERING



SOFTWARE ONTWIKKELING = TEAM WORK

Software wordt meestal in teamverband ontwikkeld.

Het ontwikkelproces is complex en vereist diverse vaardigheden en expertise.

Elk teamlid heeft een specifieke rol en verantwoordelijkheid:

- **Klant:** Verwacht een functionerend product dat aan zijn behoeften voldoet.
 - **Projectleider:** Draagt de eindverantwoordelijkheid voor het hele project.
 - **Analist:** Vertaalt de wensen van de klant naar een technische analyse.
 - **Ontwikkelaars:** Implementeren de software op basis van de analyse.
 - **Testers:** Detecteren fouten en tekortkomingen in de software.
- ❗ Ondanks goede samenwerking, gaat softwareontwikkeling vaak mis...



**ONDANKS TEAMWORK LOOPT HET VAAK FOUT
MET HET ONTWIKKelen VAN SOFTWARE...**



How the customer explained it



How the Project Leader
understood it



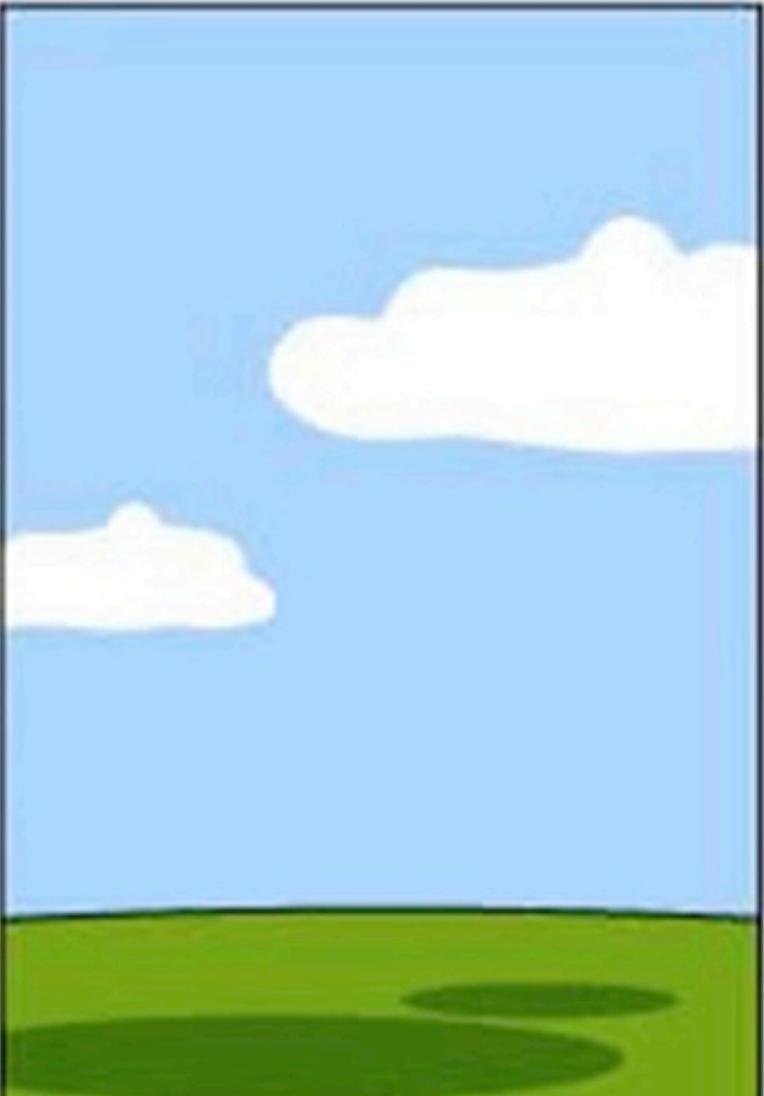
How the Analyst designed it



How the Programmer wrote it



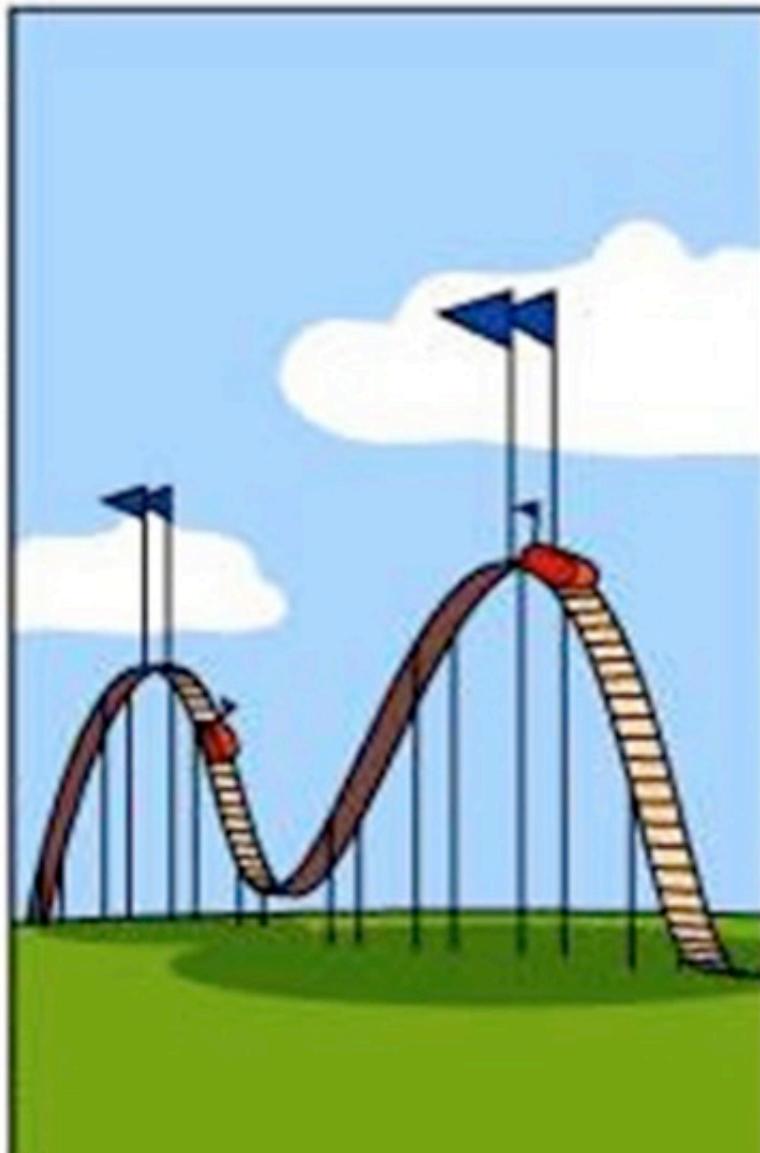
How the Business Consultant
described it



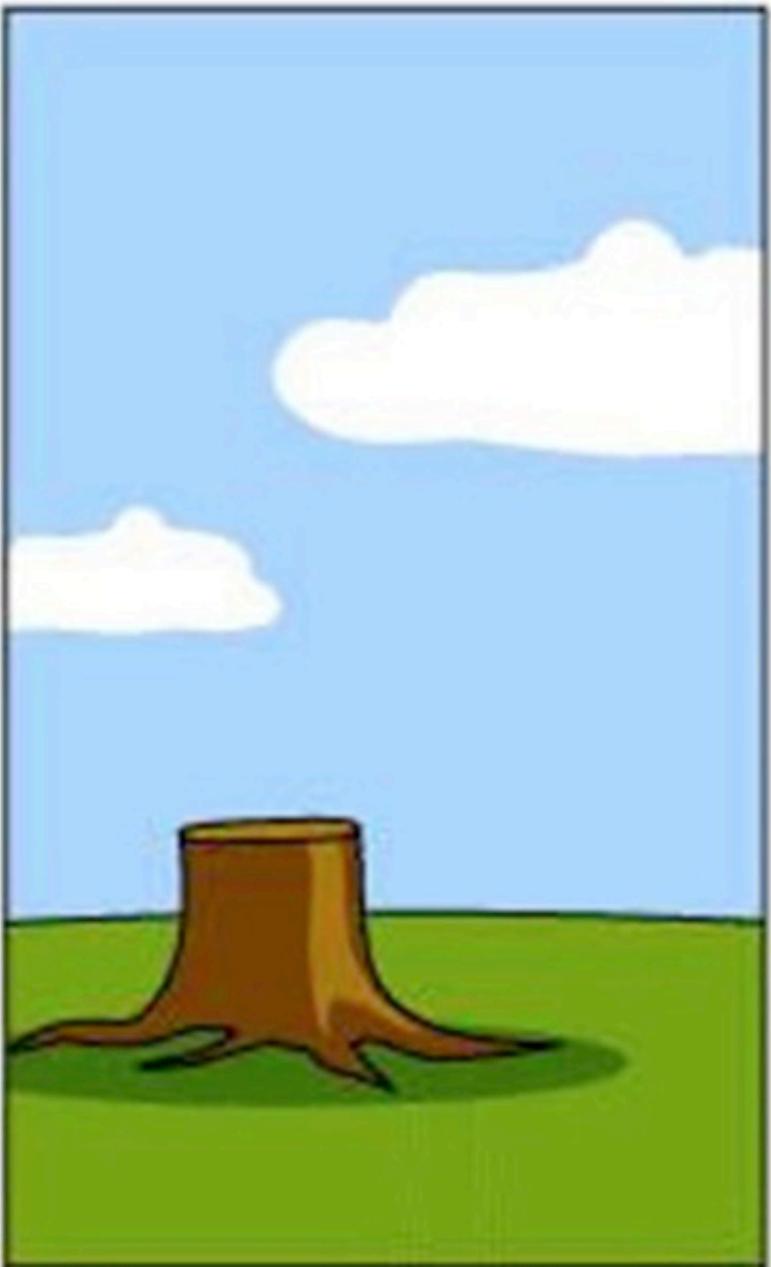
How the project was
documented



What operations installed



How the customer was billed



How it was supported



What the customer really
needed

WAT KAN ER MISLOPEN

Verlies van focus

Chaotische planning

Kwaliteitsproblemen

Verhoogd risico op miscommunicatie

Gebrek aan transparantie

Verhoogde kosten

Gebrek aan flexibiliteit

WAT KAN ER MISLOPEN

- **Verlies van focus:** Zonder een duidelijke structuur raken teams snel afgeleid door taken die niet bijdragen aan de projectdoelstellingen. Dit leidt tot tijd- en middelenverspilling.
- **Chaotische planning:** Een gebrek aan gestructureerde planning kan leiden tot gemiste mijlpalen, overschreden deadlines en een project dat uit de hand loopt.
- **Kwaliteitsproblemen:** Zonder kwaliteitsborging worden testen, code reviews en controles vaak overgeslagen, wat resulteert in software vol bugs en fouten.
- **Miscommunicatie:** Een onsaamhangende aanpak kan leiden tot versnipperde communicatie, waardoor belangrijke informatie verloren gaat of verkeerd wordt begrepen. Dit veroorzaakt verwarring en misverstanden binnen het team.

