BØRNE- OG
UNDERVISNINGSMINISTERIET
STYRELSEN FOR
UNDERVISNING OG KVALITET

Vejledning til Programmering B, valgfag

August 2024

Vejledning til Programmering B, valgfag

August 2024

2024

ISBN nr. [xxx xxx xxx] (web udgave)

Design: Center for Kommunikation og Presse

Denne publikation kan ikke bestilles.

Der henvises til webudgaven.

Publikationen kan hentes på:

www.uvm.dk

Børne- og Undervisningsministeriet

Departementet

Frederiksholms Kanal 21

1220 København K

Indhold

Indledning

Vejledningen præciserer, kommenterer, uddyber og giver anbefalinger vedrørende udvalgte dele af læreplanens tekst, men indfører ikke nye bindende krav.
Citater fra læreplanen er anført i citationstegn.

Følgende ændringer er foretaget i vejledningen i maj 2024:

- Afsnit om hvordan generativ AI kan inddrages i undervisningen

1 Identitet og formål

1.1 Identitet

"Programmering omfatter de metoder og teknikker, der anvendes til at få programmerbare it-komponenter til at udføre planlagte handlinger. Faget har en betydelig praktisk dimension med programmering af enkle it-komponenter og demonstration af mere komplekse teknologiske systemer. Faget er et teknisk og kreativt fag med vægt på eksperimentelle og innovative processer, samt et videns- og kundskabsfag. Disse sider af faget betinger gensidigt hinanden og sikrer faglig dybde. "

I faget arbejdes der med at udvikle programmer og programstumper, og på at forstå programmer. Der arbejdes med programkonstruktioner, og der arbejdes med udviklingsmetoder og dokumentation.

1.2 Formål

Programmering er et fagfelt der kan udvikle elevernes innovative og skabende kompetencer. Det handler ikke bare om, at eleverne skal kunne programmere. Det handler også om at kunne tænke selv, kunne samarbejde omkring idéudvikling, skabelse og videndeling. Hermed er programmering også med til at udvikle elevernes digitale- og studiekompetencer. Programmering kan medvirke til at skabe interesse for teknologi og it, nationalt og globalt, og dermed også bidrage til elevernes karrierekompetencer, samtidig med at de bliver introduceret til abstrakt tænkning, logisk opbygning og problemløsning.

Blandt de faglige mål indgår, at eleverne skal kunne:

- "demonstrere viden om fagets identitet og metoder,
- løse en enkelt problemstilling gennem udviklingen af et program bl.a. i samspil med andre fag."

Dette betyder, at der ikke alene undervises i programmering, men også om programmering. Når faget indgår i et samarbejde med andre fag, uanset om det er i studieområdet (SO), studieområdeprojektet (SOP), eller det er et fagligt samarbejde i en studieretning, så skal eleverne kunne inddrage og anvende relevante programmeringsmæssige metoder, redegøre for disse i et sprog, som man også uden for faget kan forstå, samt forholde sig til fagets muligheder og begrænsninger i arbejdet med den konkrete problemstilling.

I læreplanens afsnit 1.1 og 1.2 er der givet en kompakt beskrivelse af fagets identitet og af det overordnede formål med undervisningen.

2 Faglige mål og fagligt indhold

2.1 Faglige mål

Eleverne skal kunne:

- "bruge programmering til at undersøge et emne eller problemområde, med henblik på via programmets funktion at skabe ny indsigt eller til at løse et problem"

Programmering i gymnasiet giver eleverne muligheder for selv at skabe produkter fra bunden. Timerammen i programmering på B niveau giver mulighed for at eleverne opnår sikkerhed og rutine i at udvikle programmer, og dermed kan programmering blive et værdifuldt værktøj, som eleverne kan inddrage i andre sammenhænge. Når eleverne i programmering arbejder med et emne fra et andet fag, giver det mulighed for at bearbejde problemstillinger på en anden måde, end fagene normalt ville kunne. Det kan f.eks. være i form af beregning, visualisering, simulering, test eller eksperiment. Dermed kan eleverne opnå ny og mere nuanceret indsigt i emnet.

- "behandle problemstillinger i samspil med andre fag"

Det er oplagt at tage udgangspunkt i problemstillinger fra elevernes hverdag, problemer fra andre fag, programmer, der ligner dem eleverne bruger til hverdag på forskellige platforme, eller simpelthen det, der interesserer den enkelte elev. Man kan også tage udgangspunkt i problemstillinger fra de globale udfordringer, vi står over for. Mulighederne for at bruge programmering til at skabe programmer, der på en eller anden måde kan bruges i elevernes hverdag, udvides til stadighed med apps, programmerbare komponenter, Internet of Things, og den hurtigt voksende mængde data, der er tilgængelig på nettet. Formålet er ikke, at eleverne skal lære en masse forskellige programmeringssprog eller teknologier at kende, men at de indser, at de kompetencer, de tilegner sig kan anvendes i en bred vifte af aktiviteter. Det gøres ved at lade eleverne være kreative og udforskende i deres arbejde med udviklingen af programmer. Behandling af forskellige problemstillinger fra andre fag vil styrke elevernes karrierekompetence, idet de opnår forståelse for hvordan programmering som professionelt fag sjældent arbejder på at løse fagets egne problemstillinger, men ofte på at bidrage til løsning af problemstillinger fra andre fagområder. Inddragelse af programmering som et værktøj i andre fag vil ligeledes styrke elevernes muligheder for innovation. Ved at tage udgangspunkt i globale problemstillinger styrkes elevernes globale kompetencer.

Som studieretningsfag - hvor man arbejder med en samlet klasse - kan programmering indgå i samspil med stort set alle andre fag. I problembaserede undervisningsforløb kan programmeringsfagets metoder bruges til analyse af problemstillinger på lige fod med andre fag. De fleste forløb kan udformes, så eleverne får mulighed for at arbejde med de særfaglige mål fra andre fag på en ny måde, f.eks. gennem udviklingen af et program.

- "anvende avancerede konstruktioner i et programmeringssprog"

På C-niveau lærer eleverne om grundlæggende konstruktioner, og på B-niveau avancerede konstruktioner. Kernestoffet nævner en række konkrete begreber, såsom parametrisering, rekursion og polymorfi, og eleverne skal præsenteres for løsninger, hvor disse bruges. Her anbefales det at følge de didaktiske

principper, især Use-Modify-Create (Se også afsnit 3.1 Didaktiske Overvejelser), da mange elever formentlig vil have svært ved at anvende de mere abstrakte konstruktioner direkte.

- "redegøre for arkitekturen af programmer på forskellige abstraktionsniveauer, herunder relationen mellem brug og funktion"

En vigtig kompetence for programmører består i at kunne kommunikere om programmering med forskellige mennesker. En effektiv problemløsning involverer alt fra inddragelse af eksterne interessenter til grundlæggende viden om de tekniske detaljer i implementeringen, og eleverne skal være bevidste om forskellen på at kommunikere med en anden programmør om valg af arkitektur og på at kommunikere med en potentiel bruger om programmets funktion.

Den dygtige elev vil kunne veksle naturligt mellem teknisk korrekt fagsprog og naturligt sprog, afhængigt af konteksten. Dette understreges i bedømmelseskriterierne til eksamen, hvor der lægges vægt på elevens udvælgelse af relevant stof til præsentationen. Eleverne skal have mulighed for at udvikle deres fagsprog i undervisningen, hvorfor det er relevant at variere arbejds- og samarbejdsformer i undervisningen.

Det er formentlig kun de stærkeste elever, der aktivt vil kunne tage beslutninger om arkitekturen af større systemer, men det forventes, at alle eleverne kan forstå og redegøre for opbygningen af udvalgte programmer. For eksempel kan de fleste elever identificere hvilket lag i trelagsmodellen, en given del af koden hører til, mens ikke alle elever vil kunne diskutere fordele og ulemper ved at flytte funktionalitet fra et lag til et andet.

