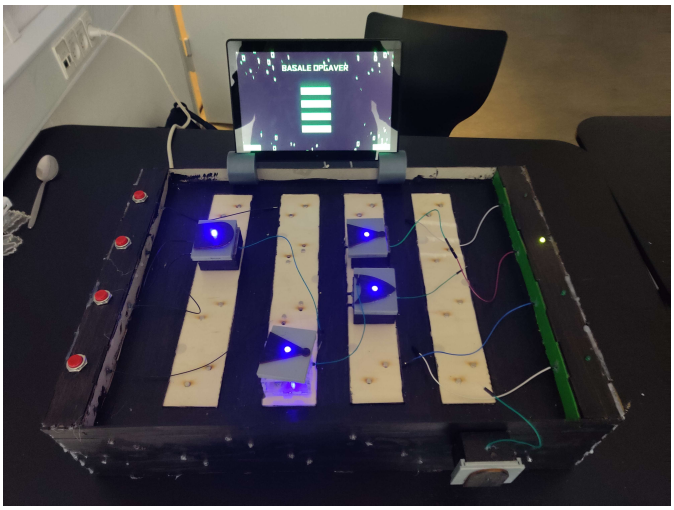
Udvikling af digital opgavesystem til fysisk læringsdisplay om digital logik

Carl Benjamin S. Dreyer

19. maj 2024



Informatik Eksamensprojekt

Klasse: 3.T

Vejleder: Daniel Withenstein

Fag: Informatik B (valgfag)

Skole: NEXT Københavns Mediegymnasie

Dato: 19. maj 2024

Indhold

1 Indledning

Dette projekt bliver udviklet i samarbejde med eksamensprojektet i teknikfag – Digital Design og Udvikling. Dette projekt fokuserer specifikt på at udvikle en IT-løsning til at interagere med et fysisk produkt. Teknikfags produktet er et digitalt læringsdisplay til at undervise computer science elever på gymnasium om logiske porte (logic gates), og hvordan de anvendes indenfor digital logik. Til dette udvikles der et fysisk produkt i teknikfaget, som fungerer som en slags legeplads for at bygge logiske kredsløb for at undersøge hvordan nogle af de mest basale logiske porte fungerer. IT-løsningen, som udarbejdes i dette projekt, skal udvide det fysiske produkt og inkorporere et opgavesystem, hvor brugere kan teste deres viden indenfor emnet ved at klare opgaver og se reel tid status på hvordan ens logiske kredsløb reagere til forskellige inputs. Opgavesystemet er en app, som skal udvikles til en tablet der kan monteres til det fysiske produkt.

2 Teori

Det følgende afsnit forklarer den basisviden, som er nødvendig for at forstå projektet ordentligt. Jeg antager at læseren har kendskab til basalt programmering og programmeringsprincipper som bitwise operatorer, asynkront kontrolflow, basalt viden omkring Unity og kendskab til programmeringsprogene C# og C. Derudover antages det at læseren har kendskab til udgiv abonner netværksprotokollen MQTT. Dette er da forklaring af disse emner er ude for omfanget af denne rapport.

2.1 Logiske operatorer og porte

Logiske operatorer arbejder med individuelle bits. En AND operator er en binær logisk operator som har to operander. Resultatet af AND operationen er et enkelt bit. Hvis begge operander (inputs) i AND operationen er logisk højre (1), så er resultatet også logisk højt (1). Derimod, hvis en eller begge af operanderne i AND operationen er logisk lave (0), så er resultatet også logisk lavt (0). For hver logisk operator kan der laves en såkaldt sandhedstabel, som beskriver hvordan resultatet afhænger af de forskellige operander (inputs). For AND opratoren kan man f.eks. lave følgende sandhedstabel:

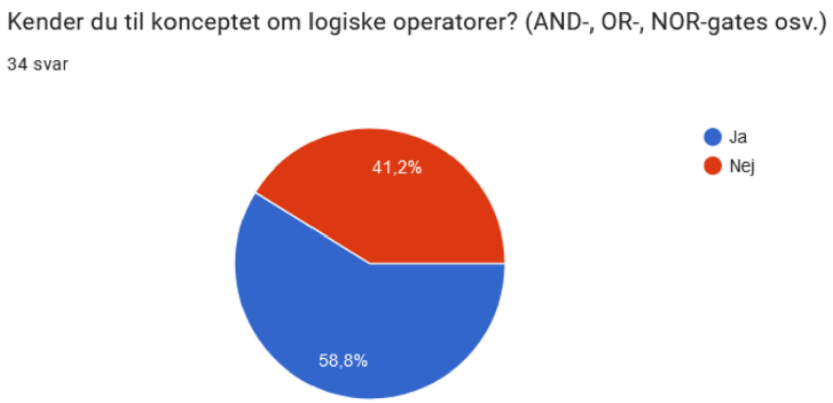
A	B	$A \cdot B$
0	0	0
1	0	0
0	1	0
1	1	1

Tabel 1: Sandhedstabel for AND operator

Her er der en kolonne for hver operand (A og B), og en kolonne for resultatet af AND operationen. Indenfor boolsk algebra skrives AND operationen, som et gange tegn.
Disse logiske operatorer kan blive anvendt indenfor elektroniske kredsløb, som logiske porte. En logisk port er et elektronisk kredsløb, der opfylder en sandhedstabel, som f.eks. AND operatoren. Logiske operatorer er typisk udviklet ud fra transistore eller MOSFETs i Resistor Transistor Logic (RTL), Transistor Transitor Logic (TTL) eller (Complementary metal-oxide-semiconductor) CMOS teknologier. For dette projekt er der blevet anvendt RTL og TTL for at bygge de logiske porte. Der vil ikke blive uddybet på hvordan de logiske porte er blevet udviklet for det fysiske produkt, da rapporten udelukkende fokuserer på det digital opgavesystem.