WAT KAN ER MISLOPEN

- **Onvoldoende transparantie:** Zonder een gestructureerde aanpak is het lastig om de voortgang van het project helder in beeld te krijgen. Dit kan leiden tot onduidelijkheid over afgeronde taken, openstaande werkzaamheden en obstakels die nog overwonnen moeten worden.
- **Hoge kosten:** Een gebrek aan structuur zorgt voor inefficiënte processen, onverwachte problemen en vertragingen, wat resulteert in extra kosten en verspilling van middelen.
- **Beperkte flexibiliteit:** Zonder duidelijke processen is het moeilijk om snel in te spelen op veranderende vereisten of prioriteiten, waardoor het team minder wendbaar wordt.

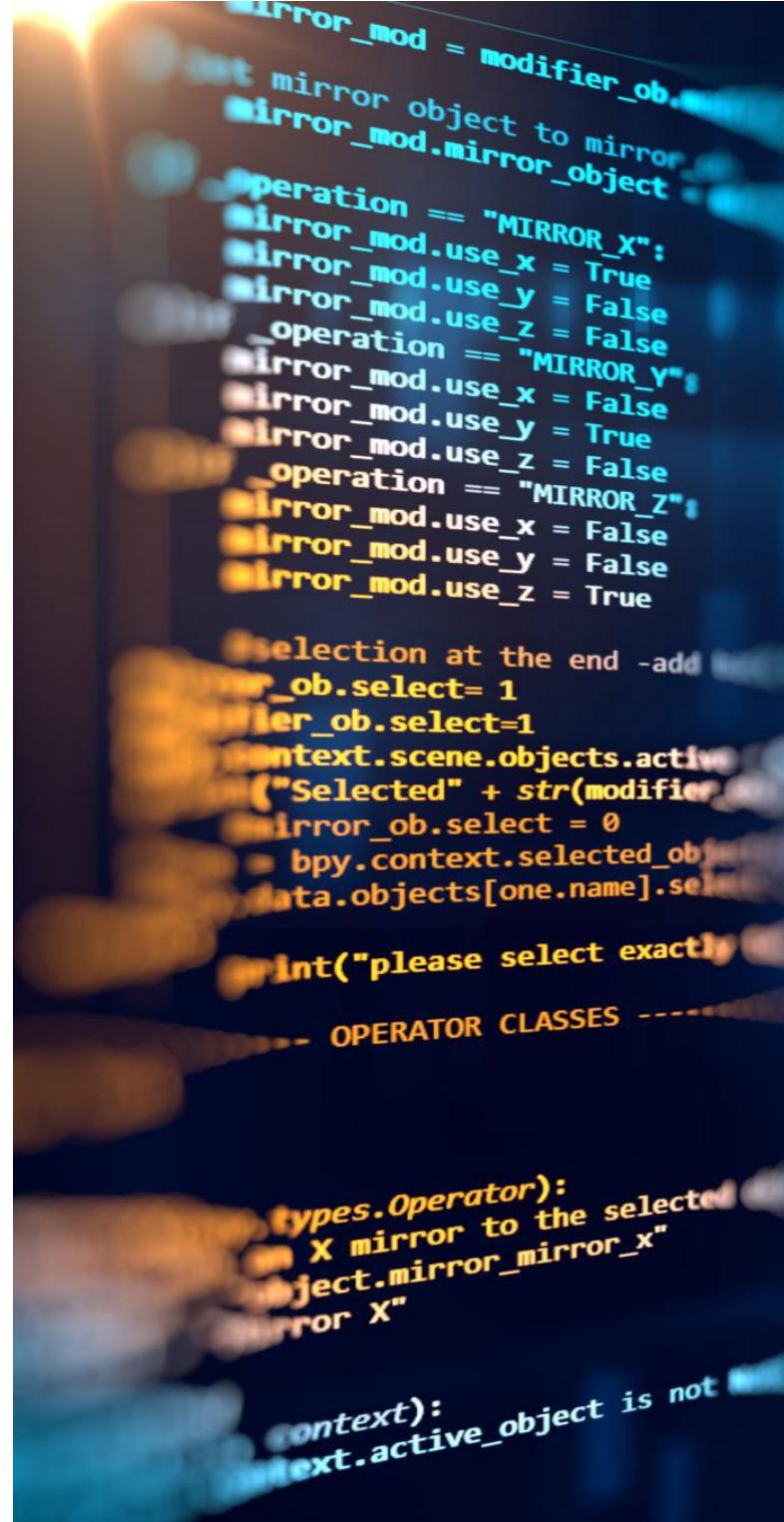
SOFTWARE ENGINEERING

- Softwareprojecten vereisen een gestructureerde aanpak om succesvol te zijn.
- **Software engineering** omvat het volledige proces van bedenken, ontwerpen, bouwen en testen van software. Het doel is om software te ontwikkelen die **betrouwbaar, efficiënt en gebruiksvriendelijk** is, zodat deze optimaal voldoet aan de verwachtingen en behoeften van de gebruiker.



DE EVOLUTIE VAN SOFTWAREONTWIKKELING

- Assembler
 - Programmeurs werken direct met machine-instructies.
 - Zeer laag niveau, dicht bij de hardware, moeilijk onderhoudbaar.
- C
 - Hogere abstractie dan assembler, maar nog steeds dicht bij de hardware.
 - Programmeurs moeten zelf veel logica en geheugenbeheer regelen.
- C++, Java, C#
 - Introductie van objectgeoriënteerd programmeren (OOP).
 - Code wordt modulair en herbruikbaar door het opdelen van functionaliteit in klassen en objecten.
- Software Engineering
 - Gestructureerde aanpak met methodologieën en best practices.
 - Gericht op schaalbaarheid, onderhoudbaarheid en samenwerking in teams.



```
mirror_mod = modifier_obj
if mirror_mod.mirror_object:
    mirror_object_to_mirror = mirror_mod.mirror_object
    if operation == "MIRROR_X":
        mirror_mod.use_x = True
        mirror_mod.use_y = False
        mirror_mod.use_z = False
    elif operation == "MIRROR_Y":
        mirror_mod.use_x = False
        mirror_mod.use_y = True
        mirror_mod.use_z = False
    elif operation == "MIRROR_Z":
        mirror_mod.use_x = False
        mirror_mod.use_y = False
        mirror_mod.use_z = True

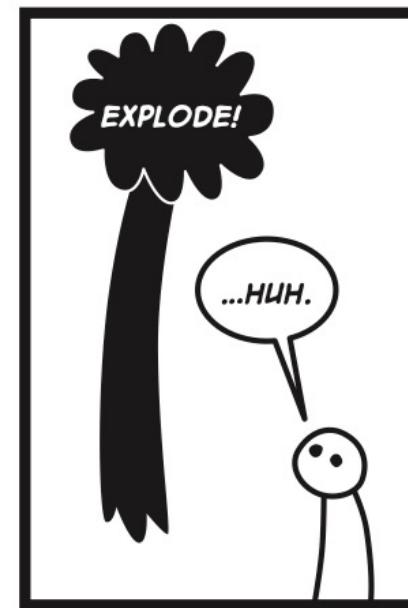
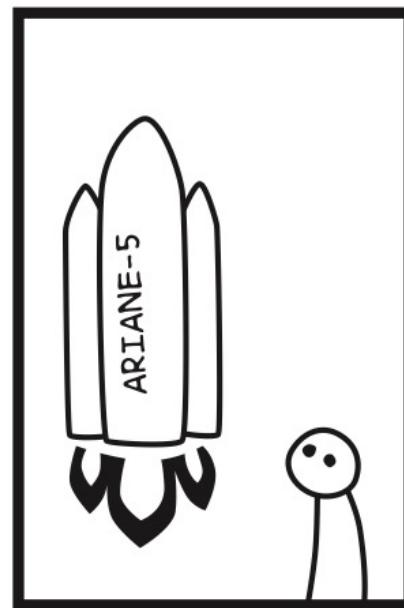
selection_at_the_end = add_selection
ob.select= 1
ier_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier))
rror_ob.select = 0
 bpy.context.selected_objects
data.objects[one.name].sele
int("please select exact")
 - OPERATOR CLASSES ----
types.Operator):
 X mirror to the selected ob
ject.mirror_mirror_x"
rror X"
context):
ext.active_object is not
```

ZONDER SOFTWARE ENGINEERING

- **Te late oplevering** – Projecten duren langer dan gepland.
- **Budgetoverschrijding** – Kosten lopen uit de hand door slechte planning of onverwachte complicaties.
- **Bugs en fouten** – Slechte codekwaliteit leidt tot fouten in de software.
- **Niet voldoen aan gebruikers- en bedrijfsbehoeften** – Software sluit niet aan op verwachtingen.
- **Integratieproblemen** – Modules werken niet goed samen.
- **Slechte prestaties onder hoge belasting** – Software crasht of reageert traag bij veel gebruikers.
- **Fatale fouten** – Kritieke softwarefouten met ernstige gevolgen.
- Voorbeelden van desastreuze softwarefouten:
 - Therac-25 – Stralingsapparaat dat patiënten een dodelijke dosis straling gaf.
 - Ariane 5 – Raketexplosie door een softwarefout, verlies: \$7 miljard.