- "redegøre for simple specifikationsmodeller og realisere disse i simple velstrukturerede programmer samt teste disse"

Målet fra C-niveau om at programmere et fungerende system udvides til at omhandle

- Specifikationsmodeller som grundlag for et program. Det kan f.eks. være pseudokode, flowdiagrammer, klassediagrammer og lignende. Eleven kan både implementere en given specifikation i sit program og trække relevant dokumentation ud af et eksisterende program. Eleven viser dette i dokumentationen af eksamensprojektet.
- Et velstruktureret program. Definitionen på et velstruktureret program kan variere med blandt andet programmeringsparadigmet, programmeringssproget og fokus i undervisningen, men i alle tilfælde er det hensigtsmæssigt, at eleven er i stand til at foretage begrundede valg om opbygning af programkoden.
- Et program, der er blevet testet. Dette omhandler såvel funktionel test som brugertest. Det er en god ide, at eleverne introduceres til fejlsøgning i koden med de muligheder som udviklingsmiljøet stiller til rådighed, og til brugertest af programmet i forhold til de opstillede krav. Begge former for test indgår som en naturlig del af den iterative udviklingsmetode.

Her er tale om en ret stor progression i forhold til hvilke krav vi stiller til elevernes produkter og overvejelser om produktet

- "rette, tilpasse og udvide avancerede programmer"

Også her skelnes der mellem enkle programmer på C-niveau og avancerede programmer på B-niveau. En mulig skelnen mellem enkle og avancerede programmer er, at enkle programmer består af én enkelt komponent, eller set fra brugerens synspunkt kun gør én ting, hvor avancerede programmer består af flere komponenter og ofte kan flere ting. Særligt teknisk indhold i programkoden kan også være med til at gøre et program avanceret.

Ikke alle programmer, man arbejder med på B-niveau behøver at være avancerede, men eleverne skal stifte bekendtskab med større systemer, som kræver at eleven bruger abstrakt dokumentation for at skaffe sig et overblik over opbygningen af programmet. For at kunne lave deres egne udvidelser til sådanne systemer skal eleverne overveje passende snitflader til andre dele af programmet, tilpasse sig den eksisterende arkitektur, osv.

- "demonstrere viden om fagets identitet og metoder"

Programmeringsfaget adskiller sig fra de andre fag i gymnasiet, ved at være det fag, hvor den iterative udviklingsmetode for alvor giver mening. Eleverne ved allerede - fra programmer de selv bruger på deres computer, telefon, osv. - at programmer altid er under udvikling og kun sjældent anses for at være færdige produkter. Dette kan eleverne overføre til deres eget arbejde, og bevidstheden om at programmeringen ikke har til formål at gøre programmet færdigt, men at tage et lille skridt ad gangen mod en løsning, kan hjælpe alle elever med at opnå tilfredsstillende resultater i faget.

- "arbejde inkrementelt og systematisk i programmeringsprocessen"

Opfyldelse af dette mål forudsætter at eleverne reflekterer over den proces de gennemgår når de løser en programmeringsopgave. Træning i trinvis forbedring af programmer sætter eleven i stand til at komme i gang med at behandle de fleste problemstillinger, også selvom den endelige løsning ikke er kendt for eleven. Det er en usikker situation for mange elever at arbejde eksperimenterende med en opgave, de ikke kan overskue i sin helhed, men den arbejdsform er vigtig og givende for både stærke og svage elever.

Systematikken kommer også til udtryk i fejlsøgningsprocessen, hvor både stærke og svage elever kan få gavn af konkrete teknikker. I strukturering af koden, oprettelse af variable, og anvendelse af algoritmemønstre og hyppigt anvendte kontrolstrukturer kan man hjælpe eleverne med at generalisere, og indse hvilke aspekter, der er de samme hver gang, og hvilke, der skal tilpasses i situationen.

Til eksamen skal eleverne i forberedelsestiden bearbejde en faglig problemstilling. Det er hensigtsmæssigt, at opgaverne er udformede så de understøtter den iterative behandling, og samtidig tilpas åbne, så de imødekommer elevernes forskelligheder.

2.2 Kernestof

Gennem kernestoffet skal eleverne opnå faglig fordybelse, viden og kundskaber. Kernestoffet er:

- "programmeringssprog og elementer i programmers opbygning, herunder variabler, typer, udtryk, kontrolstrukturer, parametrisering/abstraktionsmekanismer, rekursion, polymorfi og algoritmemønstre"

Det er tilrådeligt, at der kun vælges ét primært programmeringssprog af hensyn til den pædagogiske tilrettelæggelse og overskueligheden for eleverne. I en differentieret undervisningssituation kan der vælges et sekundært sprog til nogle elever. I et tværfagligt samarbejde med eksempelvis matematik, kan matematikprogrammer såsom Maple og Mathcad godt fungere sideløbende med det valgte sprog. Læreplanen understøtter en multi-paradigmatisk tilgang til undervisningen i programmering. Det medfører, at man eksempelvis selv kan vælge hvorvidt man ønsker at inddrage klasser og objektorienteret programmering, funktionsprogrammering (eller et funktionelt programmeringssprog) eller procedural programmering.

Det valgte sprog skal give mulighed for at arbejde med grundlæggende data- og kontrolstrukturer så eleven får en god forståelse af programmeringssprogets opbygning.

Herunder nogle forslag til temaer inden for data og kontrolstrukturer, som kan være relevante at inddrage:

- datatyper herunder primitive typer (såsom heltal, kommatal og boolske typer) og sammensatte typer (såsom lister eller arrays),
- variabler (både lokale og globale) samt typiske operationer på disse,
- sekvenser herunder at programinstruktioner afvikles sekventielt og at de kan involvere variabler der initialiseres og opdateres samt læses/skrives,
- betinget udførsel (herunder boolske udtryk, sammenligningsoperatorer og sammensatte udtryk),
- løkker, aritmetiske beregninger,

- metoder til at operere på tekstvariable (såsom konkatenering samt operationer på delelementer af strenge),
- grundlæggende input og output af bl.a. tekst og tal, skrivning og læsning til filer samt
- metoder og/eller procedurer skal også indgå, så eleven har en forståelse af, at man ved brug af disse kan reducere kompleksiteten og vedligeholdelsen af koden, gøre programmet nemmere at udvikle samt øge den generelle læsbarhed og facilitere kollaborativ udvikling. Det er en god ide at eleven får en forståelse af at procedurer/metoder kan have parametre og returværdier samt at parametre kan generalisere en løsning ved at benytte en procedure fremfor duplikeret kode.

Eleverne skal desuden have en klar forståelse af følgende begreber og deres betydning i det valgte sprog:

- Rekursion. Eleven kan beskrive og konstruere simple rekursive metoder samt identificere problemer, der kan løses ved brug af rekursion, samt udvikle en forståelse for relationen og forskellen mellem rekursion og iteration. Ydermere har eleven en indsigt i hvornår det er en fordel at bruge rekursion frem for iteration. Indføringen af rekursion kalder på, at man inddrager datastrukturer såsom træer og grafer.
  Til at udvikle elevens forståelse af rekursion kan det måske være en fordel for eleven at inddrage rekursionstræer til at visualisere de rekursive kald.

- Polymorfi. Eleven har en grundlæggende forståelse for polymorfi og de forskellige typer polymorfi som sproget tilbyder og de trænes i at benytte polymorfi til at konstruere mere generiske og elegante løsninger. Polymorfi kan have forskellige betydninger afhængig af programmeringsparadigme. Særligt for objektorienterede sprog gælder, at det vil være relevant at inddrage nedarvning, herunder supertyper og børn heraf, samt polymorfi igennem dynamisk binding.

- Algoritmemønstre. Eleven skal kunne identificere og konstruere simple algoritmemønstre i det givne sprog. De er karakteriseret ved at være konkrete lavpraktiske procedurer, der løser en generel problemtype, som udvikleren ofte møder. Identifikationen af algoritmemønstre handler om at træne eleven i at uddrage fælles karakteristika fra specifikke programeksempler for dernæst at kunne generalisere disse koncepter til en generisk procedure. Denne udviklingsproces indbefatter, at man fjerner nogle detaljer for at kunne generalisere funktionaliteten.

- Parametrisering/abstraktionsmekanismer. Eleven har en klar forståelse af, at parametre giver muligheden for at give forskellige værdier af input til procedurer/funktioner, når de kaldes i et program, og at igennem parametrisering kan en specifik løsning generaliseres.

Det er en god ide, at eleven gennem dataabstraktion, som er en mulighed for at adskille adfærd fra den faktiske implementation, øves i at søge gradvis mere generiske og abstrakte løsninger.
Det kan i praksis ske ved eksempelvis at betone, at tekststrenge og operationer på disse (herunder konkatenering og delstrenge) er meget almindelige i en lang række programmer. Det samme gør sig gældende for lister og operationer på disse (herunder tilføje, slette og søge).