3 Empathize

For at kunne designe et opgavesystem, er det altafgørende at opgaverne har den rette sværhedsgrad baseret på målgruppen – computer science elever på gymnasium.
Derfor blev der udført en spørgeskema undersøgelse med formål om at udersøge målgruppens eksisterende viden indenfor emnet omkring digital logik og logiske porte og operatorer. Spørgeskemaet fik i alt 34 svar, fra 3 forskellige computer science årgange på NEXT gymnasier i København.
Det første spørgsmåls formål er at undersøge om målgruppen har hørt har kendskab til emnet:



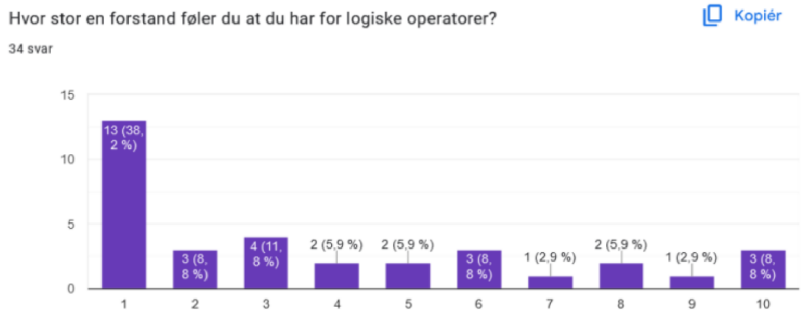
Figur 1: Spørgsmål om målgruppen har kendskab til emnet

Ud fra denne empiri, kan det ses at de fleste (58,8%), har hørt om logiske operatorer før. Dette er dog stadig en betydelig andel (41,2%), som ikke har kendskab til emnet. Det er derfor vigtigt at opgaverne forholder sig til at målgruppen potentielt ikke ved noget om emnet.
Herefter spørges ind til om, målgruppen nogensinde har anvendt deres kendskab indenfor emnet i praksis. Dette kunne f.eks. være at designe elektroniske kredsløb med logiske porte, programmere med anvendelse af logiske operatorer eller anvendt det indenfor boolsk algebra.

Figur 2: Spørgsmål om målgruppen har anvendt logiske operatorer før i praksis

Her kan det ses at der er en relativ stor gruppe af målgruppen, som aldrig har anvendt deres kendskab indenfor emnet i praksis. Dette giver en stærk indikation om at opgaverne skal være en introduktion til emnet omkring digital logik.

Herefter undersøges hvor stor forstand målgruppen har til emnet. Hvor 1 svarer til meget lidt forstand og 10 svarer til meget forstand.



Figur 3: Spørgsmål om hvor stor forstand målgruppen har til emnet

Denne empiri, bidrager til at opgavernes sværhedsgrader skal være nemme, og på et basalt niveau.

Derudover undersøges det om eleverne bliver lært om emnet i deres linjefag – programmering og informatik. Ifølge de oficielle læreplaner for STX og HTX gymnasier, logiske kredsløb eller logiske porte en del af pensum for informatik- og programmeringselever[1]. Dette hentyder til at det er første gang eleverne imødekommer emnet, og skal derfor tages højde for i designet af opgaverne.

Ud fra empirien fra spørgeskemaet og de oficielle læringsplaner for målgruppen, er der en stærk indikation til at opgaverne skal være målrettet imod nybegyndere. Dette betyder at sværhedsgraden skal være let og virke som en introduktion til emnet.

4 Define

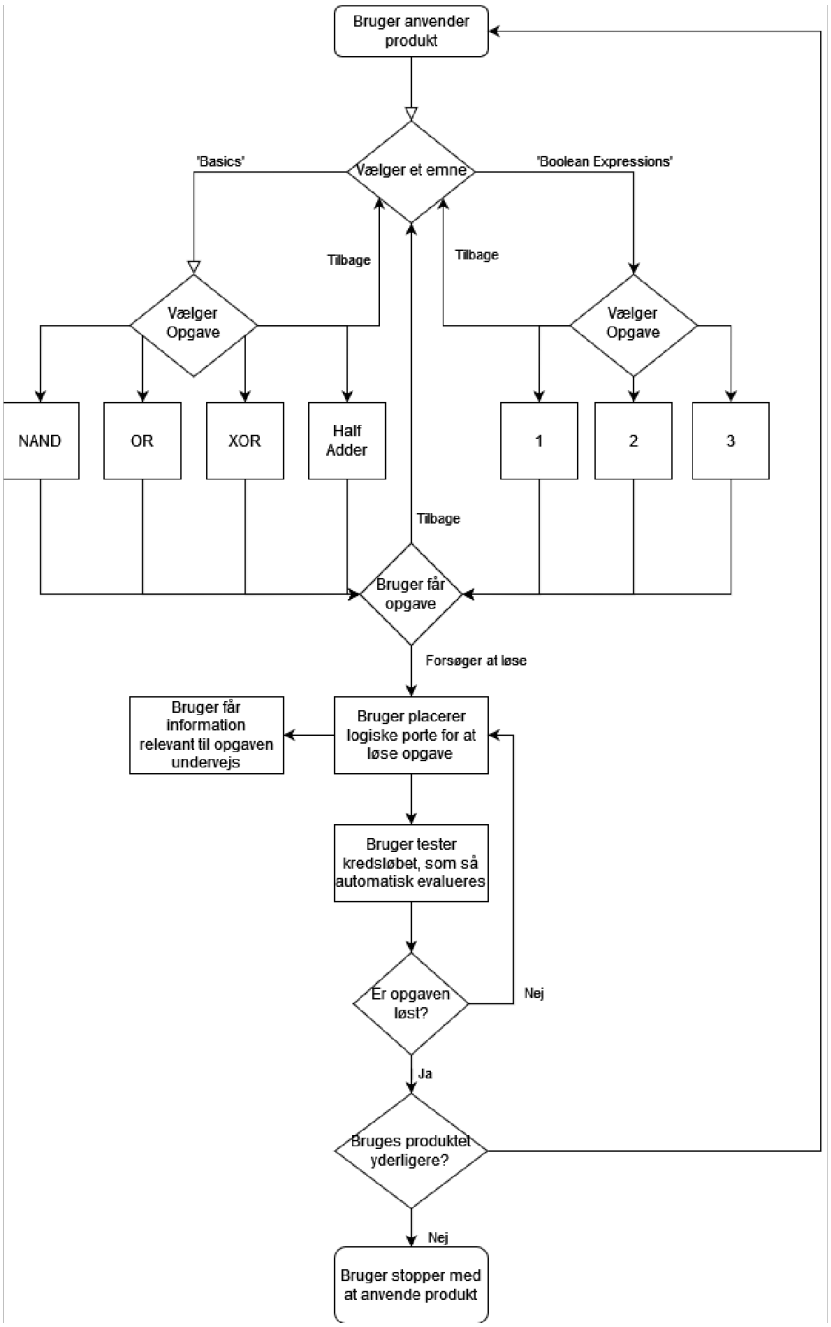
Den indsamlede empiri fra spørgeskemaundersøgelsen, medfører at opgavesystemet målrettes som en introduktion for nybegyndere. Ud fra dette, kan følgende problemformulering formuleres:

Hvordan kan man designe og udvikle et opgavesystem til et fysisk produkt omkring logiske porte, som målretter sig mod computer science elever på gymnasiet der underviser på et fundamentalt niveau?

5 Ideate

I denne fase undersøges det hvordan løsningen skal designes og implementeres på et højt niveau. Der vil blive anvendt Scrum, til udviklingen her fra. Scrum er en agil arbejdsmetode, som anvendes for at gå fra idé til produkt. I Scrum kan man hurtigt lave ændringer og tilpasninger undervejs i uviklingsprocessen[2]. Dette er især nødvendigt, når der potentielt er krav, som ændrer sig undervejs i udviklingen.
En vigtigt overvejelse er hvordan brugeren vil bruge løsningen. For dette laves et flowdiagram, som beskriver hvordan brugerens rejse gennem deres interaktion med produket:

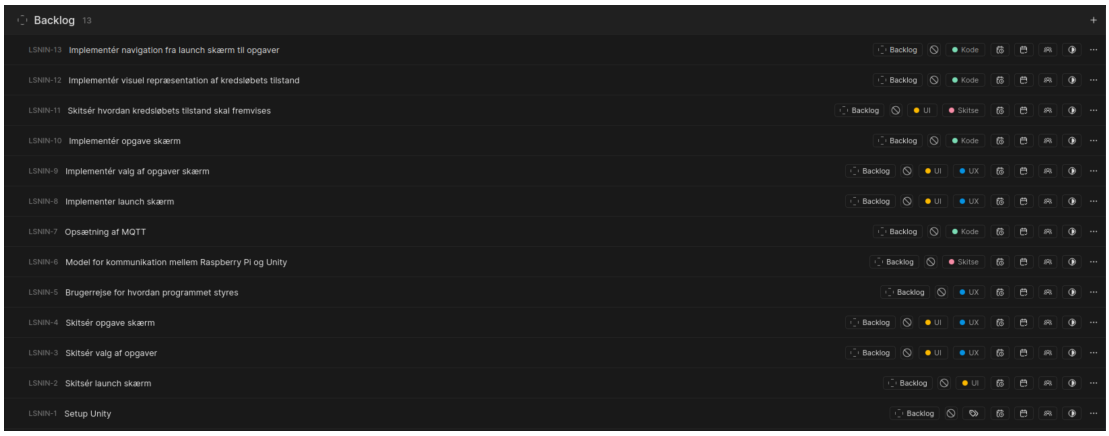
[1] Undervisnings Ministeriet 2017.
[2] Martin Damhus, Jesper Buch og Elisabeth Husum 2024, kap. 1.



Figur 4: Flowdiagram af brugerrejse

Her kan det ses hvordan det er meningen at brugeren anvender løsningen. Det starter med at brugeren vælger et emne fra launch skærmen. Dette kunne være basale opgaver som at bygge logiske kredsløb ud fra de basale logiske porte som produktet tilbyder (AND, OR, NOT og XOR). Derudover er det muligt at have et mere avanceret niveau, hvor man skal bygge et specifikt logisk kredsløb som opfylder boolske udtryk og sandhedstabeller. Det kan ses at brugeren undervejs i løsnignen af en opgave, kan anvende opgavesystemet til at få information og hints relevant for opgaven. Når brugeren mener de er færdige med opgaven, testes det logiske kredsløb, som brugeren har bygget. Hvis det er korrekt, tages de tilbage for at afprøve andre opgaver.

Ud fra denne brugerrejse, kan det ses at der skal udvikles en del ting. Dette inkludere f.eks. launch-, valg af opgave- og opgaveskærm, samt kommunikation mellem fysisk produkt og opgavesystem. For at holde styr på projektets opgaver, laves der en række af 'issues' i scrum programmet – Plane[3]:

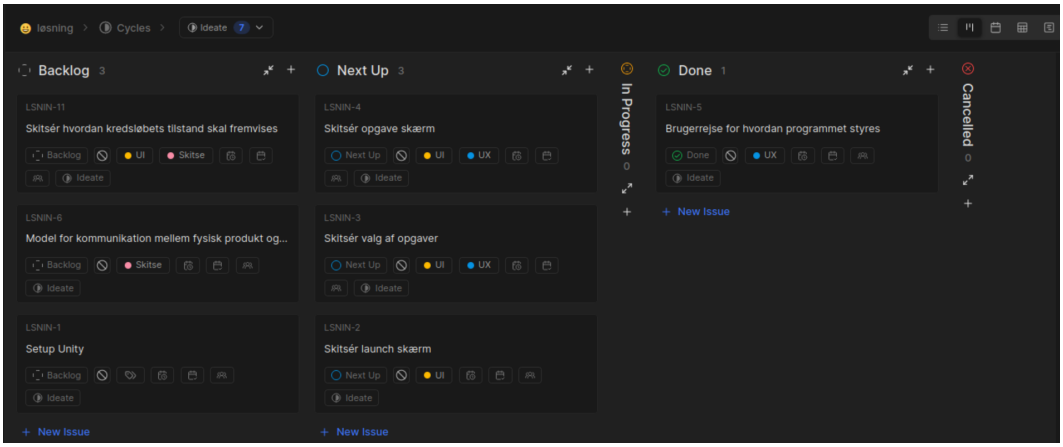


Figur 5: Liste af opgaver i scrum programmet Plane

Hver opgave har en liste af kategorier asocieret med dem. Dette er så opgaverne kan fordeles bedst ud til gruppemedlemmerne baseret på de kompetencer de har. Undervejs i udviklingen opdateres opgavens status til at være i Backlog, Next Up, In Progress eller

[3] *Plane* 2024.