FATALE FOUTEN: ARIANE 5

■ https://www.youtube.com/watch?v=gp_D8r-2hwk





WAAROM IS SOFTWARE ENGINEERING NODIG?

- **Complexiteit van Software**
 - Moderne software is groot en bevat veel componenten.
 - Code moet onderhoudbaar en uitbreidbaar zijn.
- **Communicatie en samenwerking**
 - Teams werken samen, vaak op verschillende locaties.
 - Duidelijke processen voorkomen misverstanden.
- **Kostenbeheersing**
 - Slechte planning en bugs kunnen leiden tot hoge kosten.
 - Efficiënte methodologieën helpen budgetoverschrijdingen te voorkomen.



WAAROM IS SOFTWARE ENGINEERING NODIG?

- **Betrouwbaarheid en kwaliteit**
 - Software moet stabiel en foutloos functioneren.
 - Testen en kwaliteitscontroles zijn essentieel.
- **Snelle technologische evolutie**
 - Software moet flexibel zijn voor toekomstige aanpassingen.
 - Nieuwe technologieën vragen om continue optimalisatie.

DE LEVENSCYCLUS VAN SOFTWARE



DE LEVENSCYCLUS VAN SOFTWARE

1. Beeldvorming (vereisten)
2. Specificatie
3. Ontwerp (representatie)
4. Implementatie
5. Integratie
6. Onderhoud
7. Buiten werking stellen

Opmerkingen

- Geen testfase!
 - Gebeurt gedurende het volledige project
- Geen documentatiefase!
 - Gebeurt gedurende het volledige project



BEELDVORMING

Identificatie van behoeften en doelen

Analyse van de huidige situatie

Vaststellen van scope

Risicoanalyse

Projectplan ontwikkelen

BEELDVORMING

- **Behoeften en doelen bepalen**
 - Het team brengt de behoeften van stakeholders in kaart en stelt duidelijke projectdoelen vast.
 - Dit gebeurt via gesprekken met klanten, gebruikers en andere belanghebbenden om hun verwachtingen en vereisten te begrijpen.
- **Analyse van de huidige situatie**
 - Onderzoek naar bestaande systemen, processen en technologieën om de noodzakelijke verbeteringen te identificeren.
 - Help bij het bepalen van wat behouden kan blijven en welke veranderingen nodig zijn voor succes.
- **Afbakening van de scope**
 - Definiëren welke functionaliteiten en kenmerken binnen het project vallen.
 - Identificeren van beperkingen zoals budget, tijd en middelen om realistische verwachtingen te stellen.
- **Risicoanalyse**
 - Het team onderzoekt mogelijke risico's die het project kunnen beïnvloeden.
 - Denk aan technologische uitdagingen, resourcebeperkingen en veranderende vereisten.
 - Strategieën worden ontwikkeld om deze risico's te beheersen en te minimaliseren.
- **Ontwikkelen van een projectplan**
 - Opstellen van een gedetailleerd plan met projectdoelstellingen, mijlpalen, resourceverdeling en tijdslijnen.
 - Dit biedt een gestructureerde leidraad voor de verdere ontwikkeling en uitvoering van het project.

SPECIFICATIE

Verzamelen van vereisten

- **functionele vereisten** (wat de software moet doen)
- **niet-functionele vereisten** (hoe de software moet presteren, zoals prestaties, beveiliging, enz.).

Analyse van vereisten

Specificaties opstellen

Prototype ontwikkelen (optioneel)

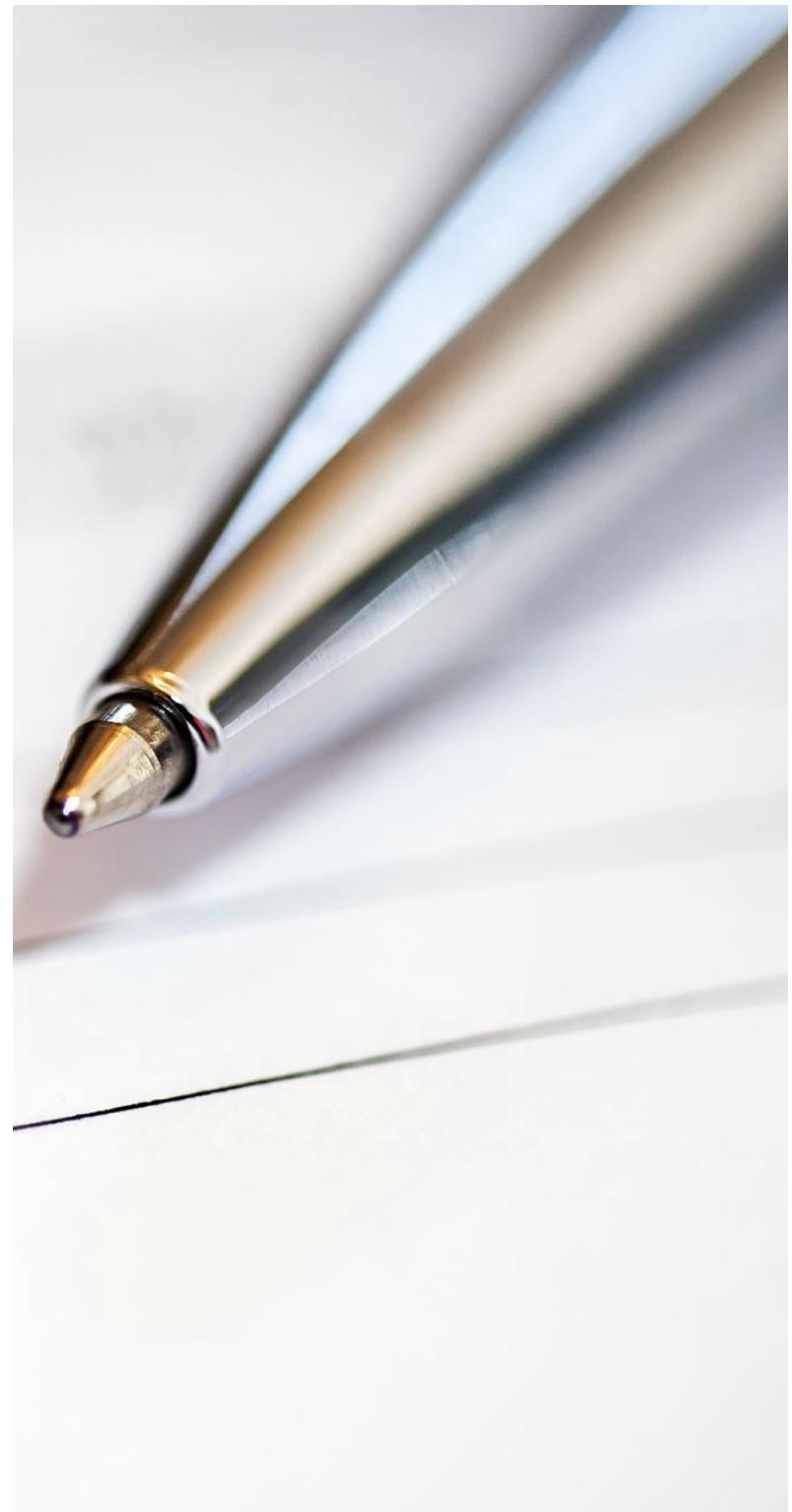
Validatie en goedkeuring

SPECIFICATIE

- **Vereisten verzamelen**
 - Het team verzamelt gedetailleerde informatie over de functionaliteiten en prestaties die de software moet leveren.
 - Dit omvat:
 - Functionele vereisten (wat de software moet doen).
 - Niet-functionele vereisten (hoe de software moet presteren, zoals snelheid, beveiliging en schaalbaarheid).
- **Analyse en validatie van vereisten**
 - De verzamelde vereisten worden gecontroleerd op volledigheid, consistentie en haalbaarheid.
 - Mogelijke tegenstrijdigheden of onduidelijkheden worden opgelost in overleg met belanghebbenden.
- **Opstellen van specificaties**
 - Op basis van gevalideerde vereisten worden twee documenten opgesteld:
 - Functionele specificatie: Beschrijft de gewenste functionaliteiten.
 - Technische specificatie: Geeft aan hoe de software ontworpen en geïmplementeerd wordt.
- **(Optioneel) Ontwikkelen van een prototype**
 - Een prototype kan worden gebouwd om het ontwerp en de werking te testen.
 - Dit helpt bij het vroegtijdig opsporen van problemen en geeft belanghebbenden een beter beeld van de verwachte software.
- **Validatie en Goedkeuring**
 - De specificaties worden beoordeeld en goedgekeurd door klanten, gebruikers en projectmanagers.
 - Dit zorgt ervoor dat alle betrokkenen akkoord gaan met de vereisten voordat de ontwikkeling start.

