Bug

index.html

--

codeanalysis - Copy (2).py

--#BUG: Does not pick up 'TODO' items where square brackets immediately follow colon (e.g. TODO:[e] Blah...).

--print (" BUG Items: " + str(bugCount))

--#BUG: [c]Currently appending CSV row record to existing entry. Does it need a manual newline character adding?

--#BUG: [c]Resolve pypdf import issues.

codeanalysis.py

--#BUG: Does not pick up 'TODO' items where square brackets immediately follow colon (e.g. TODO:[e] Blah...).

--print (" BUG Items: " + str(bugCount))

--#BUG: [c]Currently appending CSV row record to existing entry. Does it need a manual newline character adding? THINK FIXED. BINARY MODE!

--#BUG: [c]Resolve pypdf import issues.

Critical

system.py

--#TODO: [c] Add owner, referenceURI, prerequisites[], (more next lines-->)

codeanalysis - Copy (2).py

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--#(14) Critical: [c] Count

--#(15) Critical: [c] List

--#BUG: [c]Currently appending CSV row record to existing entry. Does it need a manual newline character adding?

--#TODO: [i] Add call to PDF write here. Would like colour coding for categories (e.g replace [c] with RED [c]).

--#TODO: [i] Add call to PDF write here. Would like colour coding for categories (e.g replace [c] with RED [c]).

--sLine = "" + line #HACK: [c] Seems to be writing back the stripping to the list.

--#BUG: [c]Resolve pypdf import issues.

codeanalysis.py

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--#(14) Critical: [c] Count

--#(15) Critical: [c] List

--#BUG: [c]Currently appending CSV row record to existing entry. Does it need a manual newline character adding? THINK FIXED. BINARY MODE!

--sLine = "" + line #HACK: [c] Seems to be writing back the stripping to the list.

--#TODO: [i] Add call to PDF write here. Would like colour coding for categories (e.g replace [c] with RED [c]).

--#TODO: [i] Add call to PDF write here. Would like colour coding for categories (e.g replace [c] with RED [c]).

--#BUG: [c]Resolve pypdf import issues.

precommit.py

--#TODO: [c] Make executable.

predeploy.py

--#TODO: [c] Make executable.

Essential

edu_objects.py

--#TODO: [e] Use...-- https://developers.google.com/appengine/docs/python/datastore/typesandpropertyclasses

--#TODO: [e] add fields: assessmentType [estimates, targets, results, current-working, current-possible(?).

--#TODO: [e] headteacher (staff or string), postcode, address (long string for now), urn, schooltype, ofstedinspections{date, judgement pairs), nextearliestinspection.

--#TODO: [e] add attendance, classes (arr)

--#TODO: [e] add summary assessment values for proof.

--#TODO: [e] add classes (arr)

system.py

--#TODO: [e] Check status of unit test strut here?

--#TODO: [e] Prerequisite metrics.

codeanalysis - Copy (2).py

--# TODO: [e] Write file size metrics.

--# TODO: [e] Write file created metrics.

--# TODO: [e] Write file updated metrics.

--# TODO: [e] match unit test files to class files, and count tests. also get functions/properties without tests.

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [e] objects...is it easiest to import each file, then use dir() on the module? see p. 99)

--COMPUTERSTAMP = socket.gethostname() #os.environ['COMPUTERNAME'] #TODO: [e]Check against http://stackoverflow.com/questions/799767/getting-name-of-windows-computer-running-python-script if doesn't work across systems.

--#TODO: [e] IMPORT here.

--#(16) Essential: [e]

--#(17) Essential: [e] List

--#TODO: [e] Use jflot (or current js render library) to plot line comparison of LoC and TODO items.

--#TODO: [e]Write to PDF.

--#raw_input() #HACK: [e]Added so that results can be read when run from console. Not needed in interactive mode (or if we've run this solely to update the code analysis record).

codeanalysis.py

--# TODO: [e] Write file size metrics.

--# TODO: [e] Write file created metrics.

--# TODO: [e] Write file updated metrics.

--# TODO: [e] match unit test files to class files, and count tests. also get functions/properties without tests.

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [e] objects...is it easiest to import each file, then use dir() on the module? see p. 99)

--COMPUTERSTAMP = socket.gethostname() #os.environ['COMPUTERNAME'] #TODO: [e]Check against http://stackoverflow.com/questions/799767/getting-name-of-windows-computer-running-python-script if doesn't work across systems.

--#TODO: [e] IMPORT here.

--#(16) Essential: [e]

--#(17) Essential: [e] List

--#TODO: [e] Use jflot (or current js render library) to plot line comparison of LoC and TODO items.

--#TODO: [e]Write to PDF.

--#input() #HACK: [e]Added so that results can be read when run from console. Not needed in interactive mode (or if we've run this solely to update the code analysis record).

precommit.py

--#TODO: [e] Run codeanalysis.

predeploy.py

--#TODO: [e] Load test strut and check unit tests.

--#TODO: [e] Run code analysis.

--#TODO: [e] Make local backup.

Desirable

404.html

--

edu_objects.py

--#TODO: [d] add proper assessment.

--#TODO: [d] add other attendance values.

index.html

--

--

--

App Engine Login (MUST USE CORRECT LOGIN EMAIL!)

system.py

--#TODO [d] Add singleton behaviour (not really an issue if never redefined...).

--self.version = "alpha" #TODO: [d] Get version number here.