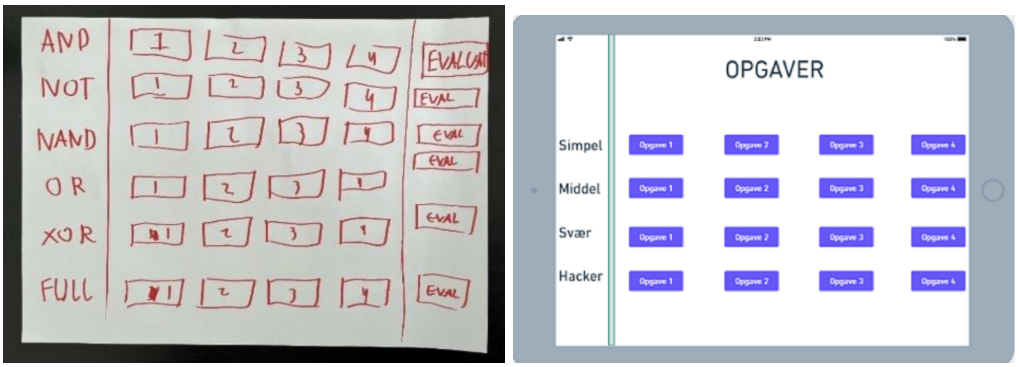
Done. På den måde er det nemmere at have et komplet overblik over udviklingen.
For at kunne nå projektets deadline med mest sikkerhed, grupperes opgaverne i sprints.
I plane bliver dette gjort ved at lave en ny 'cycle' (ækvivalent til sprint):



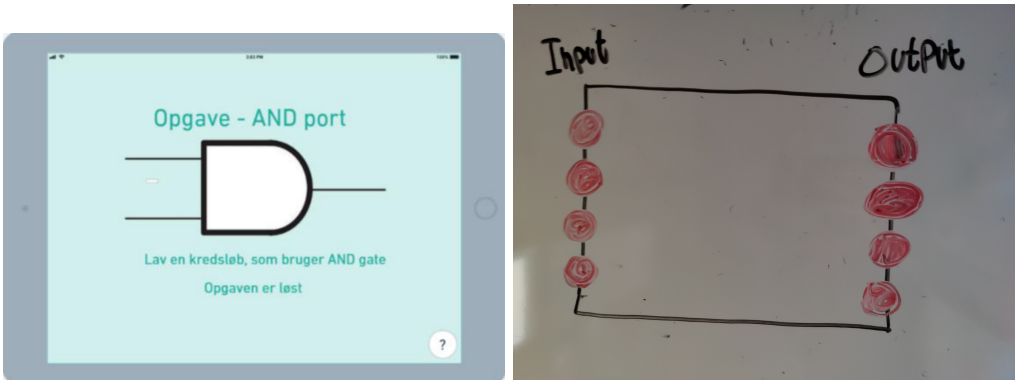
Figur 6: Projektets første sprint til idégenering- og konceptudviklingsfasen

Hver sprint har et forløb på maks. 2 uger, som afhænger af sprintets opgaver, samt hvor meget tid der kan afsættes af gruppens medlemmer til at arbejde på opgaverne. Sprintets maksimale tidsforløb, er for at sikre at der hele tiden sker fremskridt, selvom fremskridtene evt. er små, som beskrevet i det agile manifest[4].
For dette sprint, blev der tegnet skitser af opgavesystemets forskellige sider, og hvordan kredsløbets tilstand skulle blive fremvist.

[4]Manifesto 2001, princip 3.



Figur 7: Skitse af valg af opgave skærm



Figur 8: Skitse af en opgave (venstre) og hvordan kredsløbets tilstand skal fremvises (højre)

I det fysiske produkt er der 4 knapper, som fungere som input til det logiske kredsløb. Derudover er der 4 LEDer som output af det logiske kredsløb som brugeren bygger.

6 Prototypudvikling

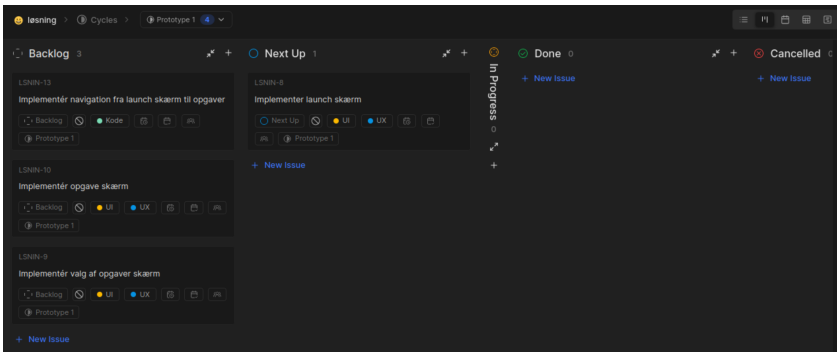
Ud fra forige afsnit, kan der laves nogle opgaver for de opkommende sprint.

Sprint	Opgaver
Prototype 1	launch-, opgave valg og opgaveskærm. Navigation mellem skærme.
Prototype 2	Tilpasninger og forbedringer til UI/UX baseret på første brugertest.
Prototype 3 (endelig)	Opsæt MQTT. Implementér statusdisplay til kredsløbets tilstand.
Prototype 4 (nåede ikke)	Evalueringsfunktionalitet til at teste om opgave er klaret.