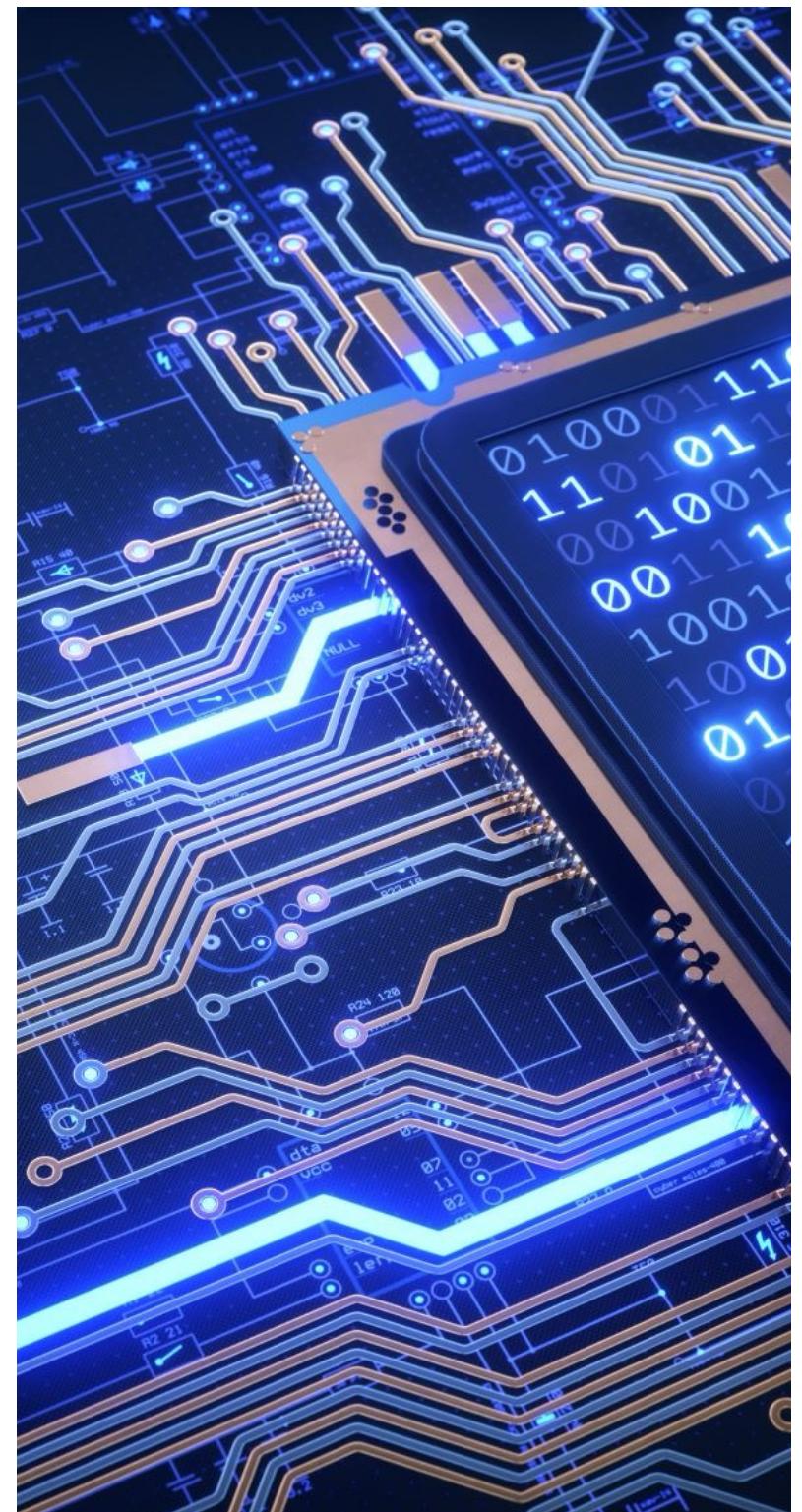
SPECIFICATIE

- Overeenkomst tussen klant en ontwikkelteam
 - De specificatie fungeert als een formele afspraak tussen de klant en de ontwikkelaars.
 - Hierin worden alle functionele en niet-functionele vereisten van de software vastgelegd.
- Geen dubbelzinnigheden
 - De specificatie moet ondubbelzinnig en concreet zijn om interpretatiefouten te voorkomen.
 - Dit voorkomt misverstanden en conflicten tijdens de ontwikkeling.
- Volledigheid en nauwkeurigheid
 - Zowel de functionele vereisten (wat de software moet doen) als de niet-functionele vereisten (zoals prestaties, beveiliging en schaalbaarheid) worden gedetailleerd beschreven.
- Juridische ondersteuning indien nodig
 - Bij complexe of bedrijfskritische projecten worden specificaties soms juridisch vastgelegd.
 - Juristen kunnen helpen om contractuele verplichtingen, intellectueel eigendom en aansprakelijkheden te verduidelijken.



ONTWERP

- Tijdens de **ontwerp fase** worden de specificaties en vereisten van de software omgezet in een **gedetailleerd technisch ontwerp**. Dit ontwerp beschrijft de **structuur, architectuur en functionaliteit** van de software om ervoor te zorgen dat de ontwikkeling gestructureerd en efficiënt verloopt.



BELANGRIJKE ASPECTEN VAN HET ONTWERP

- Architectuurontwerp
 - Bepalen van de algemene structuur van de software.
 - Identificeren van componenten, subsystemen en hun onderlinge relaties.
 - Creëren van een robuuste, schaalbare en flexibele architectuur die voldoet aan de projectvereisten.
- Databaseontwerp
 - Modelleren van de gegevensstructuur.
 - Identificeren van entiteiten, relaties en afhankelijkheden.
 - Ontwerpen van tabellen en database-objecten voor efficiënte opslag en gegevensbeheer.
- Gebruikersinterface (UI) ontwerp
 - Ontwerpen van de visuele en interactieve elementen van de software.
 - Maken van wireframes, mock-ups en prototypes om lay-out, navigatie en functionaliteit te valideren.
 - Gericht op een intuïtieve en gebruiksvriendelijke ervaring.
- Beveiligingsontwerp
 - Identificeren van mogelijke beveiligingsrisico's en bedreigingen.
 - Ontwerpen van maatregelen tegen ongeautoriseerde toegang, gegevensdiefstal en kwetsbaarheden.
 - Integratie van encryptie, authenticatie en andere beveiligingstechnieken.

IMPLEMENTATIE

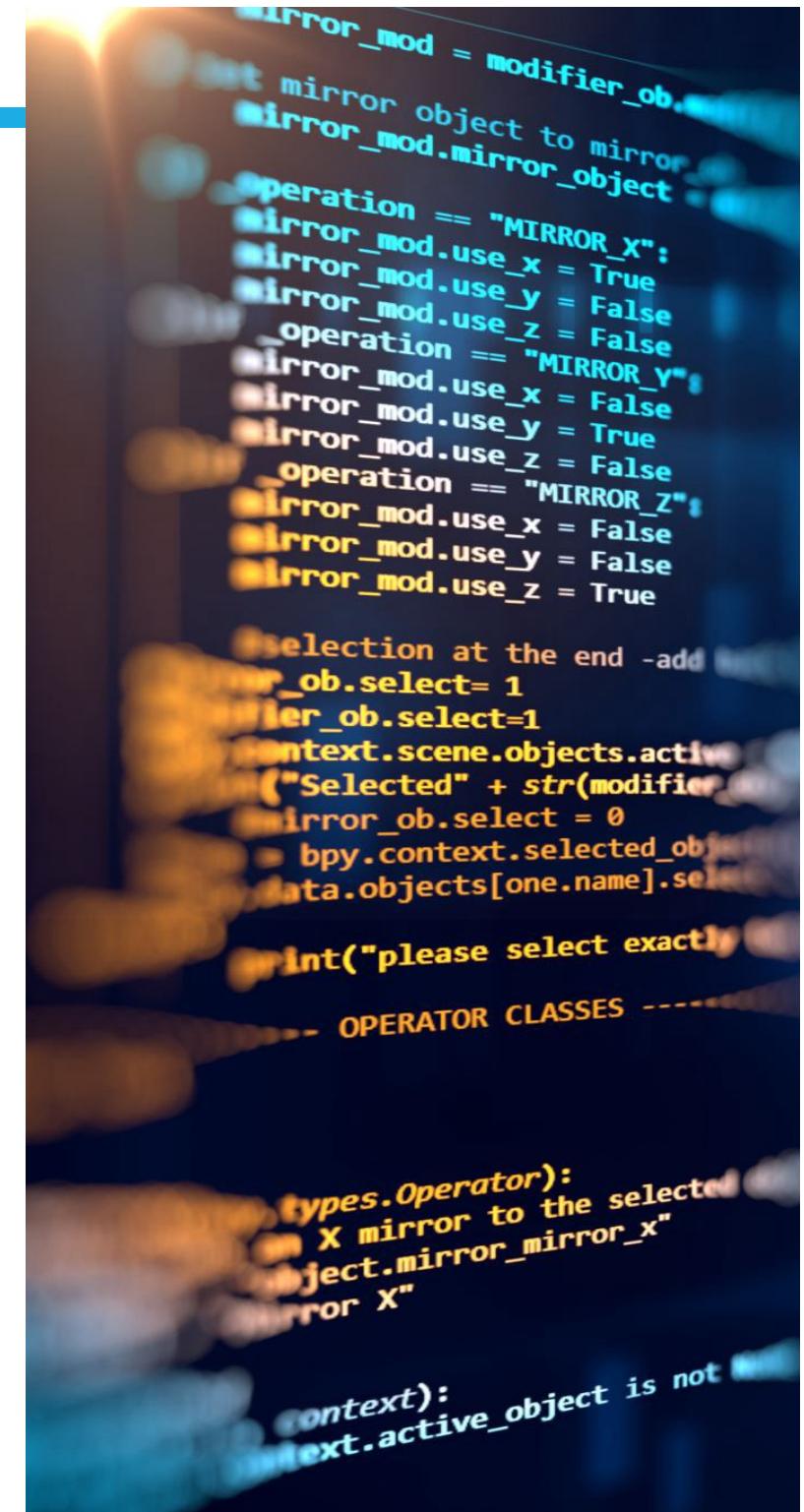
Codering

Componentintegratie

Testen (unit testing)

