

M8010: Software Design (2003-2004)

Assessed Exercise 3

Software Architecture Design and Analysis

(Spelling Check)

By Students:

Tom Morton – 02087459

¡nadie más!

Definition of Design Problem

A problem our customers have requested be resolved by means of a conspiracy of software designism is such that for such purposes as exist, requirement has been found such that a piece of text of dubiously correct spelling contained within some digital medium is desired to be checked for that the construction of words within said piece of text be conformant to some dictionary also containant with said system of digital processing, and logical, physical and metaphysical calculations. The preconditions of the former be that said text be encoded in such ways as to be conformant to design assumptions agreed in advance with all parties, and the preconditions of the latter proscribed to be sufficiently complex to model the system, while sufficiently simple to allow understanding during perusal by the former.

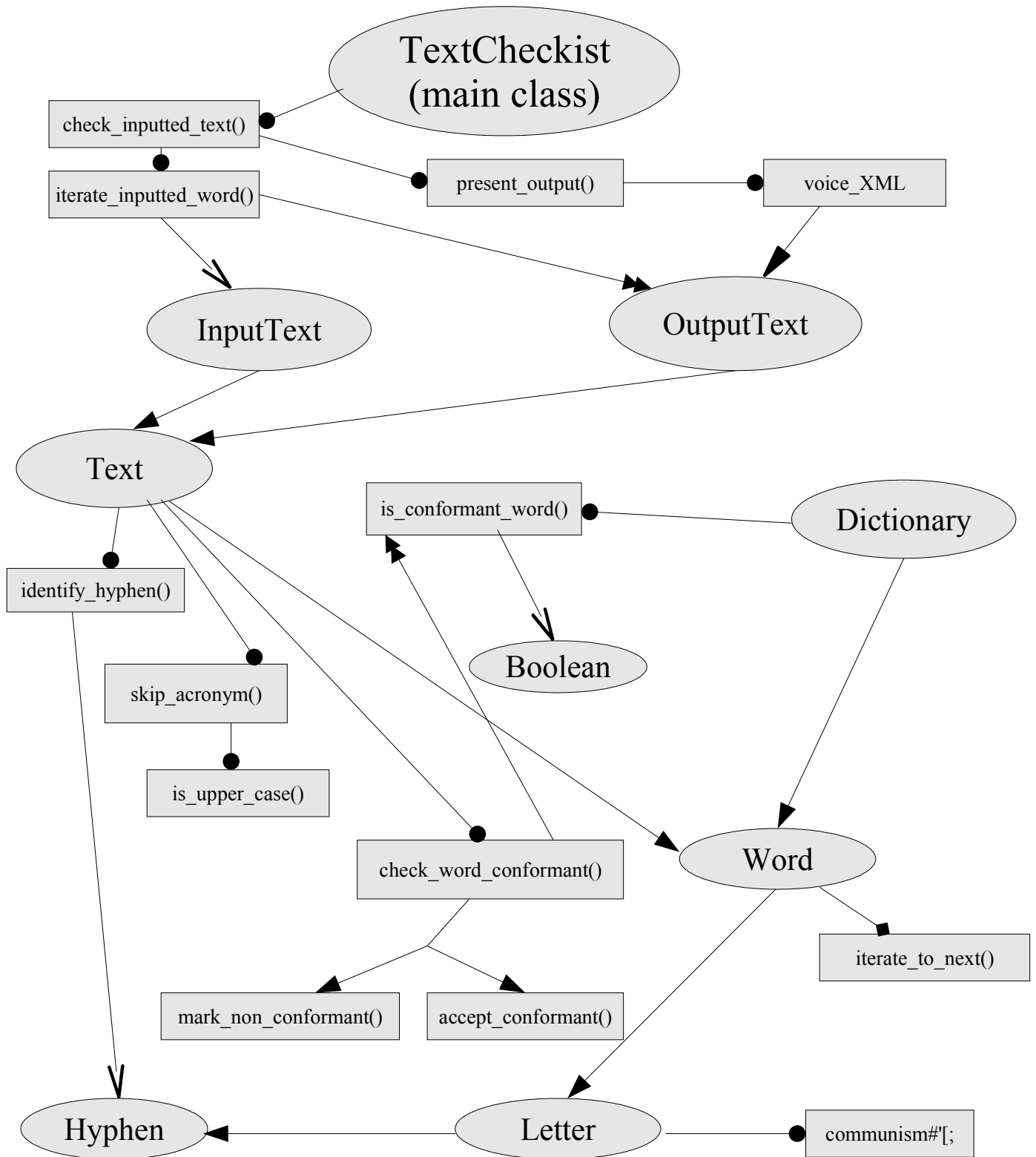
When during the course of its functioning, said system does not associate encountered words of inputted piece of text with words, acronyms or strings contained within the dictionary, as defined above, then the system will proceed to mark said piece of text inputted into system with a mark defined here to be '**'. That is, **, excluding the ' marks.

