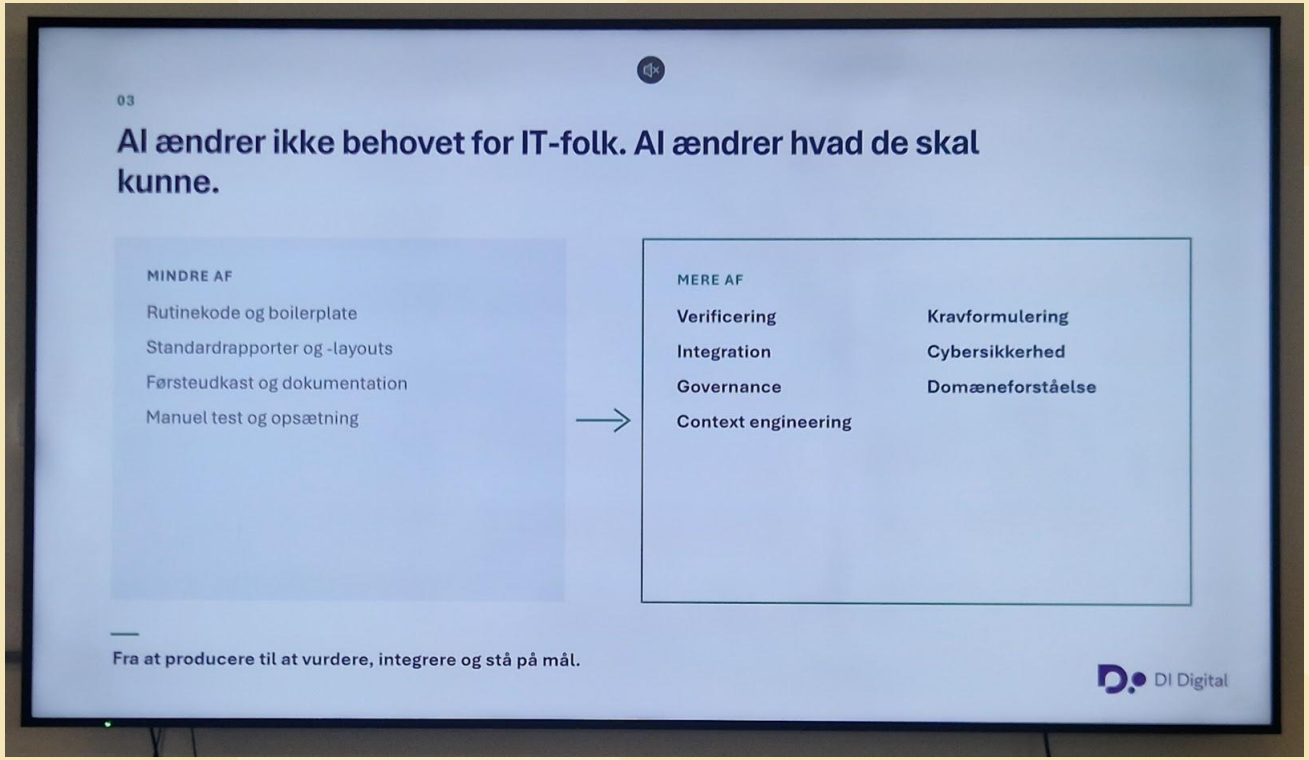
ERHVERVSAKADEMI KØBENHAVN

Reflect & Refactor



Ugens formål

- Konsolidering af jeres viden fra de sidste otte uger
- Refleksion over egen kode
  - Single Responsibility Principle (SRP)
  - Separation of Concerns (SoC)
  - Model-View-Control (MVC)
  - Dependency Injection
  - Normalisering
- Refaktorering
  - Hvor er vi blevet klogere og hvad vil vi ændre?
  - Hvad vil vi ikke ændre?
- Hvad er færdigt?
  - Hvilke user stories er opfyldt?
  - Er programmet testet?



Single Responsibility Principle (SRP)

- En klasse skal kun have én grund til at ændre sig
- En klasse skal kun have ét tydeligt ansvar

Så hvad kan få koden til at ændre sig? Fx:

- Loginreglerne ændrer sig.
- Måden, brugerne gemmes på, ændrer sig.
- Parametre fra hjemmesiden ændrer sig.