Debugging

Documentatie



```
mirror_mod = modifier_obj
# set mirror object to mirror
mirror_mod.mirror_object = mirror
if operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

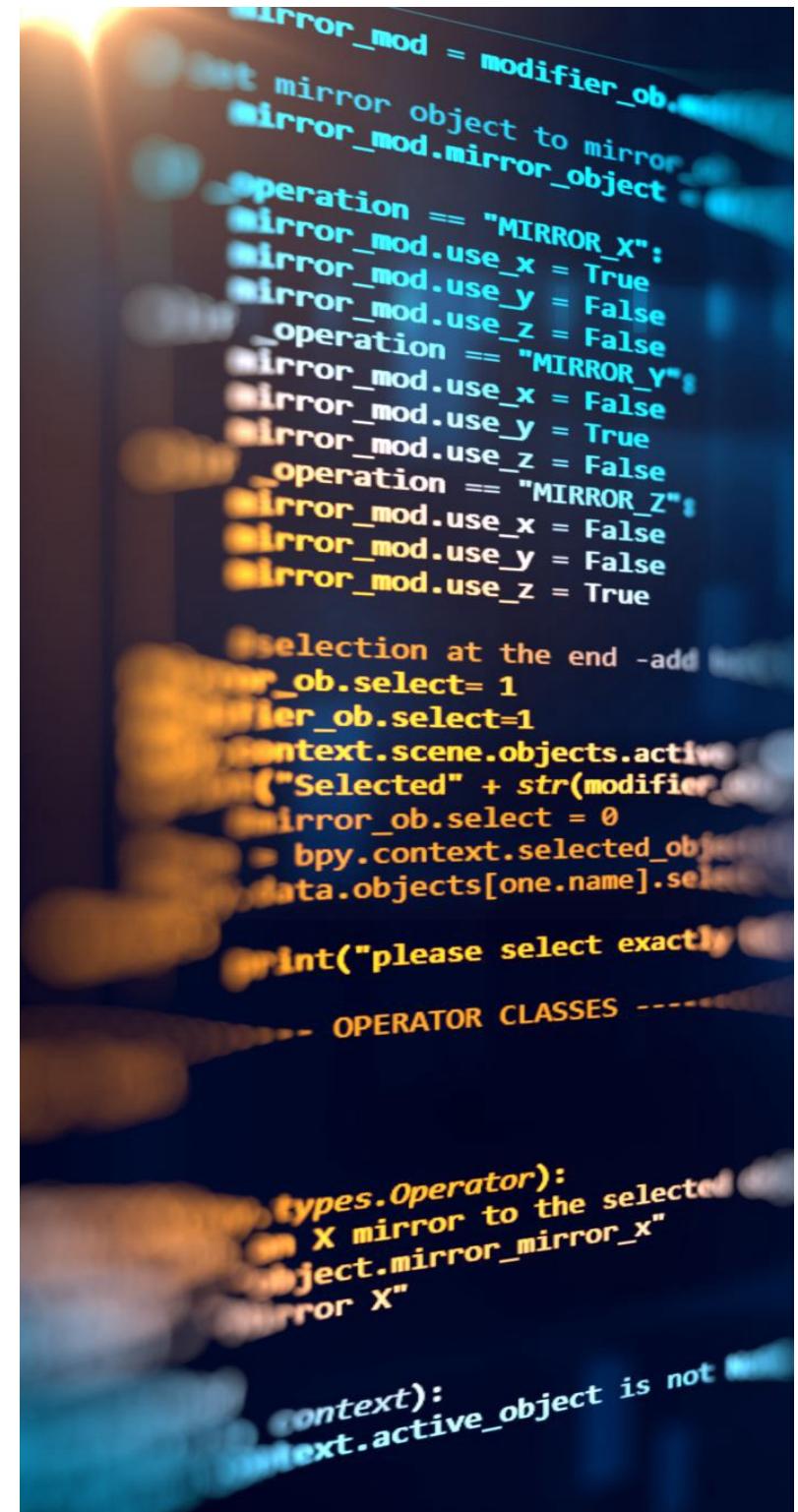
selection at the end -add
ob.select= 1
ier_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier))
rror_ob.select = 0
bpy.context.selected_obi
ata.objects[one.name].sele
int("please select exact
-- OPERATOR CLASSES ----

types.Operator):
    X mirror to the selected o
ject.mirror_mirror_x"
rror X"

context):
    context.active_object is not
```

IMPLEMENTATIE

- Tijdens de implementatiefase wordt de software daadwerkelijk gebouwd op basis van het ontwerp uit de eerdere fasen. In deze fase wordt de programmeercode geschreven en geïmplementeerd om de functionaliteiten van de software tot leven te brengen.



```
mirror_mod = modifier_obj
# set mirror object to mirror
mirror_mod.mirror_object = ob
operation = "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True

selection at the end -add
ob.select= 1
ier_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier))
rror_ob.select = 0
 bpy.context.selected_objects
data.objects[one.name].sele
int("please select exact
- OPERATOR CLASSES ----

types.Operator):
 X mirror to the selected o
ject.mirror_mirror_x"
rror X"

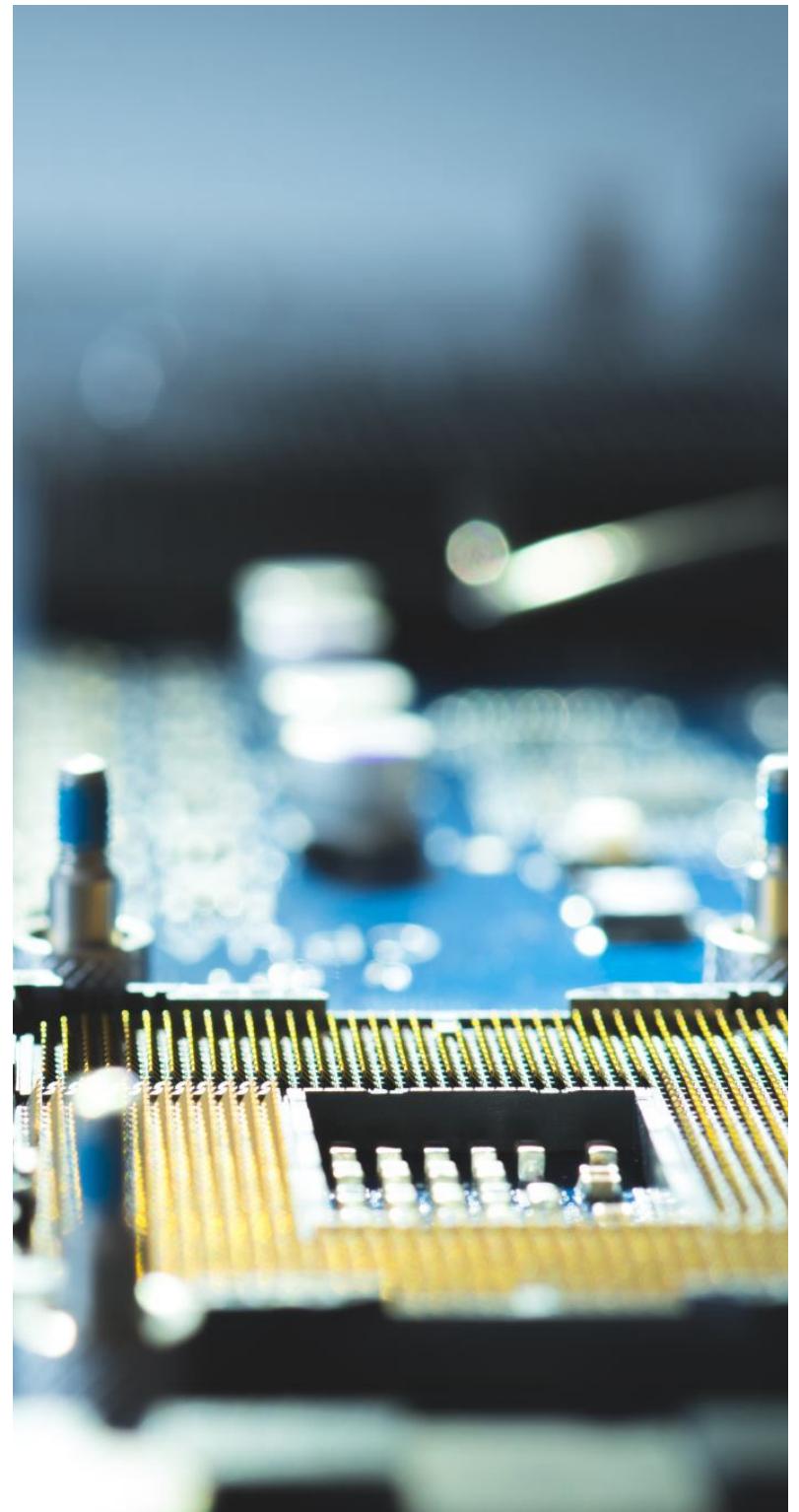
context):
text.active_object is not
```

IMPLEMENTATIE

- **Codering**
 - Het ontwikkelteam vertaalt het ontwerp naar werkende programmacode.
 - Er wordt geprogrammeerd in talen zoals Java, Python, C++, afhankelijk van de technologieën en projectvereisten.
- **Componentintegratie**
 - Indien de software uit meerdere modules bestaat, worden deze samengevoegd.
 - Interfaces en communicatiekanalen tussen de componenten worden opgezet om een samenhangend systeem te vormen.
- **Testen**
 - Tijdens de implementatie worden automatische en handmatige tests uitgevoerd om de correcte werking van individuele modules te garanderen.
 - Dit helpt bij het vroegtijdig identificeren van fouten en inconsistenties.
- **Debugging**
 - Eventuele programmefouten (bugs) worden opgespoord en gecorrigeerd.
 - Dit omvat code-analyse, foutopsporing en aanpassingen om de stabilititeit te garanderen.
- **Documentatie**
 - Tijdens de ontwikkeling wordt documentatie bijgehouden over de code:
 - Functies, variabelen, methoden en architecturale beslissingen worden vastgelegd.
 - Dit helpt bij onderhoud, samenwerking en toekomstige uitbreidingen van de software.