Man kan med fordel inddrage begreber abstrakte interfaces for et program samt overloading og overriding af metoder/funktioner til at træne eleven i at udvikle generiske metoder og funktioner, der kan bruges i mange situationer

- "arkitekturen for programmers interaktion med omgivelserne med henblik på hændelsesstyret interaktion og interaktion mellem systemer"

Eleven skal have en forståelse af hvordan programmer kan opbygges i forhold til den ønskede interaktion. Det kunne eksempelvis ske ved at introducere eleverne for Model-View-Controller eller specifikke designmønstre.

Eleven skal desuden have en forståelse af, at inputdata til programmer og hændelsesstyret interaktion kan antage mange former (fra tal og tekst i en konsolapplikation til museklik og berøring af en interaktiv flade m.m.) og være i stand til at repræsentere og manipulere med disse data i sit program i forhold til den givne arkitektur. Det kunne eksempelvis være hvordan man strukturere koden/softwaren bedst

muligt ift. indhentning af data fra en database (optimering af forespørgsler til database) eller inddrage tilstandsdiagrammer i udviklingen af et computerspil.
Ydermere skal eleven blive bevidst om, at interaktionen med programmer ikke behøver være drevet af et menneske, men også kan foregå indbyrdes mellem programmer. I praksis kunne eleverne eksempelvis arbejde i mindre grupper om at udvikle delprogrammer, som interagerer med hinanden igennem nogle på forhånd givende grænseflader.

- "generiske programdele og biblioteksmoduler"

Det er hensigtsmæssigt, at eleven opnår indsigt i, at konstruktionen af korrekte programmer afhænger af korrekte programdele, herunder bl.a. kodesegmenter og procedurer, og samtidig øves i at kombinere korrekte programdele, så et korrekt program konstrueres. Det er vigtigt for eleven at erkende, at inddelingen af et program i mindre uafhængige dele kan medvirke til i højere grad at facilitere kollaborative udviklingsprocesser, øge læsbarheden af koden samt gøre det nemmere at finde fejl. Det kan ske i praksis ved at inddrage Stepwise Improvement som beskrevet i 3.1: Didaktiske overvejelser.

Eleven trænes i at inddrage og anvende eksterne biblioteksmoduler/API'er og bliver herved bevidst om, at API'er bruges til at forbinde forskellige programmer, så de er i stand til at kommunikere med hinanden. Brugen af generiske programdele herunder API'er og muligvis eksterne biblioteksmoduler i udviklingsprocessen skal gøre eleven bevidste om, at sådanne biblioteker kan simplificere komplekse programmeringsopgaver, øge produktiviteten og læsbarheden samt reducere mulige fejl.

Eleven skal arbejde med at udvikle egne generiske API'er, der løser gradvist mere komplekse problemstillinger. Det skal gerne ske ved brug af blandt andet nogle af de ovenfor beskrevne begreber (herunder måske særligt polymorfi, algoritmemønstre og parametrisering/abstraktionsmekanismer). Et væsentligt aspekt i udviklingen af et API er at dokumentere dens funktionalitet.

- "arbejdsgange og systematik i programmeringsprocessen, herunder test og fejlfinding"

Arbejdsgangene er en vigtig del af programmeringsprocessen. Eleverne vejledes til at udvikle deres programmer i en iterativ udviklingsmetode, som beskrevet i 3.1: Didaktiske overvejelser.

Det er væsentligt, at eleven bliver bevidst om, at en programmør designer, implementerer, tester, debugger og vedligeholder programmer, når de løser et problem.

Nogle gode vaner vil også kunne støtte eleven i at beskrive programudviklingen i sin synopsis, så de bliver i stand til at reflektere over- samt dokumentere deres arbejde på en præcis og forståelig måde. Det er eksempelvis en hjælp at vænne eleverne til at skrive velordnede programmer med fornuftige sigende variabelnavne, anvende indrykninger der angiver strukturen i programmet, undgå duplikeret kode og for lange kodesegmenter, samt at kommentere programmet, så andre dels kan læse det, men også vil være i stand til at arbejde videre på koden. Det er med andre ord væsentligt at betone overfor eleven vigtigheden af systematisk dokumentation i forhold til facilitering af kollaborative udviklingsmiljøer, samt at ens programmeringsstil kan påvirke hvor let det er at afgøre korrektheden af programmet. Eleverne øves i at debugge deres program - dvs. lokalisere og rette fejl, og herved erkende, at viden, om hvad programmet formodes at gøre, er påkrævet for at kunne finde flest mulige fejl i programmet.

- "abstrakte programmeringsbeskrivelser og dokumentation"

For den enkelte elev er det centralt at erkende, at programmerbare enheder kun gør det, de er programmeret til. Gode eksempler fra hverdagen kan være med til at anskueliggøre dette. Et vigtigt element er at kunne læse og forklare avancerede programmers virkemåde, så de kan analysere egne programmer og undgå fejl. Ved avancerede programmer skal i denne sammenhæng forstås programmer, som inddrager rekursion, polymorfi, algoritmemønstre og andre abstraktionsmekanismer/parametriseringer.

Eleven skal kunne skelne mellem et programs statiske opbygning og dynamiske opførsel, som f.eks. giver sig udtryk i den korrekte implementering af en løkke samt dens virkemåde under programafvikling. Dialogen mellem lærer og elev og eleverne imellem er her et vigtigt element, der kan støtte eleverne i senere abstrakte beskrivelser af programmer, ligesom bl.a. pseudokode og flowchart støtter udviklingen af programstrukturen.

Det er en god ide, at brugen af pseudokode højere grad formaliseres så den afspejler inddragelsen af de mere avancerede konstruktioner.

Dokumentation af programkomponenter såsom programbiblioteker, kodesegmenter, procedurer og eksterne biblioteker skal hjælpe eleven til at udvikle korrekte programmer, der løser et problem. Ved eksempelvis at benytte pseudokode, flowdiagrammer og/eller problemtræer i udviklingsprocessen kan eleven blive bevidst om, at et programs funktionalitet bedst beskrives i et højniveau sprog (f.eks. ved at beskrive hvorledes brugeren interagerer med programmet) og ikke i et lavere niveau, der er karakteriseret ved at forklare, hvordan de enkelte instruktioner i programmet fungerer. Brugen af pseudokode kan medvirke til at træne eleverne i at identificere og formulere algoritmemønstre samt lette overgangen fra abstrakt problemstilling til konkret implementation. Ydermere vil det være relevant at inddrage UML-diagrammer, herunder blandt andet klassediagrammer, til at modellere brugen af polymorf (og nedarvning) i udviklingen samt give eleverne en bedre forståelse af begrebet polymorfi i almindelighed. Synopsen, som eleven skal udarbejde til eksamen, er med til at tydeliggøre for eleven, at det er vigtigt under programmeringsfasen at føre notater/logbog over de processer, de gennemarbejder.

2.3 Supplerende stof

"Eleverne vil ikke kunne opfylde de faglige mål alene ved hjælp af kernestoffet."
I forhold til de faglige samspil med de øvrige fag i uddannelsen vælges der supplerende stof med henblik på at bibringe faglig fordybelse og styrke toningen af kernestoffet. Dele af det supplerende stof vælges i samarbejde med eleverne, når det er muligt"

Det supplerende stof skal opfattes som fordybelsesområder, der ligger i forlængelse af den øvrige undervisning. Eksempelvis kan nævnes programmering af en mobilapp, et spil, en webapplikation, en robot, en databaseapplikation eller et Internet of Things produkt.

En anden mulighed er at gå i dybden med mere avancerede konstruktioner i det valgte programmeringssprog såsom netværksprogrammering, parallel programmering eller lade eleverne implementere mere komplekse strukturer (såsom grafer, prioritetskøer, hash funktioner eller multidimensionale datastrukturer).

Programmering har mange muligheder for at indgå i tværfaglige sammenhænge med forskellige fag, hvor der naturligt kan indgå programmeringsprocesser.

