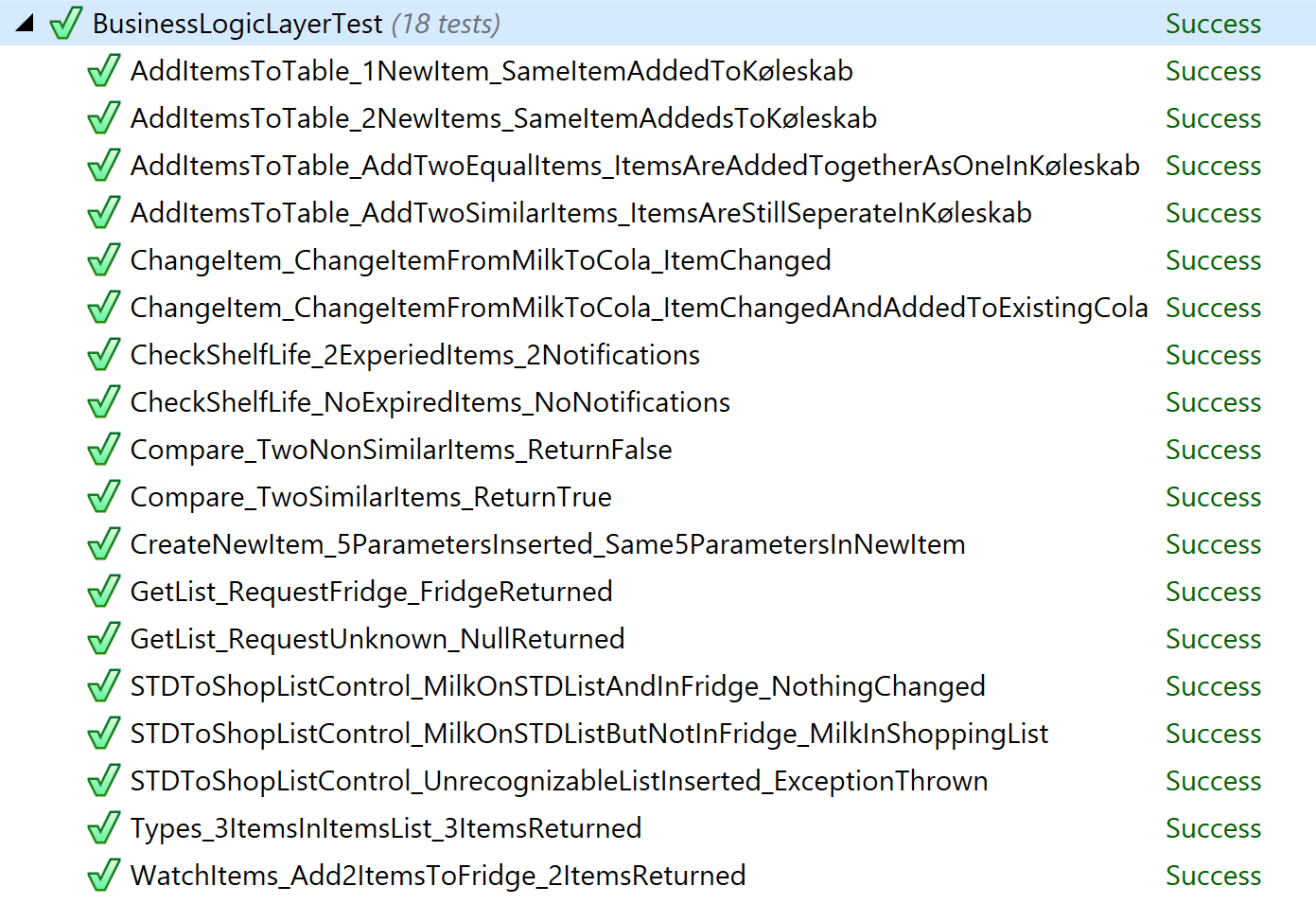
# Integrationstests

På Figur 1 ses klassen der indeholder tests af Business Logic Layeret.