Tabel 2: Udkast for projektets sprints

6.1 Prototype 1

Den første prototype udvikles for at teste hvordan brugerfladen føles. For dette, blev der lavet et nyt sprint med de krav som prototypen krævede:



Figur 9: Sprint af første prototype

De følgende skærme blev udviklet:

Figur 10: Brugerflade for launch skærm (venstre) og valg af opgaver

Figur 11: Brugerflade for en opgave

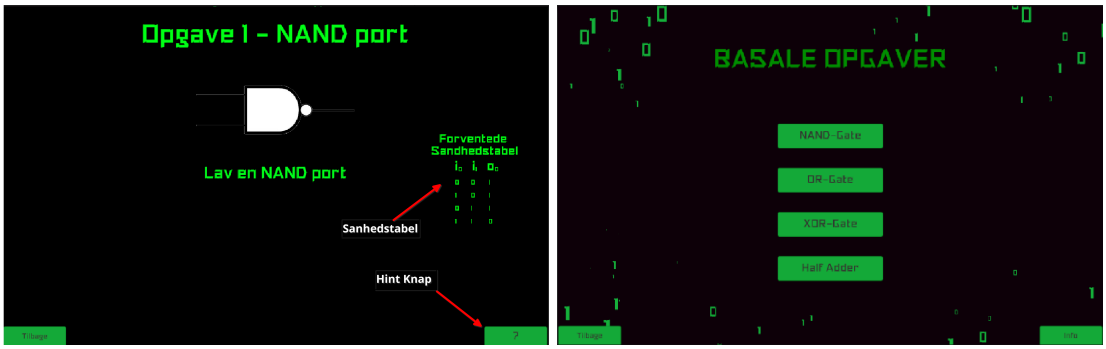
Der blev foretaget en brugertest, for at få konstruktiv feedback på brugerfladen. I empirien fra brugertesten, var brugeren forvirret om hvad hendes formål var. Derudover var hun forvirret ift. alle de opgaver der var at vælge imellem.

6.2 Prototype 2

Baseret på den indsamlede empiri fra forige brugertest, lavede vi et nyt sprint for at iterere videre på produktet. Dette er et eksempel på ændrede krav for produktet, som

forekommer undervejs i udviklingen. Med scrum er dette ret nemt håndteret med et nyt sprint.

Dette nye sprint inkluderede at udvikle en info/hints knap til at få hjælp til opgaven, som testpersonen manglede i forige test. Derudover skulle den inkludere en sandhedstabel, som testpersonen kunne sammenligne med sin eget kredsløb:



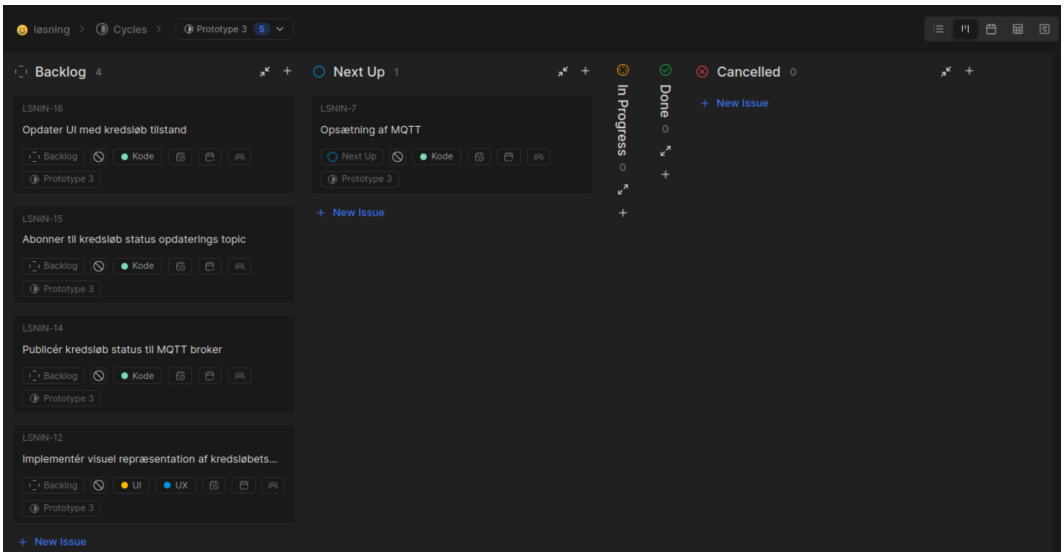
Figur 12: Opdateret opgave brugerflade (venstre) og valg af opgaver brugerflade (højre)

Antallet af opgaver blev reduceret, for at kompensere for testpersonens forvirring. Denne beslutning var delvis også foretaget for at holde det simpelt ift. KISS princippet og det 10. princip fra det agile manifest: '*Simplicity – the art of maximizing the amount of work not done – is essential.*'[5].

6.3 Endelig prototype

Den endelige prototype skulle inkludere en visuel fremvisning af kredsløbets tilstand (Se Figur 8). Dette krævede en række opgaver for at få til at fungere, som blev grupperet ind i et nyt sprint:

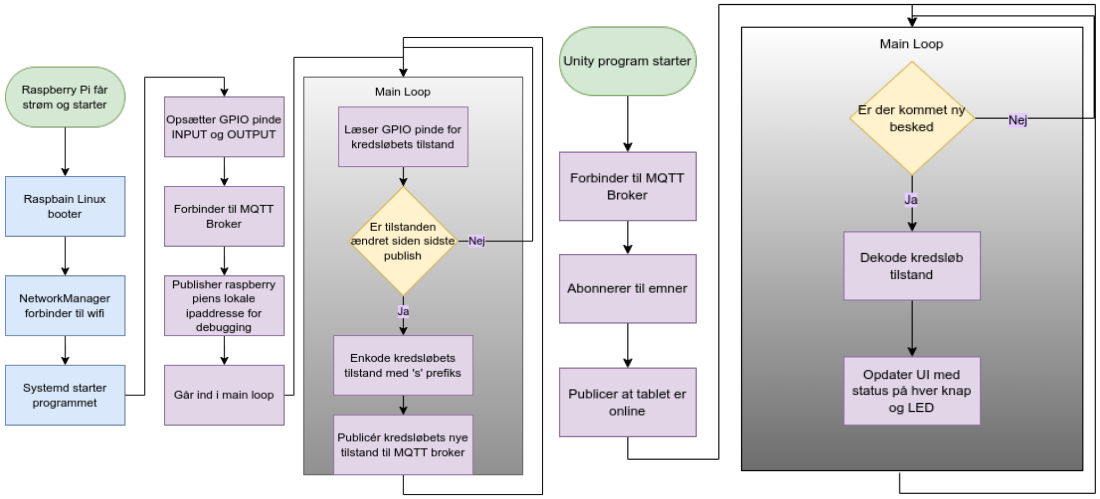
[5]Manifesto 2001, princip 10.



