

An aerial photograph of a city, likely Amsterdam, featuring a large bridge spanning a river. The city's architecture and the bridge's structure are visible in the background.

# SOFTWARE DESIGN ESSENTIALS

---

HOOFDSTUK 8: APPLICATION LIFECYCLE MANAGEMENT

# APPLICATION LIFECYCLE MANAGEMENT (ALM)

- Application Lifecycle Management (ALM) omvat het continue beheer van de volledige levenscyclus van een applicatie, waarbij de brug wordt geslagen tussen business management en software engineering. Hierdoor bestrijkt ALM de volledige softwareontwikkelingscyclus.
- *ALM = alles van idee tot uitrol*
- Brug tussen **business** en **development**
- Ondersteund door tools (bv. Azure DevOps, Jira...)

# UITDAGINGEN

- Technische uitdagingen (versiebeheer, testen)
  - Het bijhouden van wijzigingen in verschillende versies en het bijhouden van wijzigingen tussen teams kan complex zijn.
- **Communicatie en samenwerking**
  - Coördinatie tussen diverse teams (ontwikkelaars, testers, en operations) vereist een sterk communicatiesysteem om misverstanden te vermijden.
- Operationele schaalbaarheid
  - Naarmate de omvang van het project groeit, wordt het beheren van de hele levenscyclus steeds complexer, wat de behoefte aan schaalbare processen en tools vergroot.

# UITDAGINGEN

- **Testen en kwaliteitscontrole**
  - Het waarborgen van de kwaliteit van de software door consistente en uitgebreide tests uit te voeren, kan uitdagend zijn, vooral met frequente releases.
- **Release management**
  - Het plannen en uitvoeren van software-releases zonder de bedrijfsvoering te verstoren is een uitdaging, vooral in omgevingen met continue integratie en implementatie.
- **Gebruikersadoptie en training**
  - Teams moeten vertrouwd raken met de tools en processen van ALM, wat training en aanpassingstijd vereist.

---

## ALM BESTAAT UIT 3 LUIKEN



GOVERNANCE




DEVELOPMENT




OPERATIONS



# **APPLICATION GOVERNANCE**



GOVERNANCE = “WIE BESLIST WAT  
ER GEBOUWD WORDT?”



# WAT IS GOVERNANCE IN ALM?

Governance is het **kader** waarmee een organisatie haar IT-activiteiten afstemt op haar zakelijke doelen.

Het start vaak met een **business case**: een onderbouwd idee voor een applicatie, bedoeld om een **concreet zakelijk resultaat te bereiken**.

Daarbij wordt rekening gehouden met:

- het **gebruik van middelen** (zoals tijd, geld en personeel),
- **veiligheid van gegevens**, en
- **toegangsrechten voor gebruikers**.
- Zo helpt governance om te zorgen dat de organisatie **voldoet aan interne regels en externe wetgeving**, en dat de juiste keuzes worden gemaakt **vóór** er iets gebouwd wordt.

# WAAROM IS GOVERNANCE BELANGRIJK BIJ APPLICATIES?

- **Koppeling met de bedrijfsdoelen**
  - Zorgt ervoor dat de applicatie écht iets bijdraagt aan wat de organisatie wil bereiken.
- **Duidelijke verantwoordelijkheden**
  - Wie beslist? Wie is eigenaar van de applicatie? Wie mag wat aanpassen?
- **Slimme beslissingen**
  - Er is een vaste manier om beslissingen te nemen over aanpassingen, budget en onderhoud.
- **Beheersing van risico's**
  - Denk aan datalekken, slechte prestaties of fouten door verkeerde toegang.
- **Meten is weten**
  - Er worden afspraken gemaakt over hoe je **prestaties en gebruik** van de applicatie opvolgt.



## VOORBEELD: GOVERNANCE BIJ EEN STUDENTENVOLGSYSTEEM

- Een hogeschool wil een nieuwe applicatie bouwen om studentenresultaten op te volgen.
- **Business case:**
  - Minder administratie
  - Betere feedback voor studenten
  - Automatisch rapporten genereren
- **Governance beslist over:**
  - Wie mag resultaten zien of aanpassen?
  - Hoe worden persoonsgegevens beveiligd?
  - Past dit binnen het IT-budget en de planning?
- Typische vragen:
  - *“Draagt deze app bij aan onze onderwijsvisie?”*
  - *“Wie mag de puntentabel aanpassen?”*
  - *“Hoe snel moet het systeem zijn volgens onze standaard?”*
- Pas als dit duidelijk is, start de ontwikkeling.