Additionally, inputted text be definant to have a *width*, and requirement exists in order than words encountered within such text existant at the extreme right of a line, terminated with a hyphen, be considered a prefix of the word to be found existant at the extreme left of the line to be found below the line considered here. Conformancy with word lists within the dictionary will be performed on this concatenated, dehyphenated conglomeration. Words found neither at the extreme left nor at the extreme right of a line, and contaminated with hyphen, shall be required to be split into such parts as exist before the encounterment of this hyphen, and components located in extremities beyond the righthand boundaries of said hyphen. Such components as defined in the previous sentence, be required in advent of encounterment be processed individually and in their whole in manners ascribed previously for processing of words. To clarify, such as it may be required, adventation of hyphenation must be met with concerted deliberation in order that contamination of output be prevented in all cases, and subject post-conditions be insinuated to be in a state of conformancy defined previously.

Alongside the requirements to comprehend correctly the uses of hyphenation, inputted text also may be containant with acronyms, which may be defined here and forevermore as concatenations of upper case latin characters of length greater than one, and non-containant of hyphens. When such systems as defined above exist and are in a state of processing of inputted text, requirements be that acronyms located within such input should signify themselves unto the system, and be recognised, and that on recognition and the completion of such recognition mentioned previously, the acronym resultant in this input stream be skipped, this action being defined as the action of doing no action, and moving

on to the next position within this input stream in order to perform another action, either another action of skipping, as defined here, or such as is required.

First Design: Object Orientated



As can be seen, the Object Orientated Design observable in the previous page can be observed to be an advanced Object Orientated model, suitable for, but not limited to, implementation in languages containing such features as are required for suitability in the implementation of advanced Object Orientated modelled systems, for examples, Javas. The oval shaped grey things are classes and the rectangloid grey things are member functions of their respective functions, as indicated by arrows. TestCheckist, being as it is the main class, picks to, in order to achieve desired work, work, to the processing of the individual words via its member function (or shall we say 'method', to be feisty) check_inputted_text(). This function should be after consideration be called by such functions as exist within the chosen language for example Javas which construe entry into the program execution environment as provided by a suitable unix kernel such as linux. When the function runs, its primary function, but not to say only unless we are feisty would in any case and clearly be, calling in a while loop the iterate_inputted_word() function, which by this magnificent design also resides within the corpus of the dominant TestCheckist function. On subsumption of such calls as are made, said function calls in turn suitable methods as are required by the specification defined previously, in order to correctly identify and in turn and with due discourse, place into the output buffer the checked words, or such output as is generated on occurances of some mystical error, via the function present_output(). Once all has been processed as in seen fit by such as this model does do, it does do it. And into the voice_XML we go, not before time. It can clearly be found from this analysis to be certain, as jesus would say, if we can for a moment drag ourselves from dry intellectualism to ponder such questions as why such a system would be devised in the manner shown, it is an Object Orientated Approach, and can we see that all things in this system shown above are clearly objects.

Sometimes the requirements of objects systems are that the system which models objects himself is containant with objects, hypocritically so in any event, and without further discourse we can present for consideration the fullness of specified requirement within this.