Det faglige samspil i studieretningen kan tilgodeses også gennem valg af supplerende stof i programmeringsfaget. Faget er et studieretningsfag, og der er en lang række gode samspilsmuligheder med matematik. Rekursive funktioner, funktionel programmering og mængdelære, dataanalyse og statistik, numerisk analyse, modellering af funktioner og implementering af regression er blot nogle få eksempler på en lang række af temaer og emner, hvor programmering og matematik kan supplere hinanden. Udviklingen inden for it går hurtigt og mange nye områder ser dagens lys hvert år. Mobiltelefoner, netværksteknologier, 3D mv. er nogle af disse. I den udstrækning der er praktisk mulighed for det, kan det supplerende stof også kombineres med studiebesøg hos it-virksomheder, laboratorier og videregående uddannelsesinstitutioner.

2.4 Omfang

"Det forventede omfang af fagligt stof er normalt svarende til 200 - 350 sider"

Fagligt stof i faget omfatter alt fra netbaserede tutorials (herunder video-tutorials), netbaserede udviklingsværktøjer, biblioteksmoduler, dokumentationer og vejledninger, i- og e-bøger og traditionelle undervisningsmaterialer i form af bøger, udleveret tekst materiale mm.

Omfanget af fagligt stof anføres i beskrivelsen af den gennemførte undervisning (undervisningsbeskrivelsen), der færdigredigeres ved afslutningen af undervisningen i det enkelte fag. Omfanget angives normalt med en sådan detaljeringsgrad, så det af undervisningsbeskrivelsen fremgår, hvorledes det faglige stof har været vægtet i undervisningsforløbet. Dette kan fx ske ved at angive et skønsmæssigt sidetal eller en procentvis fordeling af stoffet.

Det er ikke meningen at disse eksempler på netbaserede materialer skal konverteres til konkrete sidetal, men der skal være en sammenhæng mellem studieplanens angivelse af fagligt bearbejdet stof og det forventede omfang på 200 - 350 sider.

Opgivelsen af omfang har til formål at sikre den faglige kvalitet, så eleverne hverken under- eller overbelastes fagligt. Der kan være stor forskel på sværhedsgraden af materialerne. Derfor er der tale om en kvalificeret vurdering på baggrund af omfang og sværhedsgrad, når sidetal optælles. Er der store niveauforskellige i klassen, er det muligt at give ekstra materialer til de elever, der udviser særlig talent eller overskud.

Det kan være en god øvelse at overveje elevtiden til forberedelse af et 2 siders dokument med tekst, koder, modeller osv. sammenholdt med en forberedelse af eksempelvis en videotutorial over samme tema og faglige indhold.

3 Tilrettelæggelse

Som studieretningsfag indgår faget på linje med de øvrige fag i planlægningen af den samlede studieretning, og, som nærmere uddybet i afsnit 3.4, er der adskillige muligheder for at faget kan virke katalyserende for undervisningen i flere fag.

Som valgfag er de tværfaglige samarbejdsmuligheder af naturlige grunde færre, men en differentieret undervisningsplanlægning kan gøre det muligt at finde tværfaglige samarbejdspartnere. Endelig er formaliseret tværfagligt samarbejde ikke altid en nødvendig forudsætning for, at den enkelte elev kan erkende et samspil mellem fagene. Gennem undervisningsdifferentiering kan eleven, eller en gruppe af elever, arbejde med tværfaglige opgaver og projekter. Som oplagte eksempler på tværfaglige samarbejder nævnes teknikfag ift. programmering af mikrocontrollere eller i matematik ift. numeriske beregninger og matematisk modellering.

3.1 Didaktiske principper

- "Undervisningen tilrettelægges ved brug af anerkendte didaktiske principper, herunder 'use-modify-create'-progression fra at anvende udleverede programmer til at modificere disse for til sidst selvstændigt at skabe (nye dele af) programmer; 'Stepwise Improvement', som teknik til trinvis, iterativ og systematisk udvikling af programmer og 'Worked Examples' (kombineret med 'faded guidance'), til illustration af eksemplariske løsningsprocesser."

Læreplanen udelukker ikke anvendelse af andre didaktiske tilgange til programmeringsundervisningen.

Stepwise improvement (fig. 1) er et eksempel på en didaktisk- og metodisk tilgang til arbejdet med programmering. For alle projektforløb gælder at selve processen med fordel kan brydes ned i flere enkeltelementer, i starten med en høj grad af lærerstyrede elevarbejder med gennemprøvede eksempler (vejledninger, tutorials mm), Worked Examples (WE).

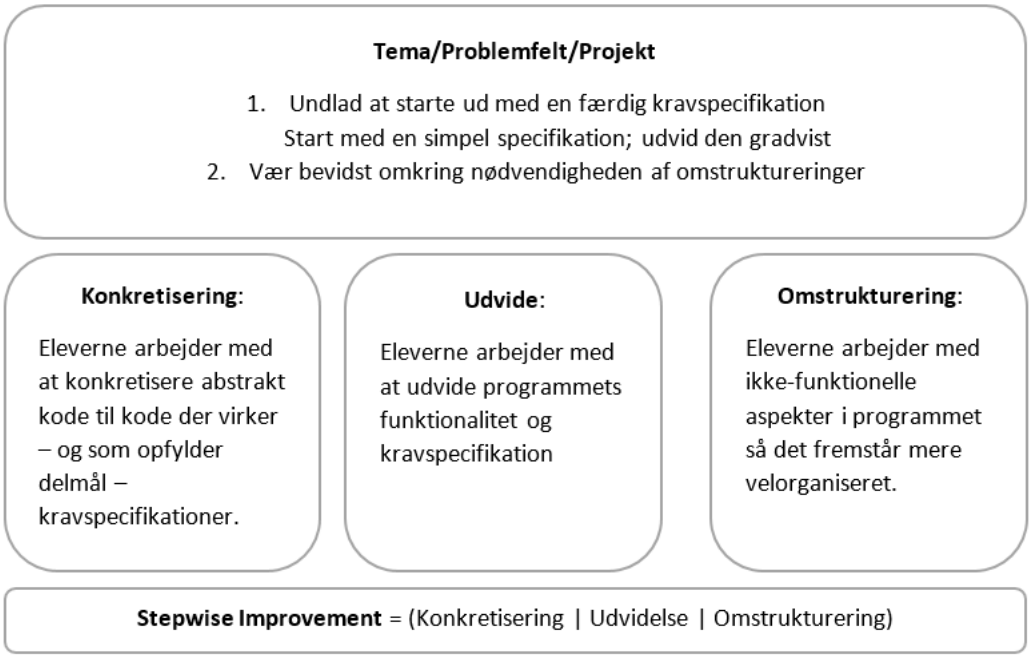

Fig 1

Eleverne skal gradvist kunne overtage processen med egen produktion (fig. 2), dels gennem forbedring og løsning af konkrete delopgaver i deres projekt med basis i de gennemprøvede eksempler (WE), dels gennem arbejdet med at udvide kravspecifikationerne til produktet (udvide) og til den færdige produktion (omstrukturere).
Modellen kan bruges som et planlægningsværktøj til, hvordan man kommer fra A til B til C, og som sådan er modellen ret lavpraktisk. I stedet for at gå mere eller mindre tilfældigt frem mod et færdigt program, kan eleverne bevæge sig mere systematisk i 3 dimensioner ved dels at forbedre deres eksisterende programmer (f.eks. rette fejl), eller udvide dem (tilføje mere funktionalitet) eller omstrukturere (dvs. ændre på strukturen i deres programmer).