Hvilke af disse bør give anledning til at `UserController`, `UserService` eller `UserMapper` ændrer sig?

Følger jeres klasser SRP? Er der nogen, der ikke gør?

Men er det ikke bare Separation of Concerns?

- Jo, det hænger tæt sammen
- SRP handler om de enkelte klasser
- SoC går på tværs af systemet og hjælper os med at dele det op i ansvarsområder

Fx

- `UserService` tager sig af forretningslogik
- Men hvis `UserService` fx både validerer passwords og sender mails til brugeren, har vi måske samlet for mange forskellige ansvarsområder i samme klasse
- Vi ønsker et fleksibelt system, som er let at vedligeholde og som har en løs kobling mellem klasserne og en høj sammenhængskraft (coherence) i de enkelte klasser

Anvender I SoC? Er der steder, hvor I ikke gør?

Men er det så det samme som MVC?

- MVC er en arkitektur som understøtter SoC og SRP
- Når vi deler ansvar op i de tre hovedområder M-V-C er vi allerede på vej til at designe et system med løs kobling og høj sammenhængskraft

- Overvej følgende i jeres eget projekt:
  - Hvad sker der, hvis I ændrer jeres frontend?
  - Hvad sker der, hvis I ændrer jeres database?
  - Hvilke klasser skal ændres, og hvilke kan forblive uændrede?
  - Hvilke fordele giver det, at ændringer kan begrænses til bestemte dele af systemet?

Dependency Injection

- Hvor bruger I dependency injection?
- Hvilke fordele giver DI jer?
- Hvad ville alternativet være?
- Hvilke problemer kunne det give?

Hint: tænk over begreber som fleksibilitet, testbarhed og løs kobling

Normalisering

- Hvordan har I struktureret og normaliseret jeres database?
- Hvilke problemer hjælper normalisering med at forebygge?
- Hvilke konsekvenser kan det have, hvis de samme oplysninger gemmes flere steder?

Hint: tænk over konsistens, dataintegritet og redundans.

Refaktorering

- I kan nok ikke nå at refaktorere alt det, I gerne vil. Læg en plan, og prioritér det vigtigste
- Overvej, hvilke konsekvenser en refaktorering kan have for resten af koden. Hvilke andre dele af systemet kan blive påvirket?
- Koordinér i gruppen, hvem der refaktorerer hvad, og hvordan I gør det. Ellers risikerer I hurtigt at få en masse merge-konflikter!

Hint: Husk at refaktorering skal forbedre kodens struktur uden at ændre dens funktionalitet

Test

- Har I skrevet unit tests for de metoder, der kan testes isoleret fra andre systemer?
- Har I lavet integrationstests af jeres data mappers, så I undersøger, om de fungerer korrekt sammen med databasen?

Overvej følgende

- Hvilke dele af systemet er mest kritiske at teste?
- Hvilke edge cases bør I teste?
- Arbejder I med black-box- eller white-box-test? Hvorfor?

Programmets tilstand – hvor langt er I?

- Jeres fremdrift kan vurderes ud fra, hvor mange user stories I har færdiggjort, og hvor godt de opfylder jeres krav.
- En user story er færdig, når dens accept kriterierne er opfyldt
- En test af hvorvidt acceptkriterier er opfyldt kaldes en User Accept Test (UAT)
- I skal være jeres egne brugere: Gennemgå jeres user stories og tilhørende acceptkriterier, og undersøg, om funktionaliteten lever op til dem.