INTEGRATIE

- Tijdens integratie worden de afzonderlijke componenten of modules van de software samengevoegd tot een samenhangend en functionerend geheel.
- **Interne Integratie**
 - Zorgt ervoor dat alle softwarecomponenten correct samenwerken als één geheel.
- **Externe integratie**
 - Richt zich op de koppeling van de software met externe systemen en diensten.
 - Voorbeelden: betalingsverwerking, authenticatiesystemen, API's en cloudservices.
 - Doel: Garanderen dat de software naadloos functioneert in een bredere IT-infrastructuur.
- **Systeemintegratietesten**
 - Controle of alle componenten correct samenwerken en voldoen aan de specificaties.
 - Identificeren en oplossen van compatibiliteitsproblemen tussen verschillende modules.
 - Help bij het vroegtijdig opsporen van integratiefouten, zodat deze niet pas in de gebruikerstestfase aan het licht komen.





ONDERHOUD

Correctief onderhoud

Adaptief onderhoud

Patchbeheer

Gebruikersondersteuning

ONDERHOUD

- De onderhoudsfase begint zodra de software is geïmplementeerd en in gebruik is genomen. In deze fase wordt de software continu bijgewerkt en aangepast om functionaliteit, veiligheid en prestaties te garanderen.

ONDERHOUD

Correctief onderhoud

- Oplossen van bugs en fouten die na implementatie worden ontdekt.
- Gebruikers kunnen problemen rapporteren die variëren van kleine storingen tot kritieke defecten.
- Het ontwikkelteam analyseert en implementeert oplossingen om de software optimaal te laten functioneren.

Adaptief onderhoud

- Aanpassen van de software aan veranderende omgevingen, zoals:
- Nieuwe hardware of besturingssystemen.
- Updates van externe API's of interfaces.
- Zorgt ervoor dat de software compatibel en toekomstbestendig blijft.

Patchbeheer en beveiligingsupdates

- Regelmatig uitrollen van updates en patches om kwetsbaarheden te verhelpen.
- Voorkomen van beveiligingslekken en compliance-risico's.
- Het ontwikkelteam monitort dreigingen en implementeert beveiligingsmaatregelen indien nodig.

Gebruikersondersteuning

- Beantwoorden van vragen en bieden van technische hulp aan gebruikers.
- Trainen van gebruikers om de software efficiënt te gebruiken.
- Up-to-date documentatie verschaffen voor ondersteuning en probleemoplossing.

BUITEN WERKING STELLEN

- Hier zijn enkele mogelijke redenen voor het buiten werking stellen van software:
 - 1. Veroudering**
 - 2. Vervanging**
 - 3. Kostenbesparing**
 - 4. Veiligheids- of compliance-overwegingen**
 - 5. Veranderingen in bedrijfsbehoeften**

BUITEN WERKING STELLEN

- Het buiten werking stellen van software betekent dat een applicatie, systeem of onderdeel niet langer wordt gebruikt. Dit proces kan nodig zijn om veroudering, kosten of veiligheidsrisico's te beheersen en gebeurt meestal in een gecontroleerd en gepland traject.

REDENEN VOOR HET STOPZETTEN VAN SOFTWARE

- **Veroudering**
 - De software voldoet niet meer aan moderne technologische standaarden.
 - In plaats van tijd en middelen te besteden aan updates, wordt gekozen voor een nieuwe oplossing.
- **Vervanging**
 - Een nieuwe versie of een alternatieve softwareoplossing vervangt de oude.
 - De oude software wordt buiten werking gesteld om een soepele overgang mogelijk te maken.
- **Kostenbesparing**
 - Onderhoud en operationele kosten wegen niet langer op tegen de voordelen.
 - Het stopzetten van de software bespaart middelen en maakt budget vrij voor andere toepassingen.
- **Veiligheids- of compliance-overwegingen**
 - De software voldoet niet meer aan beveiligingsnormen of wettelijke vereisten.
 - Dit kan leiden tot beveiligingsrisico's, juridische problemen of datalekken.
 - Het stopzetten voorkomt mogelijke kwetsbaarheden en non-compliance.
- **Veranderingen in bedrijfsbehoeften**
 - De software ondersteunt de huidige bedrijfsprocessen of strategieën niet meer.
 - Wanneer de software niet langer waarde toevoegt, wordt deze uitgefaseerd.

SOFTWARE ONTWIKKELINGSPROCES

WATERVALMODEL

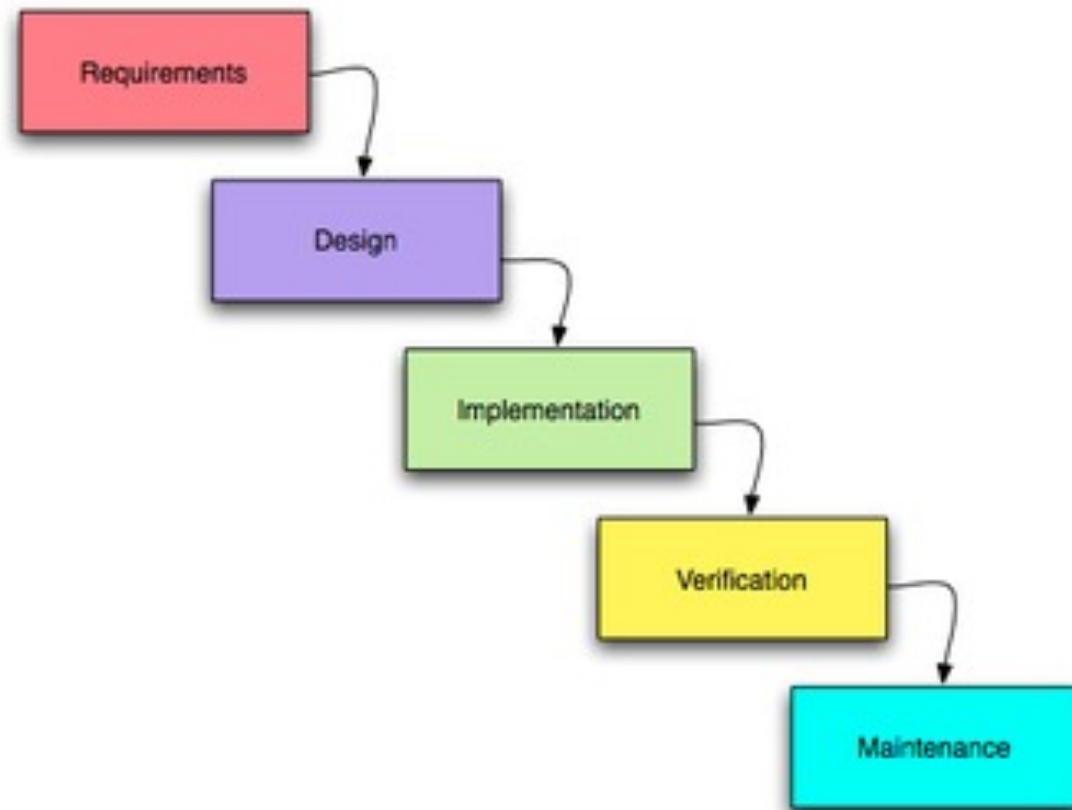
Volgende stap kan pas als vorige stap afgewerkt

Idee

- Veel tijd spenderen aan initiële fasen, bespaart veel geld!
- Fout in later stadium (bv. implementatiefase) zorgt echter voor veel problemen