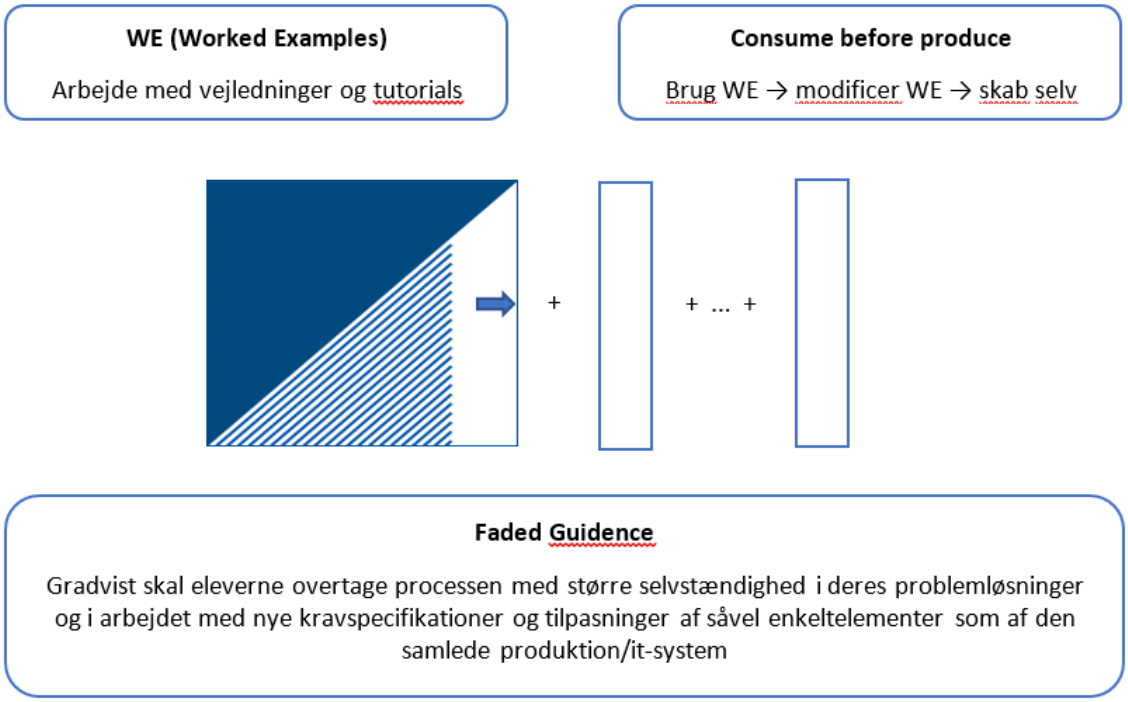

Fig 2

Disse didaktiske principper er også velegnede for elever, der ikke har programmeringsmæssige færdigheder med sig fra grundskolen, og understøtter dermed overgangen fra grundskole til gym-nasium.

- "Undervisningsformen differentieres således, at alle elever udvikler sig i undervisningsforløbet. Der veksles mellem overbliksskabende forløb, eksperimenter, øvelser og projekter."

Erfaringsmæssigt har eleverne vidt forskellige programmeringsmæssige forudsætninger ved starten af forløbet, og undervisnings-differentiering er et vigtigt redskab til at fastholde en tilstrækkelig individuel progression. Differentieringen kan eksempelvis ske gennem udstrakt inddragelse af eleverne i undervisningen gennem valg af problemstillinger, opgaver, eksempler, elevoplæg mv. Elever med erfaring kan udnyttes som en vigtig ressource for undervisningen i programmering

3.2 Arbejdsformer

- "Undervisningen tager udgangspunkt i teknologi, der indeholder programmerede funktioner. Der lægges vægt på, at eleven kan beskrive programmers funktion i normalt sprog og opnå en naturlig tilgangsvinkel til at omsætte disse funktioner til elementer i et programmeringssprog."

De projekter, som faget påtager sig, kan med fordel relatere sig til elevernes hverdag. Særligt kan det være en fordel at inddrage programmerbare teknologier, hvor eleverne (særligt i starten) kan gøre det abstrakte ved computeren mere fysisk.

Eleverne skal ret hurtigt vænne sig til at arbejde ud fra kravspecifikationer, fx. i stigende kompleksitet. At arbejde ud fra en kravspecifikation er en proces, der understøtter en kvalificering af elevens karrierekompetence, idet de bl.a. vil få lettere ved at forholde sig til forholdet mellem udvikler og kunde.

- "Der vælges et primært programmeringssprog som grundlag for undervisningen. Der arbejdes både med eksempler på større it-systemer og aktuelle teknologier, der er velkendte for eleverne. Arbejdet med aktuelle teknologier skal sætte eleverne i stand til at reflektere over egne evner og interesse for karriere indenfor programmering eller andre fagområder, hvor programmeringskompetencen er relevant"

I arbejdet med programmering anbefales det at indskrænke antallet af programmeringssprog kraftigt. Dette gøres for at skærpe elevernes kompetencer inden for et enkelt sprog og samtidig reducere kompleksiteten for den enkelte elev. I nogle tilfælde oplever vi dog, at eleverne kan have gavn af at perspektivere deres viden til et nyt sprog, og herved opnå en bedre kompetence inden for det oprindelige sprog. Det er derfor ikke et krav, at man holder sig til ét sprog.

Relevansen af de emner, der arbejdes med, er meget væsentlig på dette niveau. Det er vigtigt, at eleverne kan koble deres viden til større it-systemer. Både fordi eleverne ofte har en forforståelse, det kan være interessant at inddrage, men også for at aktualisere deres viden. Når eleverne arbejder med disse systemer, bliver de i stand til at reflektere over, hvorledes deres viden og evner passer ind i en større sammenhæng, og det er med til at skærpe deres karrierekompetencer.

- "Undervisningen tilrettelægges om muligt med udadrettede aktiviteter og/eller i samarbejde med eksterne parter, som eksemplificerer fagets anvendelses - og karrieremuligheder"

Foruden at være et digitalt dannende fag, er programmeringsfaget også meget erhvervsorienteret. Faget sikrer, at den studerende både får tekniske kompetencer og kompetence til at formidle software til forskellige målgrupper. Begge kompetencer er nødvendige i en karriere inden for softwareudvikling, hvad enten man selv er udvikler eller skal arbejde sammen med udviklere generelt. For at give den studerende en realistisk forventning til, hvordan programmering indgår i erhvervslivet (både teknisk og formidlingsmæssigt), er det vigtigt, at de projekter, der laves, retter sig ud mod samfundet. Det er ikke et krav, at der skal være samarbejde med en erhvervspartner, men det vil give en meget klar profil til faget samtidig med, at det vil skabe nogle stærke samspilsmuligheder med fx. teknik, teknologi eller kommunikation og it.

- "Undervisningen differentieres og veksler mellem overbliksskabende forløb og projektundervisning. Undervisningsformen fremmer en progression i både indholdsmæssig sværhedsgrad og selvstændighed i problemløsningen. Der udarbejdes projektdokumentation, i form af passende faglige dokumentationsformer"

Det er hensigtsmæssigt, at de overbliksskabende forløb ikke blot er oplæg fra undervisere. Her kan det tilstræbes, at eleverne ikke kun introduceres for nye metoder og værktøjer, men også får muligt for at arbejde med dem. Metoden kan lægge sig op ad den der beskrives i didaktik afsnittet omkring, "Use-Modify-Create". Det vil sige, at eleverne bevæger sig fra et redegørende- og modificerende niveau over til selv at kunne implementere førnævnte metoder og værktøjer. Der bliver dermed tale om en progression i arbejdsformen, hvor eleverne hele tiden veksler mellem erfaringsdannelse med metoder, over til selv at implementere disse i egne projekter.

Endvidere dokumenterer eleverne løbende deres arbejde, både i form af en logbog (se afsnittet: "Den enkelte elev dokumenterer[...]") og synopser (se afsnittet: "I den afsluttende periode[...]"). I begge dokumentationsformer vil det være relevant for eleverne at inddrage relevante metoder og modeller (såsom pseudokode, flowdiagrammer og UML diagrammer), der kan være med til at formidle såvel deres koncept som deres tekniske løsning.

- "Undervisningen tilrettelægges således at eleverne arbejder udforskende og kreativt og samtidig systematisk med problemløsning og programmering"

Faget lægger op til, at eleverne skal være nysgerrige på, hvorledes forskellige løsninger kan implementeres. Denne nysgerrighed kan komme til udtryk ved at prøve sig frem og nogen gange finde på løsninger, der ikke lige umiddelbart lå først for. Når resultatet er nået, kan der efterfølgende lettere arbejdes med en effektivisering og strukturering af ens kode; på den måde skabes der en systematik i det kreative og udforskende arbejde.