Figur 13: Sprint af endelig prototype

Dette krævede at både det fysiske produkt og tabletten kan kommunikere med hinanden. Dette bliver gjort over MQTT. Der er specifikt blevet valgt MQTT, fordi det løser mange problemer med direkte kommunikation, som at vide afsender og modtagers ip adresse, ved at kommunikere mellem en tredjepart broker. Derudover er MQTT's udgiv abonner mønster oplagt at anvende for dette scenarie, hvor at tabletten (og potentielt andre enheder) har brug for at vide når tilstanden af kredsløbet ændrer sig.

Det fysiske produkt har en Raspberry Pi, hvis formål er at måle tilstanden af kredsløbet og kunne sende input til det. For at opgavesystemet, kan få kredsløbets tilstand, kræver det at Raspberry Pien sender dataet over netværket.
For dette anvendes WiringPi og Paho MQTT C bibliotekerne. WiringPi er for at kunne interagere med GPIO pindene på Raspberry Pien. Paho MQTT C er for at kunne publicere denne data til MQTT brokeren. Raspberry Pien programmeres i C.



Figur 14: Flowdiagram for Raspberry Pi (venstre) og flowdiagram for opgavesystem (højre)

Raspberry Pien starter med at opsætte de forskellige GPIO pinde:

```
1  int main(int argc, char *argv[])
2  {
3      wiringPiSetup();
4      pinMode(IN0, INPUT);
5      pullUpDnControl(IN0, PUD_DOWN);
6      pinMode(IN1, INPUT);
7      pinMode(IN2, INPUT);
8      pinMode(IN3, INPUT);
9
10     pinMode(OUT0, INPUT);
11     pinMode(OUT1, INPUT);
12     pinMode(OUT2, INPUT);
13     pinMode(OUT3, INPUT);
14
15     // ... //
16 }
```

Her sættes hver pin som måler knapperne (inputtet) og LEDerne (outputtet), til at være inputs, hvor pin `IN0` bruger en pull down resistor for at invertere inputtet. Dette er fordi

at denne pin naturligt er inverteret baseret på Raspberry Piens pinmap.

Herefter laves en MQTT client, sætter de forskellige callbacks op; når forbindelsen til brokeren mistes, når en besked bliver leveret og når en besked modtages fra en abonneret topic:

```
12      // ... //
13      if ((rc = MQTTClient_create(&client, ADDRESS, CLIENTID,
14      MQTTCLIENT_PERSISTENCE_NONE, NULL)) != MQTTCLIENT_SUCCESS)
15      {
16          printf("Failed to create client, return code %d\n", rc);
17          exit(EXIT_FAILURE);
18      }
19
20      if ((   rc = MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered))
21              != MQTTCLIENT_SUCCESS)
22      {
23          printf("Failed to set callbacks, return code %d\n", rc);
24          rc = EXIT_FAILURE;
25          MQTTClient_destroy(&client);
26          return rc;
27      }
28
29      conn_opts.keepAliveInterval = 20;
30      conn_opts.cleansession = 1;
31      printf("Connecting...\n");
32      if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
33      {
34          printf("Failed to connect, return code %d\n", rc);
35          rc = EXIT_FAILURE;
36          MQTTClient_destroy(&client);
37          return rc;
38      }
39      // ... //
```

Det kan ses at den laver en MQTT client objekt (l. 13). Dette er ikke et objekt som fra objekt orienterde sprog som C#, men en `struct`, som indeholder specificerede informationer omkring klienten. Hvis dette fejler, rapporteres fejlen og programmet afslutter (ll. 16-17). Herefter registerers callbacksne (l. 20). Til sidst forbinder den klienten til

MQTT brokeren (l. 31).

Herefter går den i dens main loop hvor den tjekker for ændringer i kredsløbets tilstand og publicere dem til brokeren (Se Figur 14):

```
38      // ... //
39      do {
40          uint8_t newCircuitState = getCircuitState();
41          // Something changed... we should notify subscribers
42          if (circuitStatus != newCircuitState) {
43              circuitStatus = newCircuitState;
44              printState(circuitStatus);
45              printf("state: %s\n", statePayloadBuf);
46
47              pubmsg.payload = statePayloadBuf;
48              pubmsg.payloadlen = (int)strlen(statePayloadBuf);
49              pubmsg.qos = QOS;
50              pubmsg.retained = 0;
51              if ((   rc = MQTTClient_publishMessage(client, STATE_TOPIC, &pubmsg, &token))
52                      != MQTTCLIENT_SUCCESS) {
53                  printf("Failed to publish message, return code %d\n", rc);
54                  exit(EXIT_FAILURE);
55              }
56          }
57      } while (1);
58
59      // ... //
```

For dette bruger den en hjælpefunktion `getCircuitState()`, som finder ud af kredsløbets tilstand:

```
1   uint8_t getCircuitState() {
2       uint8_t newState = digitalRead(IN0);
3       newState |= digitalRead(IN1) << 1;
4       newState |= digitalRead(IN2) << 2;
5       newState |= digitalRead(IN3) << 3;
6       newState |= digitalRead(OUT0) << 4;
7       newState |= digitalRead(OUT1) << 5;
8       newState |= digitalRead(OUT2) << 6;
9       newState |= digitalRead(OUT3) << 7;
10
```

```
11      return newState;
12  }
```

Kredsløbets tilstand er enkodet i en enkelt byte (`uint8_t`) – et såkaldt bitfield. De første 4 bits er reserveret til tilstanden af hver knap (input). De sidste 4 bits er reserveret til tilstanden af hver LED (output). Funktionen gør brug af de bitwise operatorer som OR og left shift. Den læser tilstanden af den associerede pin for knappen og tilføjer det til bitfieldet vha. OR operatoren (l. 3). Den bruger left shift operatoren `<<` til at shifte alle bits af `digitalRead()`, som returnere 1 hvis pinden er logisk høj og 0 hvis pinden er logisk lav (l. 3). Dette gør den for hver knap og LED pin, indtil bitfieldet er genereret (ll. 4-9), og det returneres (l. 11).