## VOORBEELD: GOVERNANCE BIJ EEN MOBIELE BANKAPP

- Een bank wil een nieuwe **mobiele app** bouwen waarmee klanten hun saldo kunnen raadplegen, betalingen uitvoeren en meldingen ontvangen.
- **Business case:**
  - Verbeterde digitale klantervaring
  - Minder druk op fysieke kantoren
  - Snellere service voor standaardverrichtingen
- **Governance beslist over:**
  - Hoe worden klantgegevens veilig opgeslagen en verwerkt?
  - Wie krijgt toegang tot welke gegevens (bv. supportmedewerkers)?
  - Valt de ontwikkeling binnen het IT-budget en het securitybeleid?

## VOORBEELD: GOVERNANCE BIJ EEN MOBIELE BANKAPP

- **Externe vereisten:**
  - **Wetgeving:** De app moet voldoen aan regelgeving zoals de GDPR (gegevensbescherming).
  - **Toezichthouders:** Financiële autoriteiten eisen rapportering en monitoring van transacties.
  - **Industriestandaarden:** De app moet veilige betaalverwerking ondersteunen (bv. PCI-DSS).
  - **Klantenverwachtingen:** De app moet betrouwbaar, veilig en eenvoudig in gebruik zijn.
- **Typische vragen:**
  - *“Is deze app conform de GDPR en auditregels?”*
  - *“Wie mag klanttransacties inkijken?”*
  - *“Is de betaalflow snel en intuïtief genoeg voor klanten?”*
- Door governance toe te passen, zorgt de bank ervoor dat de app voldoet aan alle interne én externe vereisten voordat de ontwikkeling begint.
- Zo bouwt ze vertrouwen op bij klanten én toezichthouders, en beperkt ze risico's.

# APPLICATION DEVELOPMENT

- Application development is het proces waarbij een team software ontwikkelt om **een concreet probleem op te lossen of een behoefte te vervullen**.
- Het gaat verder dan enkel programmeren. Het omvat:
  - het **plannen** van wat er gebouwd moet worden,
  - het **ontwerpen** van de structuur en gebruikersinterface,
  - het **bouwen** van de code,
  - het **testen** om fouten op te sporen,
  - het **implementeren** (in productie brengen),
  - én het regelmatig **updaten en verbeteren** van de applicatie.
- Het is een **teamproces** waarin ontwikkelaars, testers, ontwerpers en stakeholders samenwerken om software te maken die werkt én waardevol is.

# APPLICATION OPERATIONS

- **Operations** gaat over het **uitrollen, draaiende houden en opvolgen** van applicaties nadat ze ontwikkeld zijn.
- Het doel? Zorgen dat applicaties **snel, stabiel en betrouwbaar** blijven werken.
- Dit omvat onder andere:
  - Het **monitoren van prestaties** (zoals laadtijd en foutmeldingen)
  - Het **beheer en onderhoud** van applicaties
  - Het snel reageren op problemen of storingen
- Alles wat nodig is om een applicatie **dag in, dag uit operationeel en bruikbaar** te houden.

## WAT DOEN ALM-TOOLS?

ALM-tools helpen teams om het **hele ontwikkelproces van een applicatie te beheren** – van idee tot onderhoud. Ze bieden ondersteuning bij veelvoorkomende uitdagingen en zorgen dat iedereen vlot kan samenwerken.

Typische onderdelen van een ALM-tool zijn:

- **Vereistenbeheer**
  - Documenteren wat de applicatie moet kunnen.
- **Ontwerp en modellering**
  - Tools zoals UML om de **structuur en architectuur** visueel voor te stellen.
- **Codeontwikkeling**
  - Ondersteuning bij het **schrijven en genereren** van code.
- **Testondersteuning**
  - Uitvoeren van **tests tijdens en na ontwikkeling**.
- **Versiebeheer**
  - Code veilig **opslaan, opvolgen en samenwerken** via een versiebeheersysteem (zoals Git).
- **Code reviews**
  - Anderen laten meekijken en feedback geven op je code vóór integratie.
- Zo zorgen ALM-tools voor **overzicht, samenwerking en kwaliteitsbewaking** doorheen het hele proces.