Det er vigtigt at have en åben dialog med eleverne om brug af generativ kunstig intelligens og understøtte dem til hensigtsmæssig brug af generativ kunstig intelligens, så der forekommer læring. Dette kunne fx være brug af generativ kunstig intelligens som redskab til at forstå kode, at debugge kode, at sparre med om emner for at øge forståelsen eller at få hjælp til at løse en opgave uden at svaret bliver generet. Det kunne også være til at opnå domænekendskab ved forundersøgelser til databasemodellering eller udvikling af interaktionsdesign. Det anbefales desuden at eleverne reflekterer over hvad man opnår, ved brug af disse værktøjer, samt at eleverne er klar over, hvad der ikke er hensigtsmæssig brug af disse værktøjer. Det er fx ikke hensigtsmæssig brug at genere hele eller dele af opgavebesvarelser eller at programmere med autokorrektur med indbygget generativ kunstig intelligens som fx Github Copilot.

Der gælder ved brug af disse værktøjer lige som når de finder kodestumper på nettet, at eleverne skal sætte sig grundigt ind i den kode, de gør brug af, såvel som de skal kildehenvise.

- "I den afsluttende periode af undervisningen afsættes 20 timers undervisningstid til, at eleverne med vejledning fra læreren udarbejder et eksamensprojekt i grupper på to til tre. Hvor dette ikke er muligt eller ønskeligt, kan man lade eleverne arbejde individuelt. Eksamensprojektet består af et produkt og en synopsis. Synopsen skal dokumentere udviklingen af det færdige produkt og har et omfang på 5-8 normalsider, eksklusiv koder, rutediagrammer, bilag mm"

For nogle elever er det meget naturligt at indgå i gruppearbejde, når der er tale om udvikling af programmer, mens det for andre kan være en meget vanskelig opgave. Beslutningen om at gå i grupper kan derfor nøje justeres efter dialog med underviseren, så man sikrer, at alle eleverne i en gruppe kan (og vil) bidrage til softwaren. Dette er bl.a. på grund af den stigende grad af kompleksitet i projekterne. Når eleverne vælger arbejdsgrupper, er det hensigtsmæssigt, at de er opmærksomme på at få delt opgaverne ligeligt, så hvert gruppemedlem er klar over, hvordan vedkommende skal bidrage til softwareløsningen.

Det er vigtigt, at eleverne lærer arbejdsmetoder til at arbejde sammen i grupper, fx. at bruge repositories, fx. direkte via Unity3D Cloud eller via et GIT system. På C-niveau er behovet for introduktionen til GIT ikke så nødvendigt som på B-niveau (igen grundet kompleksiteten), men det kan i mange tilfælde gøre elevernes arbejde meget lettere. Bl.a. kan introduktion til GIT være meget givtigt, hvis eleverne ønsker at gøre brug af deres programmeringskompetencer i andre fag hvor der arbejdes projektorienteret.

En synopsis skal forstås som en tekst, der skrives til eksaminator og censor som forberedelse til den mundtlige eksamen. Det vil sige, at en synopsis giver censor og eksaminator et overblik over projektet i form af relevante abstrakte dokumentationsformer og passende forklaringer i almindelig tekst.
I synopsen inddrages diagrammer og andre visualiseringer, der kan billedliggøre elevernes tanker om struktureringen af deres software. Endvidere gennemgås udvalgte dele af softwaren på kode niveau.

Det anbefales at stille et eksempel på opbygningen af en synopsis til rådighed for eleverne. En sådan eksemplarisk synopsis skal ikke omfatte for komplekse programmer, men gerne dække de dele man normalt ønsker at få dokumenteret i en synopsis så som:

- Forblad
- Kort abstract (censor kan herved orientere sig om opgavens indhold)
- Problemformulering
- Funktionsbeskrivelse (skærmlayout, indtastningsmuligheder, funktionalitet – alt efter hvad det er for et program)
- Dokumentation af selve programmet (overordnet beskrivelse af programmet, detaljeret dokumentation af dele af programmet (flowchart, pseudokode), variabler, objekter, events, igen meget afhængigt af hvad det er for et program)
- Test af programmet
- Konklusion
- Bilag (det er godt at få koden placeret i bilag, da den muligvis ikke ville kunne indeholdes indenfor synopsens 5-8 sider).

 "Eksamensprojektet udarbejdes inden for rammerne af et projektoplæg stillet af skolen. Eksamensprojektgrupperne udarbejder en fælles projektbeskrivelse, der inkluderer en beskrivelse af den enkelte eksaminands fokus. Projektbeskrivelsen godkendes af skolen, når beskrivelsen er tilstrækkelig fagligt bred og niveaumæssigt relevant"

I eksamensprojektet vælger eleverne deres problemstilling ud fra det projektoplæg som underviseren udarbejder. Projektoplægget skal være klart formuleret, og give en grundig beskrivelse af et problemfelt. Det er endvidere en god ide, at oplægget har en sådan karakter at eleverne kan relatere sig til det, både indholdsmæssigt og fagteknisk. Et grundigt formuleret oplæg er også så fleksibelt at alle de forskellige elevers kompetencer kan komme i spil. Det er vigtigt, at man igennem hele fagets forløb sikre en gradvis større selvstændighed i elevernes arbejde, således at de bliver i stand til at arbejde selvstændigt med eksamensprojektet. Derfor skal eleverne inden eksamensprojektet have afprøvet projektarbejdsformen, eksempelvis gennem flere forudgående projekter af mindre omfang med god mulighed for feedback og erfaringsudveksling mellem lærer og elev, og eleverne imellem.

Husk endvidere, at det i gruppernes projektbeskrivelser skal fremgå, hvilket ansvar den enkelte eksaminand har i projektet. Dette kan kontrolleres i forbindelse med godkendelsesprocessen.

 "Eksamensprojektets synopsis er individuelt udarbejdet.
Afleveringstidspunktet skal normalt være senest en uge før eksamensperiodens begyndelse"

Synopsen der omtales er en skriftlig opgave hvor eksaminandens problemstilling opridses og hvor hele processen fra planlægning over design til udvikling og afprøvning dokumenteres. I synopsen inddrages der diagrammer og andre visualiseringer der kan billedliggøre elevernes tanker om strukturering af deres software. Endvidere gennemgås udvalgte dele af softwaren på kode niveau.

 "Eksamensprojektet indgår i grundlaget for den afsluttende standpunktskarakter, hvis der gives en sådan, og udgør grundlaget for prøven. Eksamensprojektets synopsis er forinden prøven ikke rettet og kommenteret af eksaminator."

Det er vigtigt, at man sammen med administrationen på skolen sikrer, at der tid nok ved afslutningen af eksamensprojektet til at inddrage projekterne i årskaraktererne. Samtidig må man naturligvis ikke give eleverne feedback på synopsis og projekt inden prøven.

 "Den enkelte elev dokumenterer løbende sin faglige udvikling i en logbog. Dokumentationen i logbogen kan have form af f.eks. programmer, noter, synopser, programbeskrivelser og flowcharts."

Der skal inden eksamen udfærdiges en logbog. Den laves undervejs, og der laves eventuelt en portfolio i slutningen.

Logbogen er en refleksionsopgave, hvor man reflekterer over egen læring. Den kan således have tilknyttet et refleksionsresume for hvert forløb, hvor eleven gør sig nogle overvejelser om den læring der er opstået. Logbogen skal endvidere være et sted hvor elever og lærer kan have adgang til den. Det kan være på et fælles drev (fx. OneDrive, GoogleDrive, et skoledrev el.)

Man kan også forestille sig at eleverne dokumentere deres arbejde via GIT, fx. GITHub.

3.2.1 Skriftlighed og mundtlighed i faget

For eleverne findes der overordnet set to typer af skriftligt arbejde. Den løbende logbogsføring og refleksionen, der grundlæggende er rettet mod eleven selv, og så er der det formidlingsmæssige arbejde (synopsen).

Det mundtlige aspekt er sidestillet med det skriftlige, dels fordi den mundtlige præstation indgår som en del af bedømmelsesgrundlaget til eksamen, og dels fordi italesættelsen af fagets mange elementer er vigtig. Det mundtlige aspekt kan f.x. trænes ved at lade eleverne fortælle hinanden om udvalgte kodedele og sammenhæng mellem kode og hvad man ser, ligesom man kan lade eleverne tale indbyrdes om deres kommentarer til koden.

Den skriftlige formidlingskompetence er meget tæt knyttet til elevernes karrierekompetencer, idet de videre i deres karrierer vil kunne drage nytte af at kunne formidle software på flere forskellige niveauer (teknisk) til forskellige målgrupper.