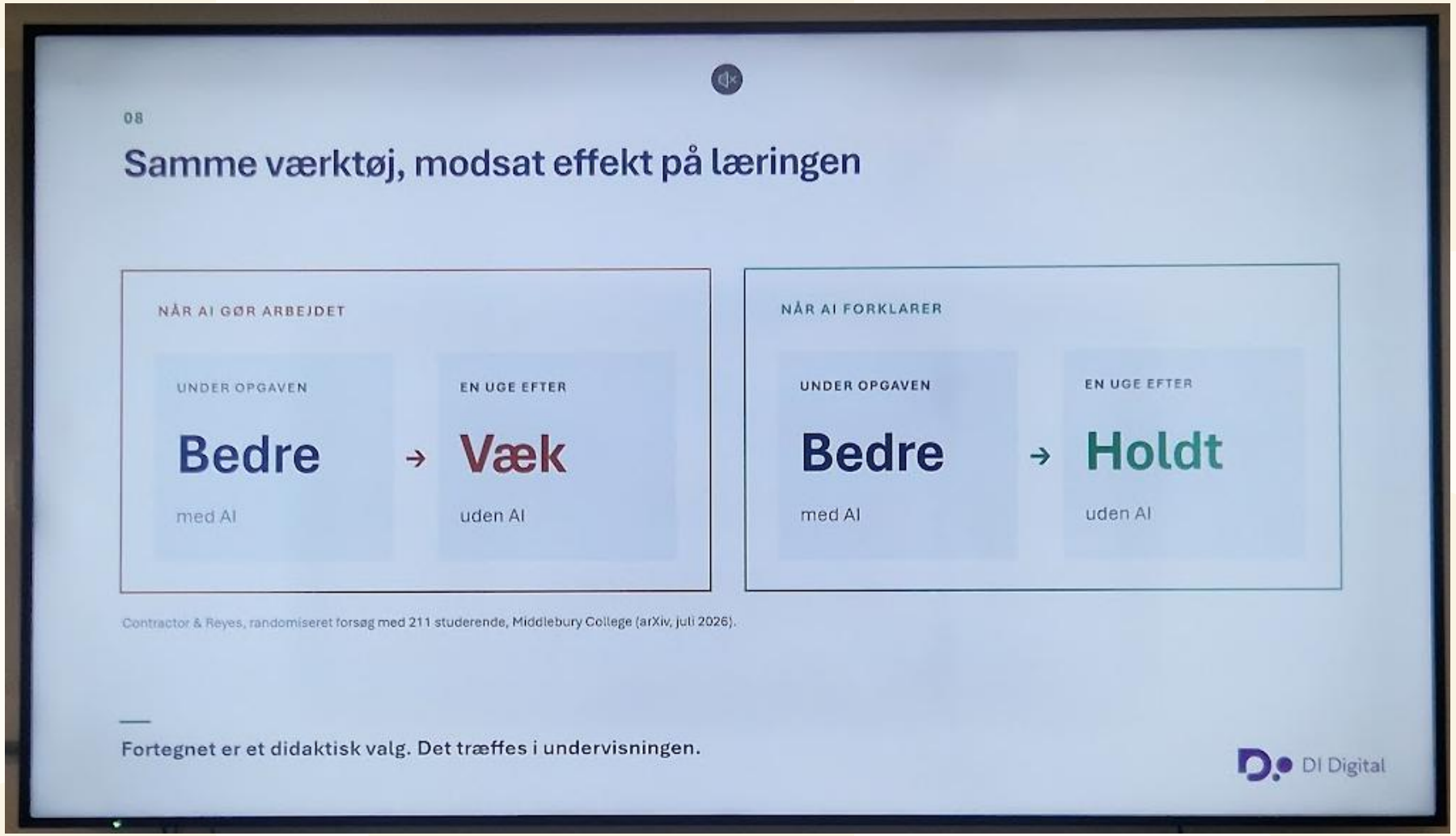
Overvej følgende

- Har I holdt jer til user stories eller har I lavet ekstra features?
- Har I fulgt jeres prioritering af user stories?
- Har I brugt backlog og GitHub Project Board til at holde styr på arbejdet?
- Holdt jeres estimater for user stories?
- Var jeres acceptkriterier konkrete nok til, at I kunne teste, om de var opfyldt?