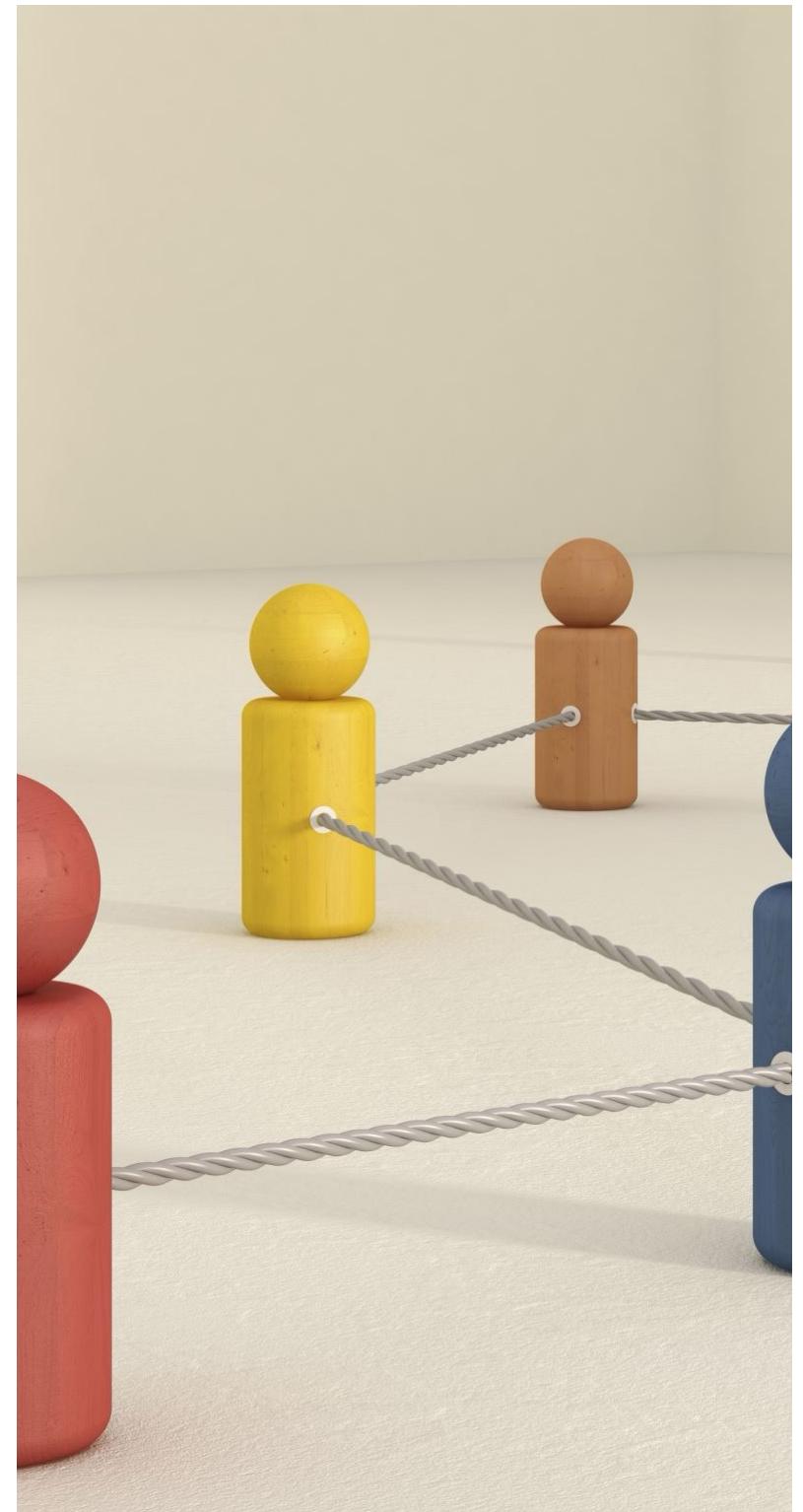
Deze situatie is niet realistisch

Te weinig tussentijds contact met klant mogelijk



WATERVALMODEL

- Watervalmodel is een voorbeeld van een sequentieel model. In dit model is de softwareontwikkelingsactiviteit verdeeld in verschillende fasen en elke fase bestaat uit een reeks taken en heeft verschillende doelstellingen.
- De uitvoer van de ene fase wordt de invoer van de volgende fase. Het is verplicht dat een fase is afgerond voordat de volgende fase start.
- Het was het eerste model dat veel werd gebruikt in de software-industrie.



NADELEN WATERVAL

- Geen tussentijdse feedback van gebruikers.
- Er wordt pas getest nadat de applicatie gebouwd is.
- Het is moeilijker, en daardoor ook duurder en tijdrovender, om fouten of bugs te corrigeren.
- Het proces is behoorlijk rigide en biedt weinig ruimte voor verandering.

AGILE

- Agile projectmanagement is een projectfilosofie of raamwerk dat een iteratieve benadering hanteert voor de voltooiing van een project.
- Software ontwikkelen via de Agile methode is compleet anders dan de traditionele Waterval methode.
- Waar je bij Waterval eerst alle eisen en wensen in kaart brengt, voordat er ook maar iets wordt ontworpen, ga je bij Agile direct aan de slag. Gedurende de ontwikkeling van de software worden de eisen en wensen steeds duidelijker.

The Agile Manifesto – a statement of values

Individuals and interactions

over

Process and tools

Working software

over

Comprehensive documentation

Customer collaboration

over

Contract negotiation

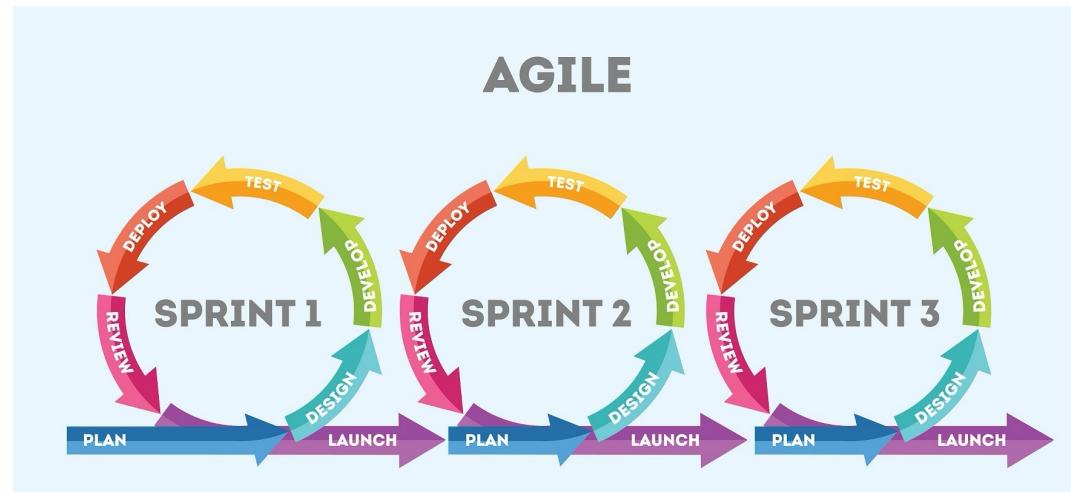
Responding to change

over

Following a plan

AGILE

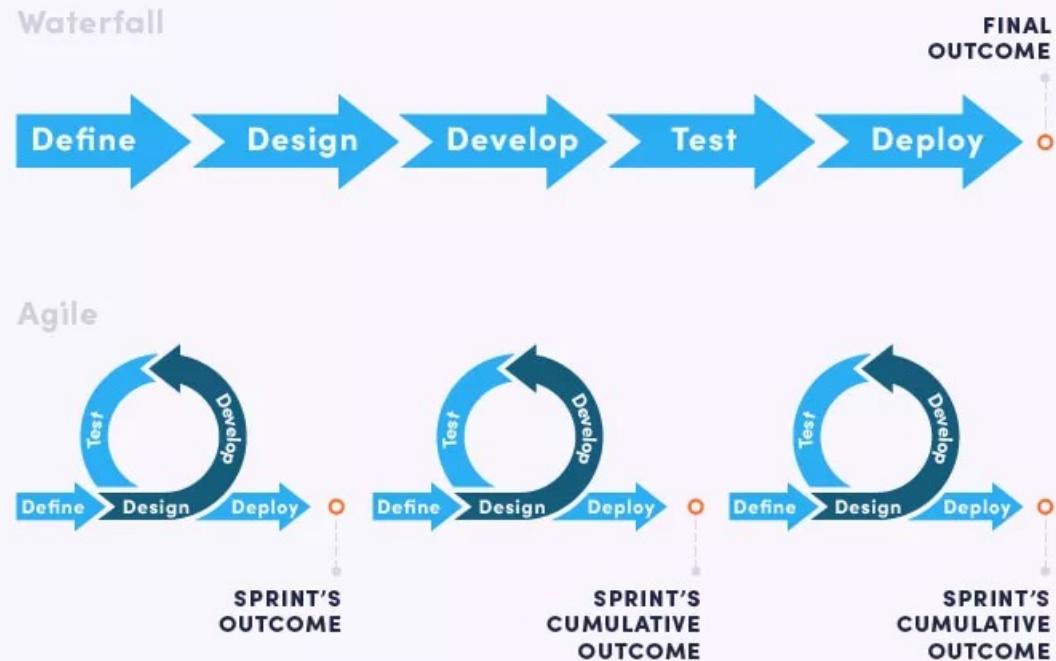
- De software wordt ontwikkeld in korte overzichtelijke timeboxes, ofwel iteraties.
- Na iedere iteratie/ontwikkelperiode, van bijvoorbeeld twee weken, wordt er iets bruikbaars opgeleverd dat getoond, getest en beoordeeld wordt.
- Omdat tijdens iedere iteratie of sprint alle noodzakelijke stappen van planning, ontwerp, ontwikkelen en testen steeds opnieuw doorlopen worden, beperk je het risico op fouten of een afwijkend resultaat. Je kunt dus continu bijsturen tijdens het proces. Doordat de klant nauw bij de ontwikkeling betrokken wordt en de productontwikkeling van dichtbij bijwoont, ontstaan er weer nieuwe ideeën en wordt het eindresultaat steeds concreter.
- Bekende Agile frameworks zijn Scrum en Kanban.



AGILE VS WATERVALMETHODE

- De Watervalmethode is een sequentieel proces, zonder overlap tussen de verschillende fasen, terwijl de Agile methode een iteratief proces is, een soort cyclus die zich continu herhaald.

Agile vs. Waterfall Quick Comparison



VOORDELEN AGILE METHODEN



Veel contact met de klant en gebruikers, waardoor het team continu kan bijsturen en feedback verwerken om de kwaliteit van het product te verbeteren.



Het is een flexibele methode, omdat alle fases van planning, ontwerp, ontwikkelen en testen meerdere keren worden doorlopen.



Je kan snel inspelen op veranderingen in de markt of wensen van de klant.



Iedere iteratie levert een werkend product, een tastbaar resultaat op.