3.3 It

- "Gennem arbejdet med udvikling af programmer i faget opnås såvel specifikke faglige digitale kompetencer som almene digitale kompetencer, hvilket er fagets bidrag til uddannelsernes overordnede krav om digital dannelse".

I faget vil eleverne stifte bekendtskab med den tekniske side af de systemer, de bruger i deres dagligdag. Indsigten i, hvordan deres programmer fra hverdagen er skruet sammen, er med til at give dem en grundlæggende forståelse for opbygningen af programmer. Denne indsigt kan de bruge til at generalisere erfaringer, når de anvender andre systemer, og samtidig når de selv udvikler nye programmer.

- "Programmeringsværktøjer, der automatisk kan generere dokumentation og test, anvendes, ligesom andre informationsteknologiske værktøjer inddrages efter behov."

Her er der tale om udviklingsmiljøer (såsom Microsofts Visual Studio) eller versioneringssystemer (såsom Git)

- "Internettet anvendes som søgningsværktøj til oplysninger, vejledninger, eksempler, programdele og biblioteksmoduler med efterlevelse af ophavsretslige regler og dokumentationskrav."

Særligt inden for programmeringsfeltet er det muligt at finde vejledninger og dele af programkode på internettet. Disse dele inddrager eleverne ofte i deres løsninger, og her er det væsentligt at tale med eleverne om, hvordan de gør programmerne til deres egne ved at ændre på dem.

- "I arbejdet med stof om konkrete teknologier og standardiseringer skal eleverne anvende originale kilder (eksempelvis dokumentation af programmeringssprog, data og diagrammer)."

Det er en god ide at eleverne trænes i at slå op i relevante referencer til API 'er og andre eksterne biblioteker samt dokumentere brugen heraf og i at tage udgangspunkt i flowdiagrammer, når de programmerer.

- "Eleverne arbejder med digital dokumentation af deres programmer, bl.a. i form af modeller og kommentarer i programmeringskoden"

Blandt relevante modeller kan nævnes flowdiagrammer, UML-diagrammer, pseudokode eller lignende.

I undervisningen tilstræbes en tilpas vekselvirkning mellem det analoge og det digitale. It og digitale medier og værktøjer, herunder kunstig intelligens, benyttes hvor det skønnes hensigtsmæssigt ift. elevernes læringsproces og digitale dannelse. I anvendelsen af it styrkes elevernes evne til at søge, udvælge og formidle relevant fagligt materiale samt til at forholde sig kritisk til de muligheder og begrænsninger, som digitale værktøjer, og produkter frembragt ved hjælp heraf, giver.

3.4 Samspil med andre fag

Samspilsdimensionen er meget vigtig, bl.a. fordi det indgår i fagets faglige mål og fordrer en kvalificering af elevernes innovationskompetence.

Her er det vigtigt at pointere forskellen fra C-niveauet, hvor de omtalte problemstillinger var af "enkel" karakter. På B-niveau kan der være tale om mere komplicerede problemstillinger, der ligger til grund for samspillet. Kravet om samarbejder er dermed skærpet på B-niveauet, og det er endnu vigtigere at forholde sig til. Man kan overveje at vælge problemstilling(er), der giver mulighed for at arbejde med elevernes globale kompetencer.

Skærpelsen sker, fordi en vigtig del af faget på dette niveau er, at skabe en bevidsthed hos eleverne om, hvordan man som udvikler arbejder sammen med andre fagligheder.

Løsningen af problemstilling kan gøres ske mange forskellige måder, og her er listet nogle forskellige tilgange til samspilsdimensionen, uanset hvilken uddannelse faget måtte indgå i.

<u>Konstruerende</u>
Faget giver selv sagt nogle tekniske færdigheder, der gør eleven i stand til at producere forskellige produkter. Faget kan i en samspilssituation ses som værende produktudviklende og på den måde kan det indgå i en situation hvor et andet fag analyserer produktet og evt. forsøger at markedsføre det.

<u>Analyserende</u>
I faget er der mange værktøjer til at analyse en given problemstilling. Eksempelvis kan man opstille en kravspecifikation for et scenarie, der er givet i et andet fag. Kravspecifikationen kan være et udgangspunkt for en udviklingsproces, hvor man efterfølgende forsøger at beskrive, hvorledes scenariet kan løses på software niveau (eksempelvis beskrevet diagrammatisk).

<u>Designende</u>
På B-niveauet opstår der endnu en samspilsmulighed i de faglige mål og kernestofområder der relaterer sig til brug og funktion. Her bliver det muligt at tale om hvordan brugerens oplevelse af softwaren bedst varetages når systemet skal konstrueres. Brugerinteraktion og interaktionsdesign kan på den måde fungere som samspilselementer, hvor programmeringsfaget enten bidrager til andre fags indhold eller drager nytte af de metoder eleverne anvender i disse.

De analyserende og designende tilgange kan være en fordel, da de åbner mulighed for analyse og vurdering.

4 Evaluering

4.1 Løbende evaluering

Den løbende evaluering af undervisningsforløb kan kombineres med at træne eleverne i udformning af den logbog, som de laver i forbindelse med eksamensprojektet. Her kan eleverne reflektere over deres faglige udbytte af forløbet, arbejdsprocessen og de anvendte arbejdsformer, perspektivere til andre fag, osv. Suppleret med spørgsmål fra læreren om f.eks. sværhedsgrad, motivation og relevans, kan eleverne få indflydelse på kommende projekter i undervisningen.

Det er vigtigt at synliggøre kravene for eleverne i de enkelte forløb, da de som nybegyndere ikke er bekendte med fagets niveau, og derfor ikke altid kan vurdere om en opgave er svær eller let, og i hvilken grad de opfylder målene. Dette er særlig relevant når man har en høj grad af differentiering på holdet, og i høj grad lader eleverne være med til at sætte mål for deres egen læring.

Det er naturligt at man vurderer elevernes evner til at beherske det valgte programmeringsmiljø, når man hjælper eleverne i den daglige undervisning, både i den første indlæringsfase af sprogets grundelementer, samt i de mere projektorienterede forløb.

Den afsluttende evaluering i faget er baseret på et eksamensprojekt, med et afsluttende produkt og tilhørende synopsis. Det er en god ide, at eleverne trænes i at udforme relevant dokumentation af deres arbejde, både jvf. det tilhørende kernestofområde, men også fordi de er helt uerfarne med den skriftlige diskurs i faget. Her er det vigtigt at givende løbende formativ feedback.

4.2 Prøveform

Eksaminanden skal inden eksamensperiodens begyndelse aflevere sit eksamensprojekt i form af produkt og synopsis, som eksaminator evaluerer som forberedelse til den mundtlige eksamen. Det er en betingelse for adgang til eksaminationen, at eksaminanden har afleveret sit eksamensprojekt. Synopsen er forinden prøven ikke rettet og kommenteret af læreren.

Eksaminanden afleverer to eksemplarer af synopsen. Den ene afleveres til eksaminator; den anden kan skolen evt. fremsende til censor.

Det er hensigtsmæssigt, at skolen stiller værktøj til rådighed for eleverne, så de kan opbevare og præsentere deres eksamensprojekt i elektronisk form. Eleverne skal løbende samle deres programmeringsarbejde i en portfolio, og eksamensprojektet kan indgå som en naturlig del af denne.

Afleveringsfristen for eksamensprojektets produkt og synopsis fastsættes af skolen, dog således at tidsfristen er senest en uge før eksamensperiodens begyndelse. Produkt og synopsis skal være til rådighed ved eksaminationen.

"Eksaminator og censor drøfter inden den mundtlige del af prøven, hvilke problemstillinger fra eksamensprojektet eksaminanden skal uddybe."

Det vil ofte være en fordel, at eksaminanden medbringer eget udstyr til at understøtte sin fremlæggelse. Det kan være produktet på en bærbar computer og en multimediepræsentation.

Eftersom eleverne i en projektgruppe eksamineres i det samme eksamensprojekt, bør skolen sikre, at elever, der er blevet eksamineret, og elever, der venter på at blive eksamineret, ikke kan kommunikere sammen. Det kan fx ske ved, at eksaminander, der endnu ikke er blevet eksamineret, venter i et rum uden kommunikationsudstyr.