De tests der er at finde i BusinessLogicLayerTest er som udgangspunkt unittests, men bl.a. pga. manglende mulighed for at substituere ’Listen’ ud, vil flere af disse tests agere som integrationstests. Stort set alle funktionaliteter bliver testet, på nær private funktioner. Blandt andet er der ikke skrevet en test for den private funktion ’CreateList’, der bliver brugt til at oprette en liste hvis den ikke eksisterer i forvejen.

Figur 1 - Test af BusinessLogicLayer

Testene er integrationstests, da der allerede i BLL’ens constructor bliver oprettet forbindelse til databasen.

Fra henholdsvis testfilen og BLL-filen:

|  |  |
| --- | --- |
| **public** BLL uut**;**  **[**SetUp**]**  **public** void SetUp**()**  **{**  uut **=** **new** BLL**();**  uut**.**CurrentList **=** "Køleskab"**;**  var items **=** uut**.**WatchItems**;**  **}** | **public** BLL**()**  **{**  Context **=** **new** AdoNetContext**(**\_connectionFactory**);**  \_itemRepository **=** **new** ItemRepository**(**Context**);**  \_listRepository **=** **new** ListRepository**(**Context**);**  \_listItemRepository **=** **new** ListItemRepository**(**Context**);**  **}** |

Kodestump 1 - BLL test SetUp og BLL constructoren

Dette medfører at det er nødvendigt at rydde op efter udførsel af test. Her er et eksempel på en test:

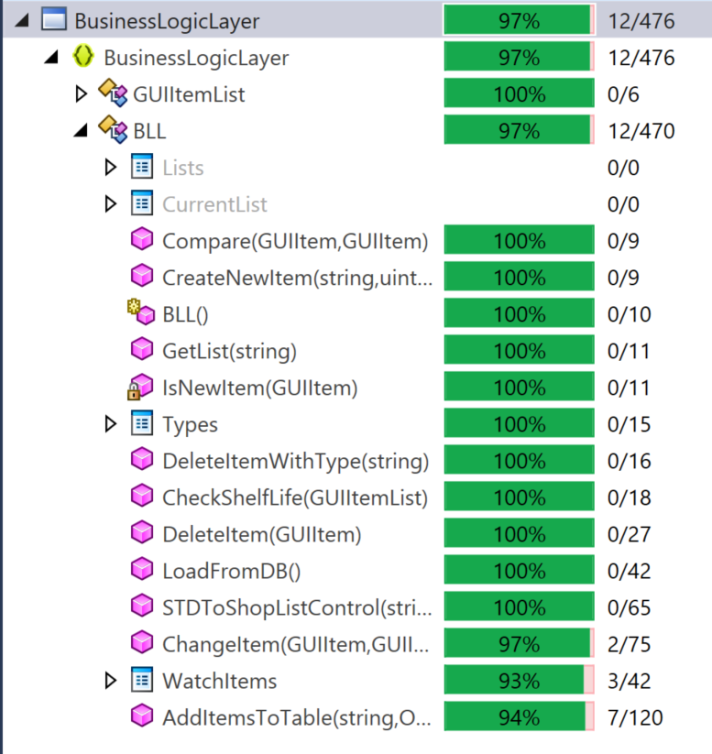
|  |
| --- |
| **[**Test**]**  **public** void AddItemsToTable\_1NewItem\_SameItemAddedToKøleskab**()**  **{**  var items **=** **new** ObservableCollection**<**GUIItem**>();**  var rnd **=** **new** Random**();**  string type **=** rnd**.**Next**(**int**.**MinValue**,** int**.**MaxValue**).**ToString**();**  items**.**Add**(new** GUIItem**(**type**,** 5**,** 1**,** "Unit"**)** **{** ShelfLife **=** DateTime**.**Now **});**  string listName **=** "Køleskab"**;**  uut**.**AddItemsToTable**(**listName**,** items**);**  //Find the list...  GUIItemList list **=** **null;**  **foreach** **(**var guiItemList **in** uut**.**Lists**)**  **{**  **if** **(**guiItemList**.**Name**.**Equals**(**"Køleskab"**))**  list **=** guiItemList**;**  **}**  int EqualItems **=** 0**;**  **foreach** **(**var guiItem1 **in** items**)**  **{**  **foreach** **(**var guiItem2 **in** list**.**ItemList**)**  **{**  **if** **(**uut**.**Compare**(**guiItem1**,** guiItem2**))**  EqualItems**++;**  **}**  **}**  Assert**.**AreEqual**(**1**,** EqualItems**);**  **foreach** **(**var guiItem **in** items**)**  **{**  uut**.**DeleteItem**(**guiItem**);**  uut**.**DeleteItemWithType**(**guiItem**.**Type**);**  **}**  **}** |