## HOE HELPEN ALM-TOOLS IN DE PRAKTIJK?

ALM-tools bieden doorgaans oplossingen voor uitdagingen binnen Application Lifecycle Management, zoals:

Tijdens ontwikkeling:

- **Testbeheer**  
Uitvoeren en opvolgen van testcases en testresultaten.
- **Buildbeheer**  
Analyseren of de applicatie correct gebouwd is (builds slagen of falen).
- **Bug tracking**  
Fouten rapporteren, opvolgen en beheren in het team.

# HOE HELPEN ALM- TOOLS IN DE PRAKTIJK?

Tijdens en na uitrol (operations):

- **Deployment**  
Applicaties automatisch of handmatig uitrollen naar de juiste omgeving (bv. test of productie).
- **Monitoring**  
De prestaties van de applicatie opvolgen (bv. snelheid, beschikbaarheid).
- **Probleembeheer**  
In kaart brengen en oplossen van incidenten of technische storingen.



# HOE HELPEN ALM-TOOLS IN DE PRAKTIJK?

ALM-tools ondersteunen governance door:

## 1. Vereistenbeheer

- Documenteren van wat er gebouwd moet worden.
- Tools: Azure DevOps (Work Items), Jira (Epics, Stories), GitHub Issues

## 2. Projectplanning & prioriteiten

- Roadmaps, sprints, backlogs beheren.
- Tools: Jira, GitLab Projects, Azure Boards

## 3. Toegangsbeheer & rechten

- Bepalen wie toegang heeft tot welke broncode, tests, settings...
- Tools: GitHub Teams & Roles, Azure DevOps security policies

## 4. Compliance & auditlogs

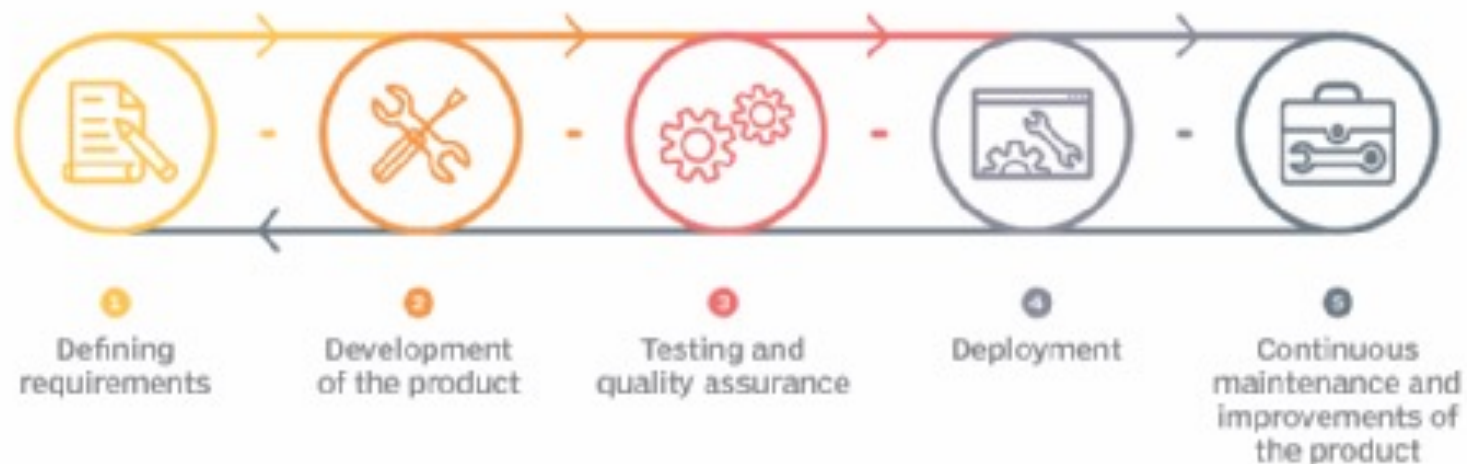
- Aantonen wie wat heeft gedaan en wanneer (versiebeheer, change tracking)
- Tools: GitLab audit logs, Azure DevOps logging, GitHub history

## 5. Besluitvorming en traceability

- Vragen beantwoorden als:
  - *“Waarom hebben we dit gebouwd?”*
  - *“Waar komen deze vereisten vandaan?”*
- Mogelijk via koppeling tussen vereisten, code, tests en issues