Der er afsat ca. 30 minutter til eksaminationen. og der er ca. 60 min's forberedelsestid. Eksaminanden starter med at præsentere sit eksamensprojekt, med tilhørende teoretiske overvejelser, for eksaminator og censor. Eksaminators supplerende spørgsmål fungerer som oplæg til en uddybende samtale om de punkter, som ikke er berørt, eller der ikke fyldestgørende er redegjort for i synopsen.

For at kunne evaluere alle de faglige mål, involverer eksamensformen også at eleven trækker en ukendt opgave.

" Opgaverne, der indgår som grundlag for prøven, skal tilsammen dække de faglige mål. "

Udtryk som "du kan eventuelt komme ind på..." eller "hvis der er tid, kan du..." skal undgås, fordi de skaber usikkerhed om, hvornår opgaven er fuldt besvaret.

Det er hensigtsmæssigt, at eksaminationstiden fordeles ligeligt mellem eksamensprojektet og den ukendte opgave, og at eksaminandens præsentationer af to de dele efterlader tid til generelle spørgsmål fra censor og eksaminator.

Opgaver og materialer sendes til censor mindst 5 hverdage før prøvens afholdelse, medmindre særlige forhold er til hinder herfor. Det kan betyde, at udsendelsen må foretages, før eksamensplanen er offentliggjort. Udsendelsen af opgaver og materialer må da kun ske i et omfang, der ikke medfører, at andre dele af eksamensplanen kan udledes.

Adgang til Internettet og værktøjer med generativ kunstig intelligens:

Det fremgår af Bekendtgørelse om visse regler om prøver og eksamen i de gymnasiale uddannelser (§ 6, stk. 3), at adgang til internettet ikke er tilladt for eksaminanderne under prøverne, medmindre der for den enkelte prøve er givet adgang hertil. Der er altså som udgangspunkt ikke adgang til internettet som fagligt hjælpemiddel ved prøverne i gymnasiet.

For mundtlige prøver betyder det, at internettet som fagligt hjælpemiddel ikke er tilladt under prøven. Undtaget herfra er dog fagene informatik C, informatik B, it A, programmering C, programmering B og informatik B merkantil eux, hvor der er særlige faglige grunde, der taler for dette.

I de nævnte fag, hvor brug af internettet er tilladt som fagligt hjælpemiddel ved de mundtlige prøver, inkluderer de tilladte hjælpemidler anvendelsen af værktøjer med generativ kunstig intelligens på en måde, som afspejler, hvordan man arbejder i den daglige undervisning. Dette kunne fx være brug af ChatGPT til at debugge kode. I forhold til denne type hjælpemidler er det vigtigt at være opmærksom på, at den tilladte anvendelse er betinget af, at hjælpemidlerne kun benyttes i et omfang, hvor prøvebesvarelsen er selvstændig og udelukkende elevens egen.

Det betyder, at ved prøven i programmering B, må eksaminanderne godt bruge internettet som fagligt hjælpemiddel i forberedelsestiden. Eleverne har også adgang til internettet som fagligt hjælpemiddel, når de udarbejder eksamensprojektet.
Hvis adgang til internettet i programmering B er nødvendig for eksaminandens præsentation af sit eksamensprojekt eller for eksaminandens besvarelse af den lodtrukne opgave, er dette tilladt under eksaminationen.

"Regler vedrørende eksaminandernes brug af internettet for at tilgå tilladte hjælpemidler ved prøverne fremgår af § 6 i "Bekendtgørelse om visse regler om prøver og eksamen i de gymnasiale uddannelser". I vejledningen til denne bekendtgørelse er der givet eksempler på, hvilke hjælpemidler der må, og hvilke der ikke må tilgås via internettet."

4.3 Bedømmelseskriterier

Med internettets muligheder for at finde mange gode programmer og programdele, skal det fremgå klart, hvilke dele af programmet eleven selv har fremstillet, og hvornår der er tale om andres programdele og biblioteksmoduler. Oprindelsen skal fremgå af synopsen, og eleven skal kunne dokumentere hvordan de bruges.

I tvivlstilfælde kan man bede eleven om at ændre en lille smule på funktionaliteten af programmet. Det kan ligeledes være relevant at bede eleven redegøre for udvalgte kontrolstrukturer i programmet, og argumentere for valget af netop disse frem for andre af eleven kendte kontrolstrukturer.
Det kan også være afklarende at spørge om hvordan programmet er blevet udviklet – hvad har eleven startet med at løse, og hvordan er opbygningen ellers sket. Det kan også give et godt billede af elevens evner, at få oplyst hvilke problemer, der har været i udviklingen af programmet. Især for de dygtige elever viser det noget om deres kompetencer inden for programmering, hvis de kan give forslag til hvordan problemstillingen ellers kunne have været løst.

Bedømmelseskriterierne omfatter både elevens præsentation af eksamensprojektet og besvarelse af den opgave der er givet ved forberedelsen, altså to ting:

1. Eksamensprojektet
2. Opgavebesvarelsen fra forberedelsen

- "Ved prøve, hvor faget indgår i samspil med andre fag, lægges der vægt på at eksaminanden
    - kan demonstrere viden om fagets identitet og metoder
    - behandle problemstillinger i samspil med andre fag"

Hvis en eksaminand eksempelvis har lavet et projekt i samspil med teknikfaget hvor eksaminanden programmerer en mikrocontroller, er det hensigtsmæssigt at, der indgår forklaring af de el tekniske begreber og fænomener der er relevante for projektet (det kunne være et begreb som pulsbreddemodulation el.)

4.3.1 Oversigt over karakterskalaen

Karakter	Betegnelse	Beskrivelse
12	Fremragende	Karakteren 12 gives for den fremragende præstation, der demonstrerer udtømmende opfyldelse af fagets mål, med ingen eller få uvæsentlige mangler.

Karakter	Betegnelse	Beskrivelse
7	God	Karakteren 7 gives for den gode præstation, der demonstrerer opfyldelse af fagets mål, med en del mangler.
02	Tilstrækkelig	Karakteren 02 gives for den tilstrækkelige præstation, der demonstrerer den minimalt acceptable grad af opfyldelse af fagets mål.

Nedenstående er vist en vejledende karakterbeskrivelse for Programmering B valgfag for karaktererne 12, 7 og 02.

Beskrivelsen er udarbejdet med udgangspunkt i læreplanens faglige mål og bedømmelseskriterier

Karakter	Beskrivelse	Programmering B valgfag
12	Fremragende	Eksamensprojektet præsenteres glimrende og fagligt sikkert mht. planlægning, gennemførsel og evaluering. Eksamensprojektet lever op til de stillede krav med kun få uvæsentlige mangler. Der argumenteres fagligt velbegrundet for valg af programmeringsløsninger. Eksamensopgaven præsenteres glimrende og fagligt sikkert, og lever op til de stillede krav med kun få uvæsentlige mangler. Eksaminanden perspektiverer fagligt kvalificeret sin eksamensopgave til såvel egne it-produkter som til opgavens teoretiske indhold. Eksaminanden besvarer glimrende og fagligt sikkert uddybende og supplerende spørgsmål under samtalen.
7	God	Eksamensprojektet præsenteres mht. planlægning, gennemførsel og evaluering. Eksamensprojektet lever med en del mangler op til de stillede krav. Der argumenteres for valg af programmeringsløsninger. Eksamensopgaven præsenteres og lever med en del mangler op til de stillede krav. Eksaminanden perspektiverer sin viden til såvel egne it-produkter som til opgavens teoretiske indhold. Eksaminanden besvarer uddybende og supplerende spørgsmål under samtalen.

Karakter	Beskrivelse	Programmering B valgfag
02	Tilstrækkelig	Eksamensprojektet præsenteres sparsomt og knapt mht. planlægning, gennemførsel og evaluering. Eksamensprojektet lever minimalt acceptabelt op til de stillede krav. Der argumenteres minimalt acceptabelt for valg af programmeringsløsninger. Eksamensopgaven præsenteres sparsomt og knapt, og lever minimalt acceptabelt op til de stillede krav. Eksaminanden perspektiverer sparsomt og knapt sin viden til såvel egne it-produkter som til opgavens teoretiske indhold. Eksaminanden besvarer tilstrækkeligt sparsomt og knapt uddybende på supplerende spørgsmål under samtalen.

BØRNE- OG
UNDERVISNINGSMINISTERIET
STYRELSEN FOR
UNDERVISNING OG KVALITET