# Konklusion

Her gives en samlet konklusion på projektarbejdet. Hvad er lykkedes, hvad er evt. Ikke lykkedes og årsagen til dette. Konklusionen skal gerne indeholde et klart budskab og forholde sig objektivt til de opstillede krav og opnået resultater.

Desuden sammenfattes de slutninger, der kan drages af de resultater, som er omtalt i rapportens tidligere afsnit. Konklusioner kan være såvel positive som negative. Man skal tage sig i agt for ikke at undertrykke de negative fund (hvis f.eks. en metode har vist sig uegnet, bør det opfattes som et bidrag til ens erfaringsmateriale, ikke som et personligt nederlag).

I konklusionen trækkes desuden de store linier op. Væsentlige kvantitative resultater kan nævnes, hvorimod den detaljerede redegørelse og argumentationen henvises til diskussionen i rapportens hoveddel.

Projektet er endt ud med en WPF-applikation, en web-applikation og tilhørende database. Den ene database kører lokalt og den anden kører på en Azure server sammen med web-applikationen. De to databaser synkroniserer automatisk, eller ved at brugeren aktivt beder om det ved et tryk på brugerinterfacet. Der er implementeret flere funktionaliteter på WPF-applikationen sammenlignet med web-applikationen, da det er her gruppens største fokus har været. Arbejdet med WPF applikationen startede før introduktionen til MVP, MVC & MVVM. Konsekvensen af dette er manglende afkobling af XAML, codebehind og business logic. Dette har gjort det utrolig svært at teste, og derfor er der heller ikke skrevet så mange unittests og integrationstests som ønsket.   
Database tilgangen er i WPF implementeret med ADO.NET, hvilket har givet en del problemer i forhold til mapning, og at bevare data integriteten i selve databasen.

I web-applikationen har vi haft fokus på at få designet og struktureret programmet bedre end vores WPF. Her har vi brugt MVC, og haft fokus på at overholde dette designprincip. Formålet har været at bruge alle de erfaringer vi har fået under udviklingen af WPF-applikationen og udvikle en velstruktureret web-applikation.  
Databasen tilgangen er lavet med entity-framework og mange features og mapping sker derfor automatisk. Dette har gjort web-applikationen væsentligt mere læsbar, og vedligeholdelsesvenlig end WPF-applikationen.

I projektet er der brugt continous integration i form af et GIT repository. Brugen af GIT har gjort det utrolig let at arbejde i det samme Visual Studio projekt på samme tid, og rette fejl der kommer når nye features implementeres. Der har dog været en nogle problemer i form af mergeing, når der er blevet ændret i præcis de samme filer, imellem hvert push/pull. Ved disse merges er der nogle få gange gået data tabt, som skulle bevares. På trods af dette opvejer fordelene klart de ulemper der har måtte være.

Git var ligeledes en nødvendighed da automatering af tests i form af Jenkins var et ønske, og sammenkoblingen af GIT & Jenkins var blevet introduceret i I4SWT. Det var dog ikke muligt da der uden adgang til selve Jenkins serveren ikke kunne laves de nødvendige rettelser for at få Jenkins til at builde projektet.

Selve gruppearbejdet har været udført med en agil tankegang, og har primært været inspireret af SCRUM, i form af stand-up møder, retrospektmøder, sprints og backlog. Den agile udviklingsproces har gjort det muligt hele tiden at have et fungerende produkt, som kunne vises til vejlederen, og løbende tage de vigtigste opgaver. Det har også muliggjort at alle i projektgruppen hele tiden er klar over hvor langt projektet er, og kan hjælpe hindanden med det samme en forhindres i at arbejde videre.

Alt dette har resulteret i et enestående slutresultat, med et projekt der opfylder alle de krav der blev sat i starten af projektet. Det har dog ikke været muligt at implementere nogle ekstra tilføjelser, grundet tidspres. Dog er der blevet stiftet bekendtskab med nye teknologier, som Azure, hvilket der ikke er blevet undervist i. Ligeledes er der synkronisering imellem to databaser, og en web-applikation, hvilket der ikke har været stort fokus på uden for projektet.