# ALM

## Application lifecycle management



---

# POPULAIRE TOOLS

Er bestaan verschillende softwareplatformen die organisaties ondersteunen bij **Application Lifecycle Management (ALM)**. Enkele populaire tools zijn:

- **Microsoft Azure DevOps**  
Biedt geïntegreerde tools voor **versiebeheer, CI/CD, agile planning en releasebeheer**.
- **Atlassian Jira**  
Populair voor **bugtracking en agile projectbeheer**. Helpt teams om issues op te volgen en werk te plannen doorheen de volledige levenscyclus.
- **IBM Engineering Lifecycle Management (ELM)**  
Een krachtige suite voor grote organisaties, met functies voor **vereistenbeheer, ontwerp, testing en implementatie**.
- **GitLab**  
Een all-in-one **DevOps-platform** met **versiebeheer, CI/CD, issue tracking en monitoring** in één omgeving.
- Deze tools helpen teams om hun ontwikkelproces te **organiseren, automatiseren en verbeteren**, met meer zicht op kwaliteit, voortgang en samenwerking.

---

# POPULAIRE TOOLS

## GitHub

Bekend als platform voor **versiebeheer met Git**, maar biedt intussen ook krachtige ALM-functionaliteiten:

- **Issues en Projects** voor planning
- **GitHub Actions** voor CI/CD
- **Pull requests & code review**
- Integraties met test- en deploymenttools

Voorals populair bij open source, studenten en kleinere teams – maar ook groeiend in enterprise.

# VOORDELEN

ALM-tools helpen organisaties om **sneller en betrouwbaarder software op te leveren**, die voldoet aan de gestelde eisen en regelgeving.

Ze brengen **tools, teams en processen samen in één platform**, waardoor:

- de samenwerking tussen **IT en business** vlotter verloopt,
- er **meer overzicht en transparantie** is in het ontwikkelproces,
- en software sneller én met hogere kwaliteit kan worden geleverd.

---

**ALM & AGILE:  
CONTINUOUS  
EVERYTHING**

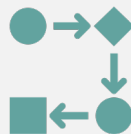
ALM is cruciaal om een effectieve **continuous integration (CI)** en **continuous delivery (CD)** omgeving te garanderen.

Het zorgt voor de structuur, tools en processen die nodig zijn om **continu te integreren, testen en opleveren van software** mogelijk te maken.

## CONTINUOUS INTEGRATION



“Continuous integration has become a de facto requirement for any software team doing Agile development today. ”



“Continuous integration is building and testing your application every time you commit a change to version control”

## WAT IS CONTINUOUS INTEGRATION (CI)?

- **Continuous Integration** is een ontwikkelpraktijk waarbij programmeurs hun code **regelmatig (meerdere keren per dag)** toevoegen aan een gedeelde codebasis, de zogenaamde **repository**.
- Elke keer als iemand code incheckt:
  - wordt er **automatisch een build uitgevoerd**,
  - en worden er **tests gestart** om te controleren of alles nog goed werkt.
- Zo kan het team **snel fouten opsporen**, en zie je meteen of de nieuwe code:
  - technisch correct is,
  - én geen bestaande functionaliteit breekt.
- Het doel van CI is om problemen **vroeg te detecteren**, zodat integratie vlot en betrouwbaar blijft verlopen.



## WAT IS CONTINUOUS DELIVERY (CD)?

- **Continuous Delivery** bouwt verder op **Continuous Integration** en zorgt ervoor dat de software **op elk moment klaar is om uit te rollen**.  
Of anders gezegd: *“je code is altijd in een staat waarin je kunt releasen”*.
- Wat betekent dat concreet?
  - Elke wijziging die in de code komt, **doorloopt automatisch alle tests**.
  - De applicatie wordt klaargezet voor release — inclusief de juiste **instellingen en configuraties**.
  - Een nieuwe versie kan met één druk op de knop worden uitgerold.
- Dankzij CD kunnen teams:
  - **vaker en sneller releasen**,
  - **sneller feedback krijgen** van echte gebruikers,
  - en sneller inspelen op fouten of nieuwe behoeften.
- CD maakt softwareontwikkeling **sneller, betrouwbaarder en flexibeler**.