Analysis of Design Modifiability

Within the course of use, the customers who did specify and purchase this system find that additional specifications are required additionally to be needed. Such desires as they have are containant with demands as follow: In the advent of a non dictionary conformant word is encountered, not limited to, but subject and extended with, proceeds to identify such words as are non-conformant, and to provide dictionary conformant alternative word, in order that users unsure of their desire to say, will say conformant with the dictionary's wish. In example, on encounterment of a word such as 'blorek',

systems which have onto their board taken these new requirements would, from lists of conformant words axiomatically defined to be words containant with the dictionary, found and presented a word such as 'blorn', which is dictionary conformant and likely to be inevitably the word the user intended to use originally. This should be presented in such a form as where previously ****blork**** was containant with the output, now such output will you know as, ****blork (blorn)****, and righteously if implemented conformantly.

Additionally to the above presupposition, a requirement requires that users, defined as the customer's friends who subjected software design methodology to this, would be, is required that they can add and append words to the dictionary, such as they desire, and that it is stored within a file, and so on. Onto the challenge to implementation!!!

Scenario Evaluation!

No.	Description	Component	Change
1	To when non-conformant words are found to indicate with words of conformant nature which are alternative.	Text class, specific methods such as the <code>check_word_conformant()</code>	TRIVIAL, make it mark additionally with alternative word, which is found easy with change to the <code>is_word_conformant()</code> of Dictionary class. Easy
2	Allow liberally users to change if they so desire to add words to the dictionary, and to store in an external files these stuff.	Only the Dictionary class methods, due to smart design customer satisfied.	Adding methods to Dictionary, such to be that users can add stuff like the box to the left says.

- The arrow indicates sharing

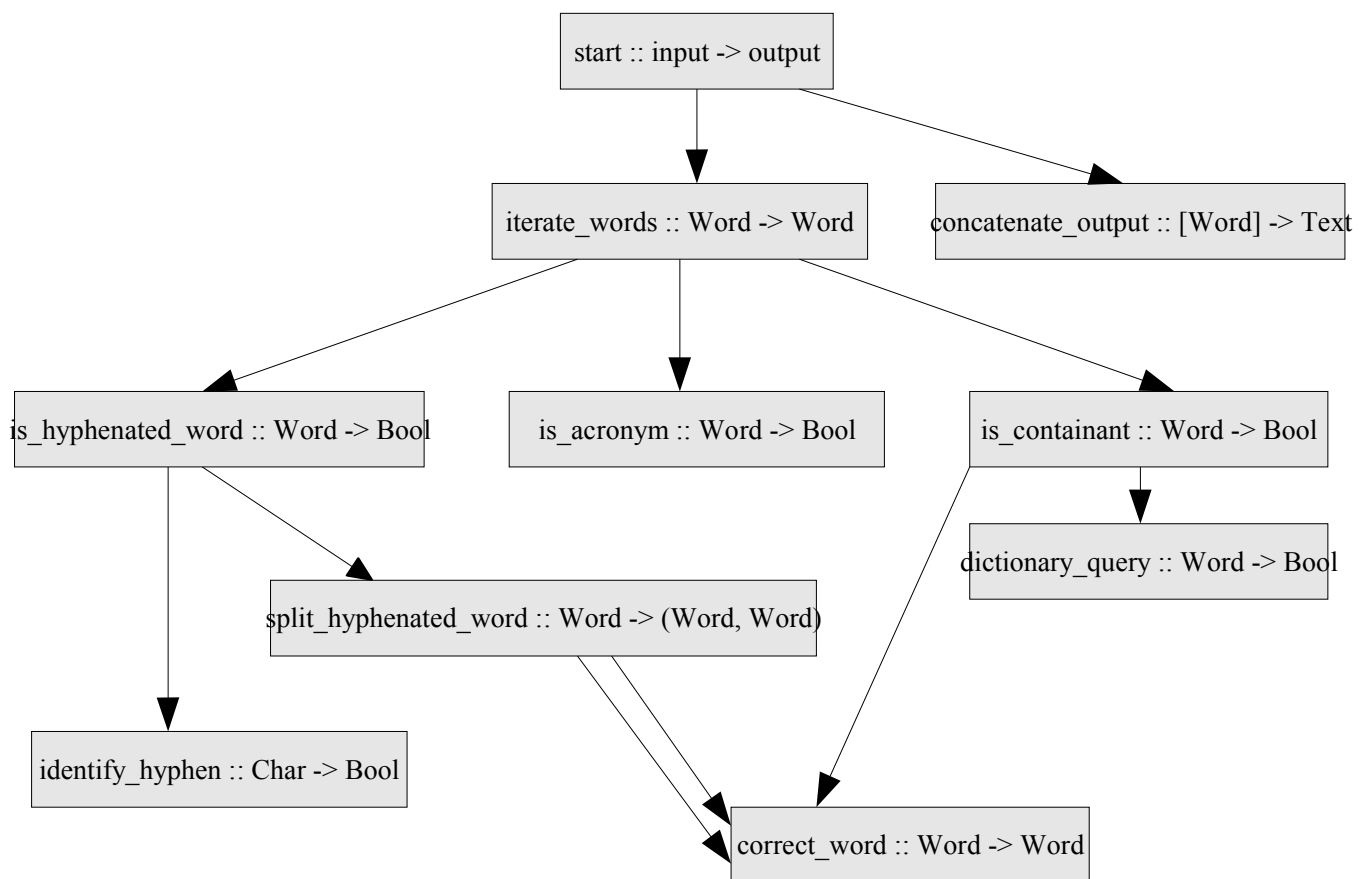
Clearly the Objected Oriented design superiority makes this design be able to make any additions without fail, and find ourself with a customer so happy that this progress may be made simply and easily with the will to do so only, and some other things,. Clearly this design is very excellent, and so