--self.passedUnitTests = False #TODO: [d] Get version number here.

codeanalysis - Copy (2).py

--# TODO: [d] see page 8 of python data vis.

--# TODO: [d] Write class count metrics.

--# TODO: [d] Write function count metrics.

--# TODO: [d] Write class list of file.

--# TODO: [d] Write function list of file.

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [d] Check for 'conflicted copy' in title of file, and write as a system level bug with list of conflicted versions. [i] Get a diff of changes.

--# TODO: [d] Include results of unit tests (perhaps last unit test run outputs a text file and this reads that, rather than run unit test suite?).

--#(18) Desirable: [d]

--#(19) Desirable: [d] List

--#(24) Other tagging: [other] #TODO: [d] append tag to list of additional tags. Means can get a list of specific todo related to [unittests], for example.

--#TODO: [d] Display other tag list.

--#if line.strip()[0:2] == "..": #TODO: [d] change to option of true/false of showing only file and not path breaks this.

--#TODO: [d]Check against http://stackoverflow.com/questions/2666863/list-to-csv-in-python

--#TODO: [d] Remove all these debug strings.

codeanalysis.py

--# TODO: [d] see page 8 of python data vis.

--# TODO: [d] Write class count metrics.

--# TODO: [d] Write function count metrics.

--# TODO: [d] Write class list of file.

--# TODO: [d] Write function list of file.

--# TODO: [e] output to do list to pdf. [i] ideally colour code to make easier ([i]idealistic = grey, [c]critical = amber (bugs are red!), [e]essential = green, [d]desirable = black)

--# TODO: [d] Check for 'conflicted copy' in title of file, and write as a system level bug with list of conflicted versions. [i] Get a diff of changes.

--# TODO: [d] Include results of unit tests (perhaps last unit test run outputs a text file and this reads that, rather than run unit test suite?).

--#(18) Desirable: [d]

--#(19) Desirable: [d] List

--#(24) Other tagging: [other] #TODO: [d] append tag to list of additional tags. Means can get a list of specific todo related to [unittests], for example.

--#TODO: [d] Display other tag list.

--#if line.strip()[0:2] == "..": #TODO: [d] change to option of true/false of showing only file and not path breaks this.

--#TODO: [d]Check against http://stackoverflow.com/questions/2666863/list-to-csv-in-python

--#TODO: [d] Remove all these debug strings.

Hack

index.html

--

App Engine Login (MUST USE CORRECT LOGIN EMAIL!)

codeanalysis - Copy (2).py

--#HACK: [i] Note the open() calls. This is probably quite inefficient, but shouldn't be an issue until/unless the project gets very big.

--PROJECTROOTPATH = rootpath #HACK: lazy.

--nLoCCode = 0 #HACK: Will include docstrings in lines of code.

--elif tagCheck[0:1] == "[" : #HACK: [i]May not work with strings such as '[tagname]textimmediatelyfollowing'.

--todoList.append(" " + os.path.basename(item[0])) #HACK: [i] Not a good way to get the class filename from the stored path.

--criticalList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--essentialList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--desirableList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--idealisticList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--#HACK: [i]Quick and dirty. If bDoAggregate, then print out the details, else you're writing a record to the log. This whole module should be refactored as a class, but pretty low priority.

--print (" HACK Items: " + str(hackCount))

--#HACK: [i]'entry' values mapping to columns is not consecutive. e.g. hackCount = no, LocCode != no + 1.

--#HACK: This works, but it means that we can't see where the TODO came from.

--todoList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--criticalList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--essentialList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--desirableList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--idealisticList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--sLine = "" + line #HACK: [c] Seems to be writing back the stripping to the list.

--#raw_input() #HACK: [e]Added so that results can be read when run from console. Not needed in interactive mode (or if we've run this solely to update the code analysis record).

codeanalysis.py

--#HACK: [i] Note the open() calls. This is probably quite inefficient, but shouldn't be an issue until/unless the project gets very big.

--PROJECTROOTPATH = rootpath #HACK: lazy.

--nLoCCode = 0 #HACK: Will include docstrings in lines of code.

--elif tagCheck[0:1] == "[" : #HACK: [i]May not work with strings such as '[tagname]textimmediatelyfollowing'.

--todoList.append(" " + os.path.basename(item[0])) #HACK: [i] Not a good way to get the class filename from the stored path.

--bugList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--hackList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--criticalList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--essentialList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--desirableList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--idealisticList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--#HACK: [i]Quick and dirty. If bDoAggregate, then print out the details, else you're writing a record to the log. This whole module should be refactored as a class, but pretty low priority.

--print (" HACK Items: " + str(hackCount))

--#HACK: [i]'entry' values mapping to columns is not consecutive. e.g. hackCount = no, LocCode != no + 1.

--sLine = "" + line #HACK: [c] Seems to be writing back the stripping to the list.

--#HACK: This works, but it means that we can't see where the TODO came from.

--todoList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--bugList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--hackList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--criticalList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--essentialList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--desirableList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--idealisticList.append(" " + os.path.basename(item[0])) #HACK: [i]Not a good way to get the class filename from the stored path.

--#input() #HACK: [e]Added so that results can be read when run from console. Not needed in interactive mode (or if we've run this solely to update the code analysis record).

precommit.py

--#HACK: [i] Exists because Github for Windows can't execute precommit hooks :/ .