- Till och börja med så ETL språket är ett simpelt och lätt programmeringsspråk som liknar de mesta vanliga programmeringsspråk till ex: Ruby, Python osv, men med hänsyn till målgruppen som kanske inte har programmerat innan. ETL är ett imperativ programmeringsspråk som innehåller de grundläggande saker som täcker de flesta viktiga delar som en nybörjare ska börja med och behärska för att lära sig programmering. Språket kommer även passa lärarna som vill lära ut programmering till de nybörjare eller möjligtvis till en grupp av barn i grundskolan. ETL liknar det skriftliga engelska vilket gör det lättare för språket läsbarhet.
- Vi har byggt ETL språket med tanken på att det finns inget annat språk som har samma enkel nivå då det kan vara svårt för vissa nybörjare att lära sig en av de kända språk liksom python, ruby, java eller c++. Hade vi som inte programmerat innan började med ett språk i ETLs enkelhet nivå i gymnasieskolan kanske då skulle det vara till stor hjälp dvs ett bra framsteg för att fortsätta med programmering inlärning i framtiden istället för att direkt börja med python till exempel.
- Som vi nämnde tidigare, det krävs inga tidigare kunskaper inom programmering när man programmerar med ETL, därför har vi valt att ge användaren så mycket frihet som möjligt. Till exempel, vi har ingen specifik indentering som användaren måste följa, ingen sluttecken heller som till exempel semikolon eller något liknande för att avsluta en rad kod. Vi tog även bort de sakerna som kan göra kodningen krångligare för användaren till exempel, användaren behöver inte skriva alltid <startprogram> i början av programmet och <endprogram> i slutet av programmet.

- För att kunna börja med implementeringen så behövde vi rdparse.rb filen för att kunna parsa, samt har vi designat hur kommer språk grammatiken se ut dvs själva BNFen. Där började vi med att implementera språket med hjälp av grammatiken och tokens samt de noder som var nödvändiga för denna delen. Vi fortsatte med denna processen tills vi fick flera saker fungera tillsammans.
- I lexikaliska analysdelen skapas de olika tokens som språket har. Tokens består av reguljära uttryck(RegEx) som är en följd av flera tecken som matchar en viss mönster. Därefter kommer alla tokens skickas vidare till parsningsdelen.

- Parsning delen kommer få alla tokens som är skickats från lexikaliska delen för att parsas. Den kommer matcha de reglerna som är skriven i själva BNFen dvs grammatikdelen då kan parsern börja göra sitt jobb som är att hitta det mönstret från det koden som användaren skriver och bygga därefter det abstrakta syntaxträdet.
- ETL har dynamisk scopehantering, det betyder att en funktion kommer ha sin egen scope som nummer (1) där scopet kommer behålla den funktionen samt deras argument(variabler) som används i denna funktionen i en och samma hash. Där funktionen kommer titta först på de variablerna som finns inuti det samma scopet som funktionen finns i, om den variabeln finns på den scopet så kommer språket använda värdet av denna variabeln, annars om den inte finns på samma funktions scope så kommer då funktionen titta på de variablerna som finns utanför dvs scopet nummer (0) vilket är det globala scopet. Men användaren kan inte ha tillgång till lokala variabler som finns på det lokal scopet vilket är i vår fall nummer (1).
- ETL använder sig av två verktyg för att implementera språket vilka är Ruby och befintliga parser som heter rdparse.rb.

- Vi hade inte så bra början på implementeringen eftersom vi hade gått för långt med att skriva mycket saker som inte var relevanta på den dära första tidpunkten. Detta märkte vi tack vare vår handledare under en av handledningstillfällen som rekommenderade att vi borde ta saker ett steg i taget för att testa och se om de fungerar eller inte.
- Ett av de problemen vi hade under arbetet var att minustecknet inte fungerade som det ska när man skriver (x - y) med mellanrum. Vi lyckades lösa problemet genom att ändra på vår tokens så att de matchar bara tal oavsett de är positiva eller negativa, sedan ändrade vi på Constant klassen så vi lade till en if-sats som säger om det är negativ så ska den siffran multipliceras med (-1).
- Ett stort problem vi stött på under projektet var ordningen på statement matchgrupperna samt de andra matchgrupperna i BNF. Där vi började få samma felmeddelande för flera saker vi skapade. Detta tog lång tid för att hitta vart problemet är, där vi märkte i slutet att det ligger på ordningen där minst generella ska komma först i ordningen och mest generella ska vara i slutet. Det handlar mest om erfarenhet man får under projektarbetet, skulle

vi vara medvetna på att minst generella ska vara först i ordningen så skulle det vara snabbt att fixa problemet eller kanske vi inte skulle hamna på detta problemet alls.

- Vi är glada att vi fick mycket stora erfarenheter som vi inte behärskade innan projektets gång och vi tycker också att vi har nått målet som var att förstå hur ett programmeringsspråk är uppbyggt samt vilka verktyg det behövs för att skapa ett eget programmeringsspråk.