## WAT IS CONTINUOUS DEPLOYMENT?

- **Continuous Deployment** is de volgende stap na Continuous Delivery.  
In plaats van manueel op een knop te drukken om een release uit te voeren, wordt **elke wijziging die slaagt voor de tests automatisch uitgerold naar de productieomgeving**.
- Dat betekent:
  - Geen handmatige goedkeuring meer nodig
  - Elke geslaagde commit gaat **rechtstreeks live**
- Deze aanpak is vooral geschikt voor organisaties die:
  - hun CI/CD-processen goed onder controle hebben,
  - **snel willen inspelen op feedback**,
  - en werken in een omgeving waar gebruikers **realtime veranderingen aankunnen** (zoals webapplicaties).
- Continuous Deployment zorgt voor **ultrasnelle releasecycli**, maar vereist **volledige vertrouwen in je testautomatisering**.

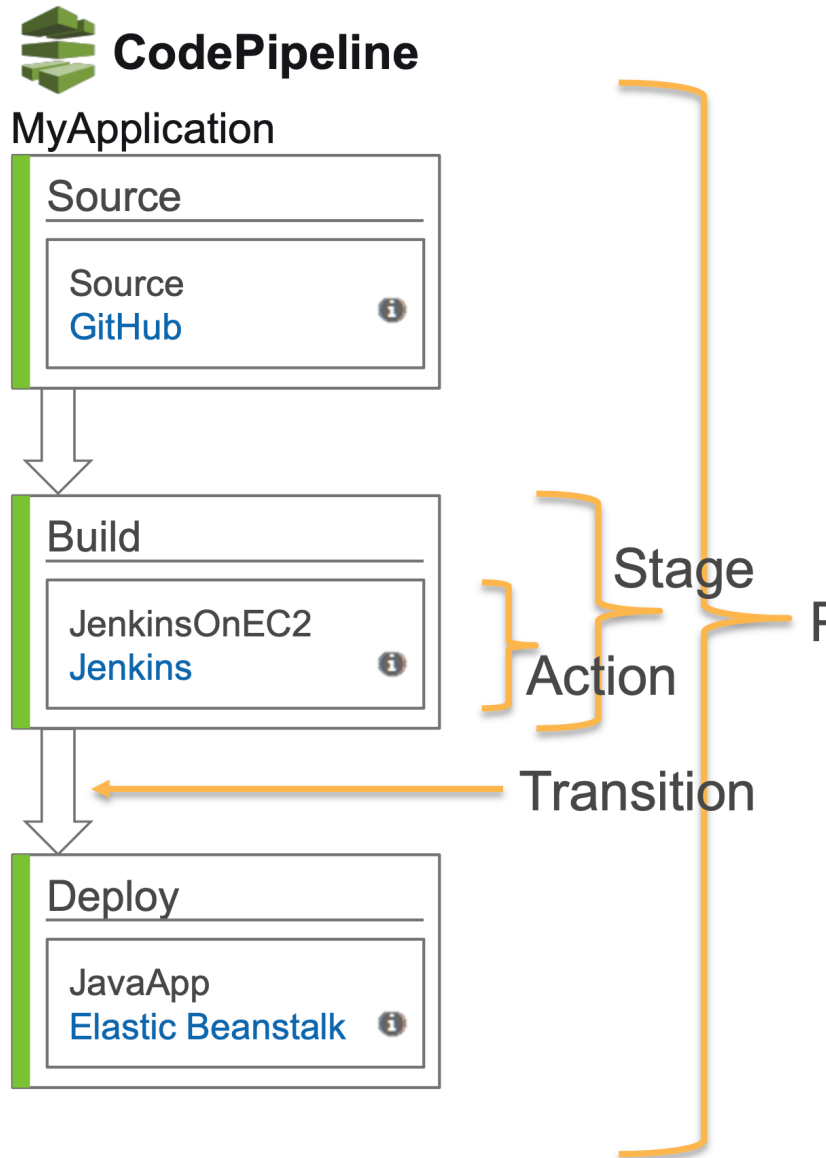
## WAT IS EEN CI/CD-PIPELINE?

- Een CI/CD-pipeline is een reeks **geautomatiseerde stappen** die software **vlot en betrouwbaar** van code naar productie brengen.
- Ofwel: *“Van inchecken tot live – volledig automatisch.”*
- De pipeline voert onder andere uit:
  - het **bouwen** van de code (build),
  - het **uitvoeren van tests**,
  - en het **uitrollen** naar een staging- of productieomgeving.
- Dit gebeurt op basis van **scripts of configuratiebestanden** die telkens op dezelfde manier draaien, zodat:
  - fouten sneller worden opgespoord,
  - releases consistentener verlopen,
  - en het hele proces **minder handmatig werk vereist**.
- Het resultaat: **snelle, veilige en herhaalbare softwarelevering**.