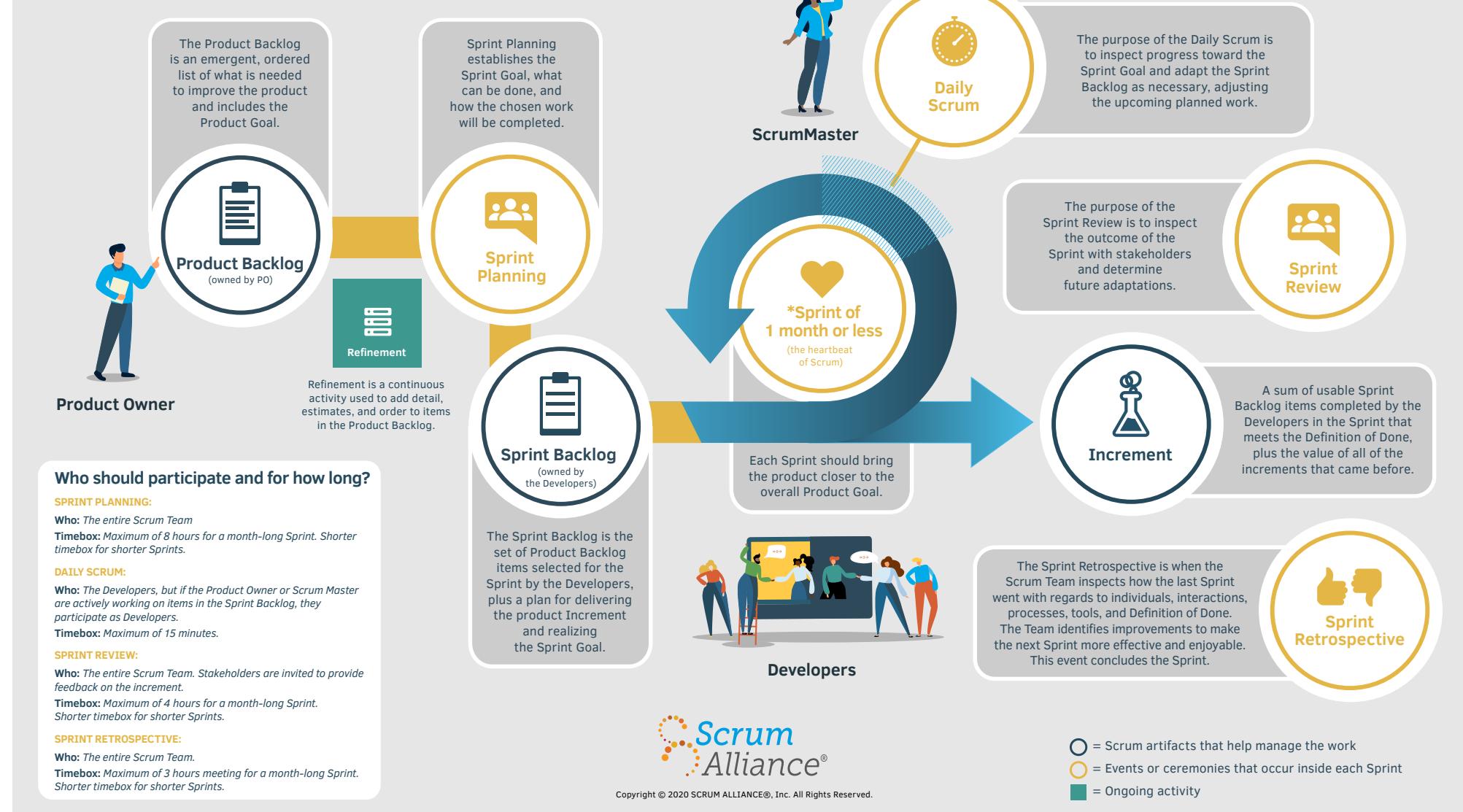
Geschikt voor (grote) complexe projecten.

NADELEN AGILE METHODEN

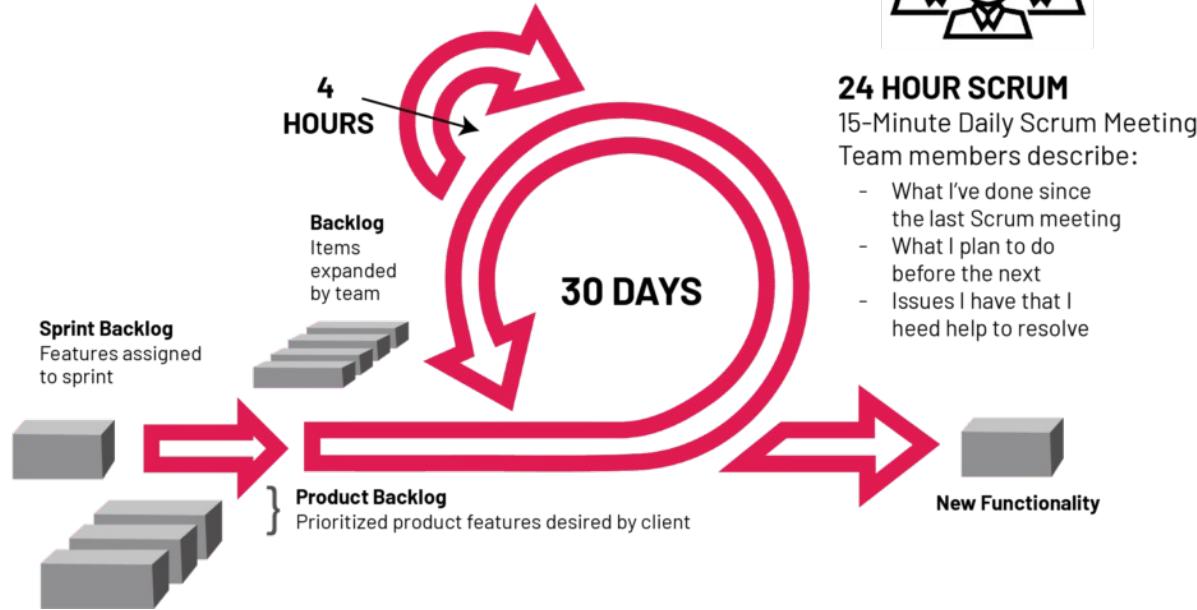
- Het is een intensief process en dit zowel voor het software ontwikkelteam als voor de klant.
- Als je niet gewend of getraind bent om agile te werken, is het best ingewikkeld.



The Scrum Framework At a Glance



SCRUM PROCESS



24 HOUR SCRUM

15-Minute Daily Scrum Meeting

Team members describe:

- What I've done since the last Scrum meeting
- What I plan to do before the next
- Issues I have that I need help to resolve

SCRUM PROCESS

- Een scrumproces onderscheidt zich van andere agile processen door specifieke concepten verdeeld in de drie categorieën rollen, artefacten en gebeurtenissen.

ROLES



Developers

Iedereen in het team die bijdraagt tot het eindresultaat



Product Owner

Bewaakt de visie voor het product



Scrum Master

Helps het team Scrum optimaal te gebruiken om het product te bouwen

ARTIFACTS

| Product Backlog | Sprint Backlog | Potentially Releasable Product Increment |
|--|---|---|
| <ul style="list-style-type: none">De product backlog is een geordende lijst van alles waarvan bekend is dat het nodig is in een product. Het evolueert voortdurend en is nooit compleet. | <ul style="list-style-type: none">De sprint backlog is een lijst van alles wat het team in een bepaalde sprint wil bereiken. Eenmaal gemaakt, kan niemand meer toevoegen aan de sprintachterstand, behalve het ontwikkelteam.Als het ontwikkelteam een item uit de sprint backlog moet laten vallen, moeten ze daarover onderhandelen met de Product Owner. Tijdens deze onderhandeling moet de Scrum Master samenwerken met het ontwikkelingsteam en de Product Owner om manieren te vinden om een kleinere increment van een item te maken in plaats van het helemaal te laten vallen. | <ul style="list-style-type: none">Aan het einde van elke sprint moet het team een productincrement voltooien dat mogelijk kan worden uitgebracht. |

EVENTS

| | |
|----------------------|---|
| The Sprint | De hartslag van Scrum. Elke sprint moet het product dichter bij het productdoel brengen en duurt een maand of minder. |
| Sprint Planning | Het hele Scrum-team stelt het sprintdoel vast, wat kan worden gedaan en hoe het gekozen werk zal worden voltooid. Maximaal 8 uur voor een Sprint van een maand. Kortere timebox voor kortere sprints. |
| Daily Scrum | De ontwikkelaars inspecteren de voortgang in de richting van het sprintdoel en passen de sprintachterstand waar nodig aan, waarbij ze het aanstaande geplande werk aanpassen. Kan Product Owner of Scrum Master omvatten als ze actief werken aan items in de sprint backlog. Maximaal 15 minuten per dag. |
| Sprint Review | Het hele Scrum-team inspecteert de uitkomst van de sprint met belanghebbenden en bepaalt toekomstige aanpassingen. Belanghebbenden worden uitgenodigd om feedback te geven over de verhoging. |
| Sprint Retrospective | Het Scrum-team inspecteert hoe de laatste sprint is verlopen met betrekking tot individuen, interacties, processen, tools en definition of done. Het team identificeert verbeteringen om de volgende sprint effectiever en plezieriger te maken. Dit is de conclusie van de sprint. Maximaal 3 uur voor een sprint van een maand, kortere timebox voor kortere sprints. |

SCRUM IN PRACTICE

- De Product Owner definieert een visie met behulp van informatie van stakeholders en gebruikers. Ze identificeren en definiëren stukjes waarde die kunnen worden geleverd om dichter bij het productdoel te komen. Voordat de ontwikkelaars aan waarde kunnen werken, moet de Product Owner de backlog ordenen zodat het team weet wat het belangrijkst is. Het team kan de producteigenaar helpen om verder te verfijnen wat er moet gebeuren, en de producteigenaar kan op de ontwikkelaars vertrouwen om hen te helpen de vereisten te begrijpen en afwegingsbeslissingen te nemen.
- Tijdens de sprintplanning nemen de ontwikkelaars een stuk uit de product backlog en beslissen hoe ze het zullen voltooien. Het team heeft een vast tijdsbestek, de sprint, om hun werk te voltooien. Ze ontmoeten elkaar tijdens de dagelijkse scrum om de voortgang naar het sprintdoel te inspecteren en plannen te maken voor de komende dag. Gaandeweg houdt de Scrum Master het team gefocust op het sprintdoel en kan hij het team als geheel helpen verbeteren.
- Aan het einde van de sprint moet het werk klaar zijn om door een gebruiker te worden gebruikt of aan een belanghebbende te worden getoond. Na elke sprint voert het team een sprintreview uit op het Increment en een retrospective op het proces. Vervolgens kiezen ze het volgende deel van de backlog en herhaalt de cyclus zich.