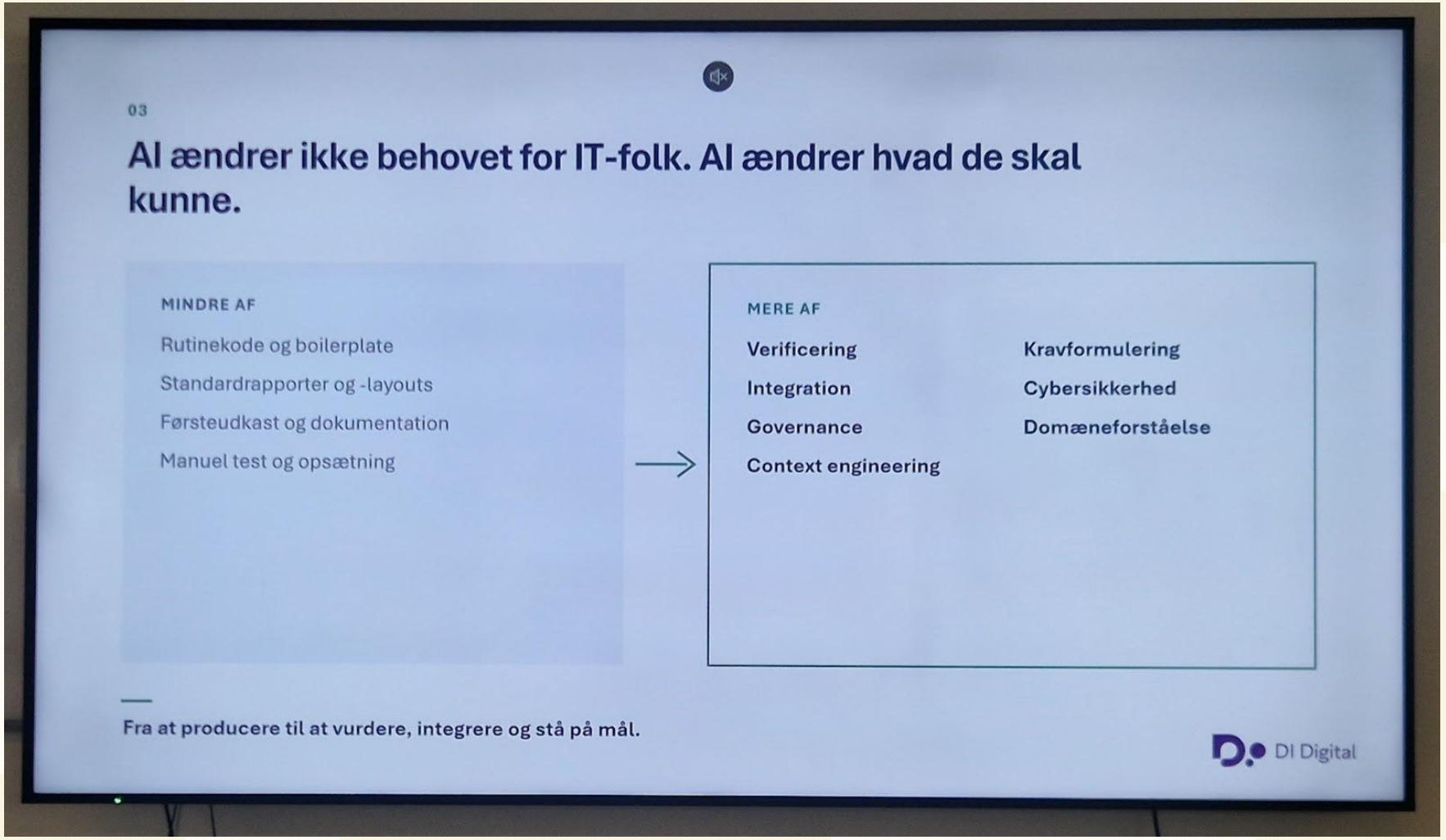
Torsdagspræsentation

- Hvad har I refaktoreret og hvorfor?
- Hvor langt er I med jeres user stories?
- Hvordan har I testet jeres kode?