Efter den nye kredsløbs tilstand er beregnet tjekker den om det matcher den gamle (l. 42). Hvis ikke de gør, må der have været sket en ændring i kredsløbets tilstand, hvilket betyder at den publicere denne ændring til brokeren (l. 51).
I Unity anvendes MQTTNet biblioteket for C# til at modtage beskeder fra brokeren.

```
1  public static async void Setup()
2  {
3      k_mqttOptions = new MqttClientOptionsBuilder()
4          .WithTcpServer(k_server, k_port)
5          .WithClientId(k_clientID)
6          .Build();
7      await ConnectClient();
8      Subscribe();
9  }
```

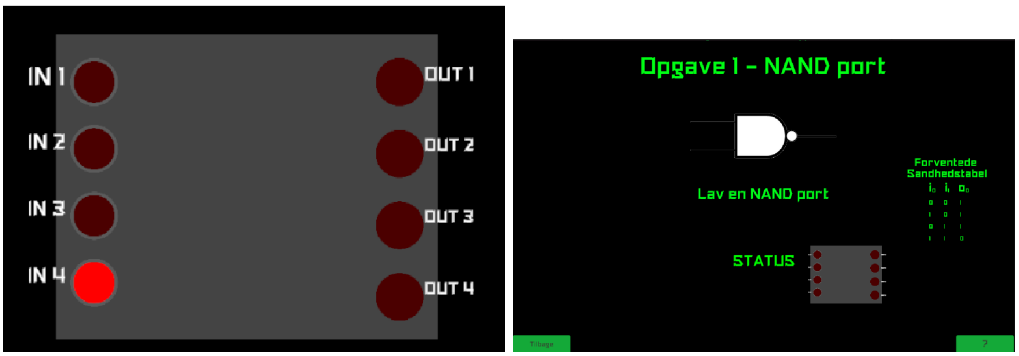
`Setup` bliver kaldt når opgavesystemet starter. Den starter med at opbygge klientens indstillinger (ll. 3-6) fra MQTTNets `MqttClientOptionsBuilder` klasse, som bruger builder design mønsteret. Herefter forbinder den til klienten (l. 7) (implementation udeladt for simplifikation). Herefter abonnere den til emnet omkring opdatering af kredsløbets tilstand (l. 8) (implementation udeladt for simplifikation).
Kredsløbets struktur bliver repræsenteret som en klasse i opgavesystemet:

```
1  public class CircuitState
2  {
```

```
3       public bool in1, in2, in3, in4;
4       public bool out1, out2, out3, out4;
5
6       public static CircuitState FromBitfield(byte field) {
7           CircuitState state = new CircuitState();
8           state.in1 = (field & 1) != 0;
9           state.in2 = ((field >> 1) & 1) != 0;
10          state.in3 = ((field >> 2) & 1) != 0;
11          state.in4 = ((field >> 3) & 1) != 0;
12
13          state.out1 = ((field >> 4) & 1) != 0;
14          state.out2 = ((field >> 5) & 1) != 0;
15          state.out3 = ((field >> 6) & 1) != 0;
16          state.out4 = ((field >> 7) & 1) != 0;
17          return state;
18      }
19      // ... //
20  }
```

Når opgavesystemet får en besked får brokeren omkring en ny opdatering af kredsløbets tilstand, så anvender den `FromBitfield` metoden, til at dekode kredsløbsdataet fra Raspberry Pien. Metoden gør også brug af bitwise operatorer til at læse værdien af hver bit i bitfeltet. Algoritmen starter med at læse den første bit (LSB) af bitfeltet ved at bruge AND operatoren (l. 8). Herefter shifter den alle bits i bitfeltet til højre hvorefter den læse det første bit (LSB) i feltet igen, som nu indeholder værdien af bitten til højre fra den forige. Dette fortsætter indtil hver bit i feltet er læst.

Hermed blev et statusdisplay implementeret for hver opgave. Når en knap trykkes på eller et output går logisk højt, så viser UI'en en lysere farve rød:



Figur 15: Visuel fremvisning af kredsløbets tilstand (venstre) og den endelige opgave brugerflade (højre)

7 Test af endelig prototype

Der blev foretaget en brugertest på den endelige prototype af produket med en computer science elev på Københavns Mediegymnasium.
En video af brugertesten kan ses her: https://www.youtube.com/watch?v=ycqhdVVqKJM.

Ud fra brugertesten kan det ses at testpersonen bygger et forkert kredsløb, formentligt fordi han ikke forstår den viste sandhedstabel, og at der ikke er nogen evalueringsfunktionalitet. Han skulle bygge e NAND port, men byggede blot en AND port.

