Goals:

To provide a standard consistent interface to compile, test, package all developed software.

High level requirements:

- Must be self-contained
- Must return exist codes
- Platform independent
- Must be in version control and sit in the root of the project
- All parts of the specification must be completely implemented, if it's not applicable it must be mocked.
- Flags can be put together in any order as long as all the requirements are met.
- All deploy-able files will be in a defined folder and the predefined structure.

Deployment folder specification

At the end of every compile the following folder structure is required.

At the root of the checkout. which build.py exists, Structure:

[*ROOT of checkout*]
|—[**SHIP**]
|—[***target***] (eg: debug)
|—[**product**] (all binary files required for deployment, can be multiple subfolders, must be target machine required structure.)
|—[**unit-test**] (all unit test binaries)
|—[**configuration**] (all run time configuration)
|—[**database**] (all database scripts)

build.py CLI specifications

Action	Flag	Function	Requirements
Pre build checks	-p	Runs any pre build checks required. eg: Checks if dependencies are installed, has port access to required systems, etc **Usage** `build.py -p` Optional: Nice to have would be to check if you have un-commit code or files which have not been added to source control	None

Action	Flag	Function	Requirements
		and warn the user. Pause for 10 seconds and run.	
Define target	-t	Defines build flavor type. eg: debug | prod | profiling | etc **Usage** `build.py -t debug -b -u` This example builds (possibly an incremental as not -c is supplied) then runs the unit test.	• Mandatory • Must have another action associated.
Clean	-c	Deletes any meta files which are generated as part of a compile for given type eg: intermediate object files and directories. Including delete the existing SHIP folder Must no delete non build generated file such as source files not yet commit to etc. **Usage** `build.py -t debug -c` This example will clean the target compile.	-t flag
Build	-b	Compiles the given target. All required binaries must be compiled, this included all unit test binaries. All files must be put in the define structure **Usage** `build.py -t debug -c -b` This example will clean and compile a debug target	-t flag
Unit Test	-u	Run all Unit tests from the defined unit-test deployment folder. If the unit test doesn't exit it should print a warning and compile (which will create the folder and binaries) then run the unit tests. All unit tests are required to produce a xml file in the	-t flag

Action	Flag	Function	Requirements
		root of the unit-test folder in junit xml format. One file per unit test binary is required. Naming convention is <name of unit-test-binary>**-results.xml** **Usage** `build.py -t debug -u` This example will run all unit tests in the SHIP\unit-test\ folder	
Coverage Reporting	-k	Compiles a binaries with coverage flags to generate coverage and static analysis data for sonar.	None. Self contained.