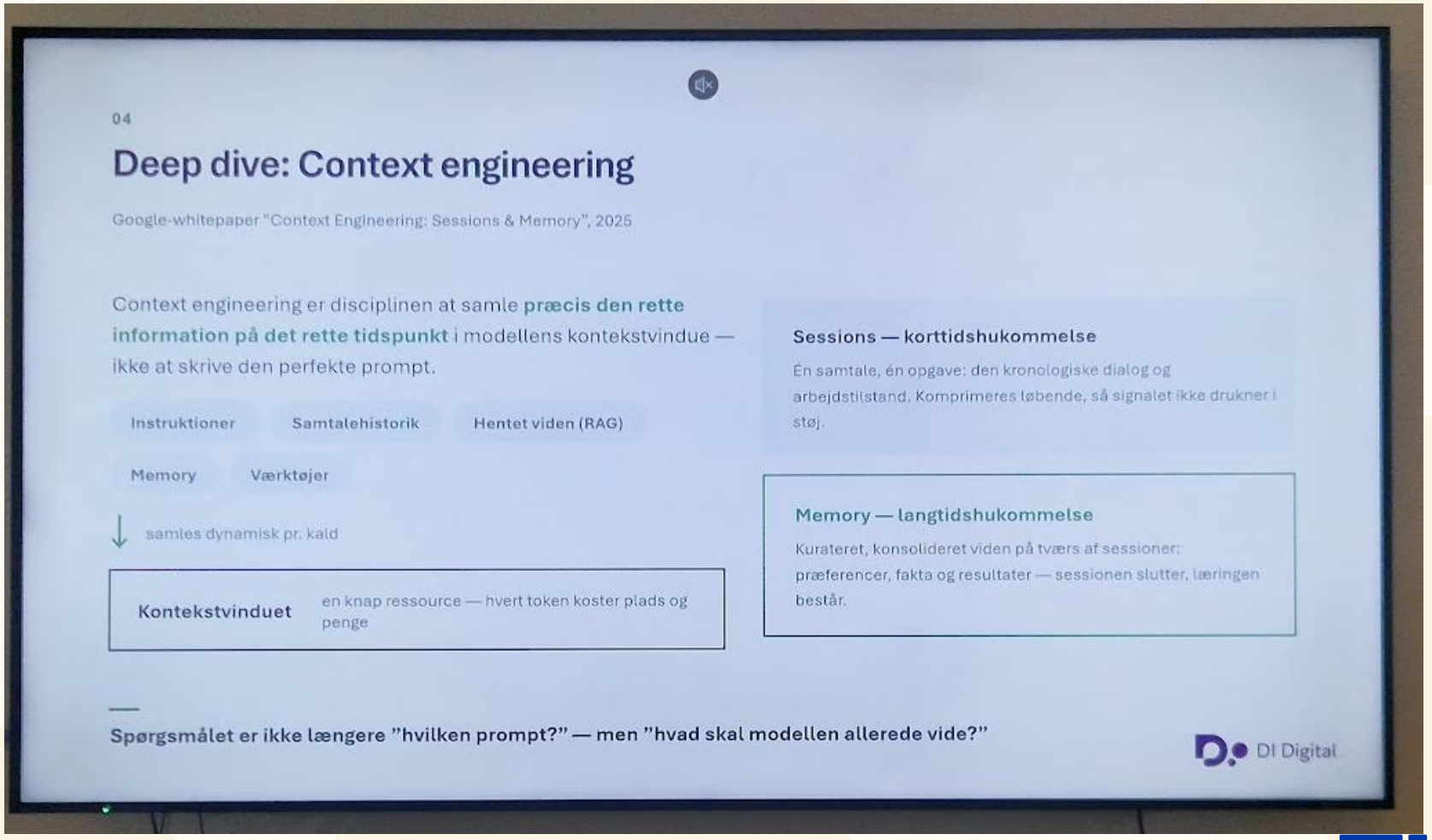
Brug af AI


08
Samme værktøj, modsat effekt på læringen
NÅR AI GØR ARBEJDET
UNDER OPGAVEN
Bedre
med AI
EN UGE EFTER
Væk
uden AI
NÅR AI FORKLARER
UNDER OPGAVEN
Bedre
med AI
EN UGE EFTER
Holdt
uden AI
Contractor & Reyes, randomiseret forsøg med 211 studerende, Middlebury College (arXiv, juli 2026).
Fortegnet er et didaktisk valg. Det træffes i undervisningen.
DI Digital

AI kompetencer



Context engineering



Hvordan kan I give kontekst?

- Skriv baggrunden for jeres spørgsmål enten i indstillinger for et LLM-projekt eller direkte i prompten
- Upload al det materiale I har. Link til kursussiden, slides, opgaver, eksempelkode, kode I selv har lavet
- Fortæl tydeligt hvad formålet med jeres spørgsmål er

Er mit program okay?

Jeg er datamatikerstuderende på 2. semester. Vi arbejder i en gruppe på et pet project, hvor vi skal udvikle et fullstack-system. Vi bruger Java, Javalin, JavaScript og PostgreSQL. Vi vil gerne forstå og forbedre vores eksisterende løsning, men vi vil ikke skifte teknologi eller bygge hele systemet om. Hjælp os med at undersøge vores kode og forstå mulige forbedringer. Forklar dine forslag, og tag én problemstilling ad gangen.

Prompt vs context engineering