Tabel 2 - AddItemsToTable test

I testen oprettes en liste af GUIItems, hvor der tilføjes en vare til listen. Derefter kaldes *AddItemsToTable*, hvor den enkelte vare bliver tilføjet til *Køleskab*. Derefter findes *Køleskab*, og der tjekkes hvorvidt varen reelt er tilføjet til *Køleskab* eller ej. Dette gøres ved at udnytte *Compare*-funktionen, som er testet tidligere. Der bruges *EqualItems* for at sikre at den samme vare ikke er tilføjet flere gange end én. Efter assertes der på om testen gik godt og der ryddes op ved at slette det *ListItem* og *Item* som varen var knyttet til.

Generelt er testene opbygget sådan; Konstruktion af varer, test af metode, oprydning.

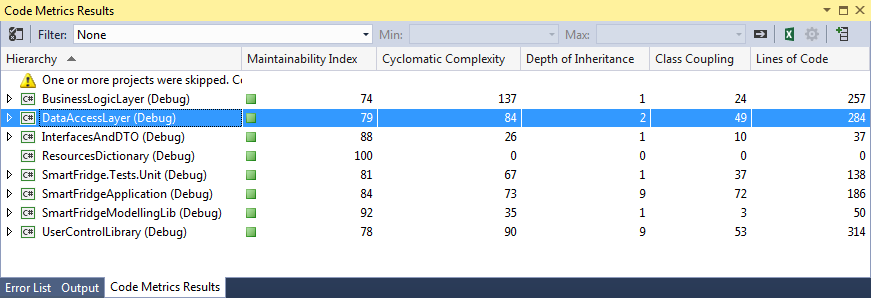
## Coverage

På Figur 2 ses et screenshot af Coverage-rapporten. Her er ikke helt opnået fuld coverage. WatchItems, ChangeItem og AddItemsToTable har alle muligheden for at kalde ’CreateList’. Da BLL-testen i høj grad er en integrationstest, medfører det oprydning af de tests man foretager sig. Der er ikke skrevet en funktion for ’DeleteList’, og derved kan Listen ikke opryddes ved test af disse specielle tilfælde. Dermed er de sidste procenter i de 3 nederste metoder i Figur 2 ikke testet, og giver BLL’en en coverage på 97%.

Figur 2 - Coverage af BusinessLogicLayer

## Static analysis

Her ses resultaterne for en statisk analyse foretaget af Visual Studio:



Figur 3 – Static analysis, code metrics

De følgende vurderinger af resultaterne tager udgangspunkt i MSDN’s retningslinjer (Microsoft, 2015)

### Maintainability index

Giver indikation af hvor let det er at vedligeholde koden.

Da en værdi mellem 20 og 100 indikerer god maintainability, har samtlige projekter i vores solution en god maintainability, med et upper peak på 84, da SmartFrdigeModellingLib ikke er applikationskode.

### Cyclomatic Complexity

Giver indikation af hvor kompleks kodetraverseringen er, altså om der er mange cyklusser.

Generelt denne værdi ikke enorm høj i solutionen, men BusinessLogicLayer har en semi-høj CC i forhold til DataAccessLayer. Dette kan skyldes et højt antal af hjælpefunktioner i BLL.

### Depth of Inheritance

Giver indikation af arveheraki-kompleksiteten. Generelt lav da der ikke bliver arbejdet meget med arv i vores solution, da det ikke har været særlig relevant.

### Class Coupling

Giver indikation af koblingen mellem klasser. Her er resultatet højt ved SmartFridgeApplication, da det er hovedprogrammet i vores solution. Med andre ord er det den store klasse der afhænger af en del små klasser. Her kunne man have valgt ligge et interface på flere klasser.