8 Evaluering

Baseret på brugertesten af den endelige prototype, kan opgavesystemet forbedres ved at inkludere en introduktion til hvad en sandhedstabel er, så brugeren ved hvornår de har lavet en opgave korrekt. Derudover som nævnt i brugertest videoen, så mangler der en test feature, til at evaluere om brugeren har klaret en opgave korrekt. Dette skyldes bl.a. at der var en del uforudsigelige problemer som opstød under udviklingen af det fysiske produkt, som medførte en større allokering af resurser til udvikling af det fysiske produkt fremfor opgavesystemet. Dette betød at de fleste sprints havde få opgaver for

at kunne nå det til sprintets deadline. Derudover kunne det skyldes dårlig udførelse af scrum. Der var mange opgaver, som var for overfladiske, idét at de krævede mere arbejde end forventet. F.eks. indeholdte opgaven omkring opsætning af MQTT mere arbejde end forventet – vælge et tilpas bibliotek for både Raspberry Pien (C) og Unity (C#), undersøge hvordan bibliotekerne fungerede, forbinde til broker og abonnere til topics for begge enheder. Dette betød at det tog længere tid end forventet at implementere de opgaver. En dekomposering af opgaverne kunne have hjulpet med dette. Derudover blev sprint reviews ofte unladt, hvilket betød at gruppens overblik var ret dårlig gennem udviklingsprocessen, som er hvad sprint reviews er lavet for at forbedre[6]. Hvis der blev udført et ordentligt sprint review efter hvert sprint, så ville det potentielt have gjort os opmærksom på at der var tidspres inden det var for sent.
Der blev derudover observeret at brugerfladen for fremvisningen af kredsløbets tilstand var forvirrende at forstå for nogle testpersoner. Der var nogle testpersoner der var i tvivl hvad de røde cirkler repræsenterede. Dette kunne blive løst i en ekstra iteration, hvor de røde circkler blev erstattet med et ikon for en knap og LED i stedet.
Udover dette, bemærkede vi at tilbage og info knappen på opgaveskærmen, godt kunne blive lidt større, så de er nemmere at se, samt få noget margen omkring dem, så de er nogen luftrum imellem knapperne og skærmens kanter.

[6]Ken Schwaber og Jeff Sutherland 2020, s. 9.

9 Konklusion

Ud fra projektets problemformulering skulle der til dette projekt udarbejdes et digital opgavesystem til et fysisk produkt om digital logik målrettet mod computer science elever på gymnasium. Opgavesystemet skulle være en introduktion til emnet, og derfor undervise om de basale logiske operatorer, som AND, NOT, OR og XOR.

Problemformuleringen blev udarbejdet baseret på en kvantitativ spørgeskema undersøgelse, som gav et empirisk fundament for hvilken en sværhedsgrad opgavesystemet skulle være. Herefter blev der idégenereret for hvordan opgavesystemet kunne fungere ud fra den indsamlede empiri fra empathize fasen. Dette resulterede i skitser af brugerflader og flowdiagrammer af brugerens rejse gennem deres interaktion med opgavesystemet. Dette fungerede som fundamentet for udviklingen af den første prototype, efterfulgt af en brugertest. Fra brugertesten, blev der fundet problemer og mulige forbedringer, som blev anvendt til at udvikle en ny iteration af prototypen. Denne iterative arbejdsproces fortsatte indtil deadline.

Undervejs blev der anvendt en agil arbejdsmetode – scrum. Scrum hjalp undervejs til at opdele projektets krav i mindrere opgaver. Den endelige prototype opfyldte alle de stilte krav foruden en evaluerings funktionalitet til at teste om kredsløbet opfyldte opgavens krav. Det viste sig at dette bl.a. var fordi at gruppens resurser blev allokeret til problemer der opstod ved udviklingen af det fysiske produkt. Derudover blev det vist at det potentielt også kunne være have været pga. af dårlig anvendelse af scrum ift. at sprint reviews ofte blev undladt, hvilket førte til at gruppen ikke indså at gruppen var bagud før det var for sent. Til sidst blev udført en bruger- og funktionalitetstest (Se sektionerne **Test af endelig prototype** og **Bilag 1 – Fysisk produkt fra teknikfag**), som kunne anvendes som empirisk fundament for videreudvikling i fremtiden.

Kildeliste

Manifesto, Agile (2001). *Principles behind the Agile Manifesto*. URL: https://agilemanifesto.org/principles.html.

Undervisnings Ministeriet (2017). URL: https://www.uvm.dk/gymnasiale-uddannelser/fag-og-laereplaner/laereplaner-2017/htx-laereplaner-2017.

Ken Schwaber og Jeff Sutherland (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

Martin Damhus, Jesper Buch og Elisabeth Husum (2024). *Informatik*. Systime. ISBN: 9788761686510. URL: https://informatik.systime.dk.

Plane (2024). URL: https://plane.so.

10 Bilag

10.1 Bilag 1 – Fysisk produkt fra teknikfag

Det fysiske produkt består af en kasse, hvori der befinder sig et 4x4 gitter med magneter der kan lede strøm. Magneterne er multifunktionelle, og fungerer både som en mekanisme til at klikke brikker fast på brættet / gitteret, samt en måde at forsyne brikken med strøm. På venstre side af gitteret er der en kolonne af fire knapper – en for hver række i gitteret. For hver knap er der en ledning man kan forbinde til brikker der er på gitteret. Hvis man klikker på knappen, sender den en elektrisk strøm ind til den brik, knappen er forbundet til. På højre side af gitteret er der en kolonne med 4 LED'er, samt en ledning for hver LED, der tager imod kredsløbets output. Hvis outputtet af en logisk port er højt, så lyser LED'en, ellers er den slukket. Under gitteret befinder der sig en mikrokontroller (Raspberry Pi), som måler de elektriske signaler for hver knap, og hver LED. Denne data sendes over netværket til en tablet. Tabletten er monteret på displayet, og skal fungere som et opgaveprogram, med en liste af forskellige opgaver. Tabletten bruger den sendte data fra mikrokontrolleren til at give brugeren en status af kredsløbet i reel tid. Derudover giver det brugeren en mulighed for at teste om de har lavet sit kredsløb rigtig ift. hvilken opgave brugeren har valgt. På forsiden af kassen er der nogle udadvendte magneter, som kan holde på evt. brikker, der ikke er i brug.
På nedenstående link, kan ses en funktionstest af produktet som helhed:
https://www.youtube.com/watch?v=RHwA4xU5wxA

10.2 Bilag 2 – Kildekode af Raspberry Pi C program

Koden kan findes på github (under src/):
https://github.com/Sveske-Juice/zero-circuittransmitter

10.3 Bilag 3 – Unity projekt kildekode

Unity projektfiler inkl. kildekode (under Assets/Scripts/) kan findes på github:
https://github.com/Gags50/LogicGateApp