Basically, the objective of this project is to use some objects and also the algebra among them (fuctors, monads, functor algebras , monad algebras), originating from Category Theory, with the help of Haskell PL to create alternative solution strategies to the problems (failure detection, purity, avoiding side-effects, ...) existing in Computer Science, particularly, in the context of functional programming.

1. Category Theory - As a mathematical background
* Categories
* Functors
* Natural Transformations
* Monads

1. Haskell programming language
* Basic Classifications of programming languages
* What is Functional Programming
* What is Haskell
* Haskell's Type System (Static and Strong)
	+ Type Signature
	+ Type Variables
	+ Type Classes
	+ Creating new types in Haskell
	+ Definition of Haskell's HASK category (Types are Objects, Functions between objects are Morphisms)
1. Functors of Haskell
* From theory to coding (theory and its representation in Haskell)
* Proofs
	+ Maybe as a functor
	+ List as a functor
	+ State as a functor
	+ Continuation as a functor
	+ Either as a functor
	+ Reader as a functor
	+ Writer as a functor
1. Monads of Haskell
* From theory to coding (theory and its representation in Haskell)
* Proofs
	+ Maybe as a monad -- Failure Detection
	+ List as a monad -- Non Determinism
	+ State as a monad -- Providing Purity
	+ Continuation as a monad -- Providing Continuation
	+ Either as a monad -- Exception Detection
	+ Reader as a monad -- Dependency
	+ Writer as a monad -- Output
	+ IO as an impure monad with the help of State monad -- Avoiding Side Effects
* Examples of monadic programming to show the advantages of each.
* Monad Transformers of Haskell
	+ MaybeT
	+ ListT
	+ StateT
	+ Examples
1. Functor Algebras
* Theory
	+ F-Algebras
	+ Initial Algebras
	+ Catamorphisms
	+ Paramorphisms
* Haskell Representations (if possible)
1. Monad Algebras
* Theory
	+ T-Algebra
* Haskell Representations (if possible)
1. Conclusion