with heavy feelings we must move to more problematic grounds of suffering. Next page is here.

Second Design: Functional

A functional language is sometimes called something like lisp or askell, which has functions who in the calling of things control flow is controlled by termination, non-termination and recursion of functions, with single return values and typically, though not always, multiple argument values.

Functional approach seems appropriate to this dictionary system due the the very nature of functions: they do things to data and return something different, which is precisely what we require in order to achieve things specified in the specification. The syntax of functional language used in this design, for the sake of your safety and comfort, is askell.



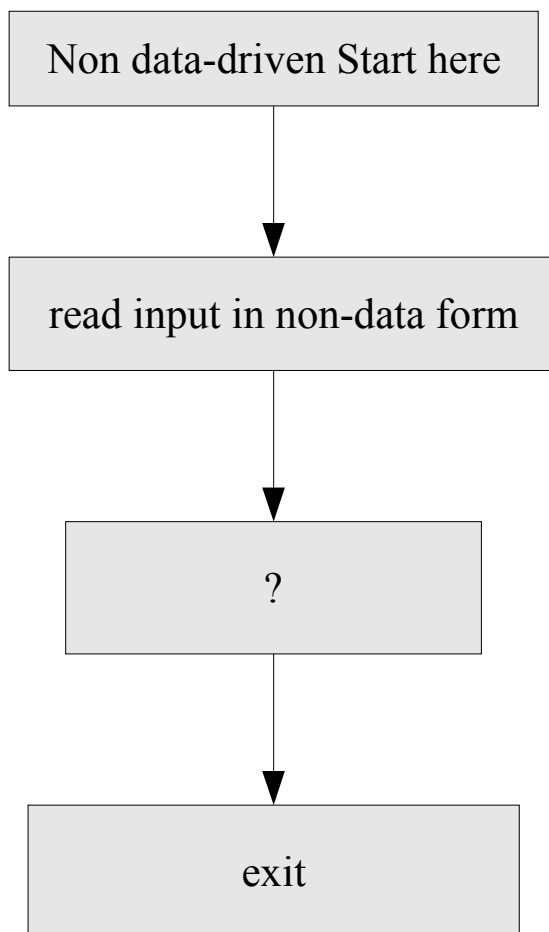
In order to understand this you must think that basically it is a call graph, it isn't. This system can be described as a data-driven systems, and as such some types have been defined, Word, which is containant of a single word which we have interest in dealing with or doing some experimentations or such on it, Char is just one letter likely any in the alphabet but defined to be not able to know immediately. In essence, it could be plotted to implement Word as maybe a list filled with Chars, which will fulfil just fine without spaces. Text is for containing the entire input given to the system when it

begins to run, in `iterate_words`, and similarly can be and is used to store the returned value, which might be the output. It can be followed through more or less where the arrows point, by an astute observer, but laymen will need to be guided like so, clearly then we call `iterate_words`, which goes through by some means all the words in the input, and does stuff to them, storing the value for returning, which will according to the specification and rigorous design, changed.