# PIPELINES

Voordelen voor ontwikkeling

- Sneller
- Veiliger
- Vereenvoudiging en standaardisatie
- Visualisatie van het proces





# **SOFTWARE VERSION CONTROL**

---

## DE BELANGRIJKSTE TAKEN VAN VERSION CONTROL



BIJHOUDEN WAT ER  
GEWIJZIGD WERD



BIJHOUDEN WIE EEN  
WIJZIGING HEEFT  
AANGEBRACHT



BIJHOUDEN WAAROM EEN  
WIJZIGING WERD  
AANGEBRACHT

---

# GEBRUIK

Version control systemen bewijzen hun nut in tal van beroepen, gaande van developers/designers over schrijvers/producers tot artiesten en componisten.

Het bijhouden van het pad dat je tijdens eender welk creatief proces (bv. programmeren) hebt doorlopen is dan ook zeer belangrijk.

## REDENEN

**Peace of mind:** alles wat we schrijven wordt geback-upt.

**Geschiedenis:** alles wat je gewijzigd hebt en vooral *waarom* wordt bijgehouden. Hierbij kunnen we bij elke wijziging een kort bericht toevoegen waarin we beschrijven wat er gewijzigd werd.

**Undo:** gemakkelijk naar vorige versies teruggaan.

**Experimenteren:** in een version control systeem is het eenvoudig om een sandbox op te zetten waarin je allerlei zaken kan uitproberen, zonder te vrezen dat er dingen mislopen.



## ALS TEAM



**Synchronisatie:** elke team member blijft up-to-date doordat iedereen toegang heeft tot de gemeenschappelijke inhoud



**Accountability:** wanneer iemand van het team een wijziging doorvoert wordt dit door het systeem bijgehouden en kan deze persoon er op aangesproken worden als er iets misloopt.



**Conflicten detecteren:** het version control systeem kan ervoor zorgen dat de *build* clean blijft door ervoor te zorgen dat er bijvoorbeeld geen bestanden of wijzigingen in bestanden van andere mensen worden overschreven.

# VERSION CONTROL CONCEPTEN

<b>Repository</b>	Een soort database waarin alle mappen, bestanden én hun wijzigingsgeschiedenis worden bewaard. Dit is het hart van het versiebeheersysteem.
<b>Working set (working copy)</b>	Een lokale kopie van de bestanden op je eigen computer. Bevat je wijzigingen, zelfs als die nog niet naar de repository zijn doorgestuurd.
<b>Add</b>	Een nieuw bestand toevoegen aan je working copy zodat het klaar is om opgenomen te worden in de repository.
<b>Commit (check-in)</b>	De wijzigingen die je lokaal maakte, opslaan in de repository. Zo worden ze gedeeld met het team en krijg je een nieuwe versiestap.
<b>Update (check-out)</b>	De laatste wijzigingen uit de repository binnenhalen in jouw working copy. Zo blijf je up-to-date met wat anderen hebben gedaan.
<b>Tag (label)</b>	Een benoemde momentopname van de repository, bv. bij een release. Zo kan je er later makkelijk naar terugverwijzen (bv. "v1.0").
<b>Revert (rollback)</b>	Een bestand of map terugzetten naar een vorige versie uit de repository. Handig als er iets is misgelopen.

## VOORBEELDEN VAN VERSION-CONTROL SOFTWARE

- Hier zijn enkele populaire voorbeelden van versiebeheersoftware:
  - **Git**  
Het meest gebruikte versiebeheersysteem vandaag.  
Ondersteunt **gedistribueerd werken**, waarbij elke ontwikkelaar een volledige kopie van de repository heeft.  
Wordt gebruikt op platforms zoals **GitHub**, **GitLab** en **Bitbucket**.
  - **Subversion (SVN)**  
Een ouder, **centraal versiebeheersysteem**.  
Vooraf nog gebruikt in legacyprojecten of bij organisaties die liever een centrale server gebruiken.
  - **Mercurial**  
Ook een **gedistribueerd systeem**, vergelijkbaar met Git.  
Bekend om zijn eenvoud en gebruiksvriendelijkheid, maar tegenwoordig minder populair.