It is unnecessary to consider the design modifiability of this design.

Fourth Design: Non data-driven

It is often said that all computer-based systems make use of data, in one form or another. Systems are sometimes referred to as data-driven, by computer science academics who are in the know. This system can be defined by similarly a lack of data-drivenness. Simply put, the system's workings are specified, and to see such a system working which isn't data-driven. Here it is.



It is not an issue, to explain the working, because data isn't involved. Suffice to say, with clear design it performs comparably to similarly specified object-oriented systems of similar functioning, or other systems as you wish such as Javas.

Analysis of Design Modifiability

Since the non data-driven system can be simply presented while still functioning optionally, and if it may be said, to be feisty, clearly the system can adapt proficiently to the problems encountered and passed on not to say created or indeed necessitated by such things. In terms of usability, and in fact also connectivity, such as can be applied to the design modifications specified previously, which hardly bare repeating. Data-less systems analysis of design modifiability is considerably trivial with respect to the former demand, as workings of the system are hardly to be described with 'opaque', since absence is the order of the days. Concluding then, in depth studies such as this, while considerably worthwhile in order to fully appraise the valueability of features such as reusability, shouldn't be simply a blunt tool with which to condemn entire sections of the specification. Sufficient care must be taken in order than a full evaluation of nascent modifiability conventions and indeed standards, in taking into hand the foremost objective of design modifiability analysis, the consideration binds us to a growing discontent and infact hatred, and the moistening of punched cards. Non data-driven systems drive a wedge between the four pronged attack of functional, imperative, object orientated, symbolic, logical and social programming. Clearly the requirement that databases might be added to by users poses a moral question: dare we play god? Some might say that a sufficiently transparent data-less implementation might arouse suspicion, and indeed without transforming the very nature of the computing, once cannot grasp the profound vapid smile of counterfeited eternities. The question of maintainability crushes with each pounding blow the notion of amendments to specifications. Approached correctly, it is clear that maintainability, darwinism and usability are interlinked. Non data-driven software gives us hope for the future. This may seem a rash proclamation, but design transparency and accessibility, conformability and security lead us forward to a darker time, when elves lived and stuff.

Comparison of the designs

When deciding which system to furnish your customer with, it is necessary to place these various systems in competitive examination, into a pit of fighting if you will. The designs on the first count must be considered by concrete objectivity, and what the diagrams of their respective in-depth design specification diagram implementations look like. In the specific case of the Object Orientated system, some of the boxes are round and some are square, which suggests complexity not found in the functional model, where all the boxes are square. Similarly, the boxes in the non data-oriented model are all square, again, suggesting inherently a greater degree of simplicity than that found and created to be noted to have in the Object Orientated systems one. It is no generalisation to say that mostly all Object Orientated systems have similar things, examples include Javas.

In terms of usability, and considering this point in isolated, with respect to the idea of how much of usability they have, we can examine that the Object Orientated one has objects for things in the system, which is intuitive to humans because we are surrounded by objects the whole damn time. This cannot be said for the data-less model, which confuses and bemuses in equal measure if the measure of usability might be measured in object-metaphor, which it might. The functional model is usable despite the absence of objects, as we saw earlier, this can be attributed to certain features or attributes of the system we have designed, namely some of them we have examined here and above also.

Moving this gripping debate onto efficiency, one must define this term in order to progress with such an analysis. The data-less model can be said with no lack of precision to be maybe about as efficient as the others, which are, respectively, efficient to a fair degree and efficient in adequation.

Clearly the field of software design has a great deal to teach us, and many interesting revelations to bestow on us. It is often the case that when one embarks on a software project, the issue of design is neglected. The rush to code concrete implementations stifles research into new '-ility', and '-ivity' words, and denies the programmer the freedom to create diagrams with boxes and lines and words, which mean things. How will management ever be able to comprehend the process of software writing without these things?