Task 19 : software architecture

WebPA Project

An Online Peer Assessment System for HE

Task 19

Software Architecture

Author: Nicola Wilkinson

Version: 1.3

Project lead by:
engCETL, Loughborough University

Project partners:
The University of Hull
Higher Education Academy, Engineering Subject Centre
Higher Education Academy, Physical Sciences Subject Centre

Executive Summary

This document will cover the software architecture of the current WebPA system in use at Loughborough University. Within the document, the definition of the term software architecture is covered, as it has various meanings dependant on which angle is taken. The architecture description language is also identified. This will be used in the rest of the document to aid in the understanding of the different software architectures described.

In order for the reader to understand this document they will need to have some familiarity with the concept of software architecture. Time has been taken to explain the concept and the main areas of software architecture that will be covered within this document. In order to read and understand the diagrams used to describe the software architecture it is useful for the reader to understand the main elements of UML. The constructs of UML will not be examined and are beyond the scope of this document. However, all efforts have been made to explain the diagrams to the user.

Content

Table of Figures

Introduction

Software architecture can be taken as having a number of different meanings dependant on which perspective taken. There are not only descriptors for the physical architecture but also methods and languages for describing the architecture of the physical software at component and theoretical levels. To assure that the reader can fully comprehend the software architecture and appreciate the system as a whole, the definition of software architecture is decided and the basic principle elements of the architecture are explained.

Definition of Software Architecture

As has been mentioned there are a number of definitions to what a software architecture is and how it's described. In general there are two groups of definitions. The first group are the classic descriptions and the second are modern. Figure 1 encapsulates the classic definition, while Figure 2 encapsulates the modern definition, although commonalities can be drawn, the definitions are seen as being separate. Within this document we will use the classic description, as it can be aligned with the Architecture description language that has been used.

> An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these structural and behavioural elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition.
> (Kruchten: The Rational Unified Process. Also cited in Booch, Rumbaugh, and Jacobson: *The Unified Modelling Language User Guide*, Addison-Wesley, 1999)

Figure 1 - Classic Description of Software Architecture

> Architecture is defined by the recommended practice as *the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.* This definition is intended to encompass a variety of uses of the term architecture by recognizing their underlying common elements. Principal among these is the need to understand and control those elements of system design that capture the system's utility, cost, and risk. In some cases, these elements are the physical components of the system and their relationships. In other cases, these elements are not physical, but instead, logical components. In still other cases, these elements are enduring principles or patterns that create enduring organizational structures. The definition is intended to encompass these distinct, but related uses, while encouraging

more rigorous definition of what constitutes the fundamental organization of a system within particular domains.

(ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems)

Figure 2 – Modern Description of Software Architecture

Architecture description languages

Architecture description languages (ADLs) are used to describe the software architecture. Over time a number of different ADLs' have been developed, but no single language has been devised. The Unified Modelling Language (UML), which was developed as a standard, to model systems and software, will be used for the WebPA software architecture. The reason behind this decision lies in the fact that UML is widely understood, where as the other ADLs, such as Acme, are not.

Software Architecture

Software architecture is organised in to views which can be seen as different types of blueprints, similar to those in traditional building architecture. Views can be seen as instances of a viewpoint, where the viewpoint describes the software architecture from a specific perspective. These different perspective views can be broken down into;

- functional/logic view
- development/structural view,
- concurrency/ process/thread view,
- physical/ deployment view,
- user action view.

These different perspective combine to describe the software architecture, each method is aimed at a different stake holder. The stake holders are not explored as part of this documentation.

Functional View

The functional view of the system describes how the whole software system can be broken down in to sub systems or modules, based on the functions that are to be carried out. To show the origins of the functions, a series of use case diagrams are included. The first of the diagrams shows the actions that the academic will carry out on the system, as can be seen in Figure 3. The second of the use cases is for the view point of the student, as shown in Figure 4.

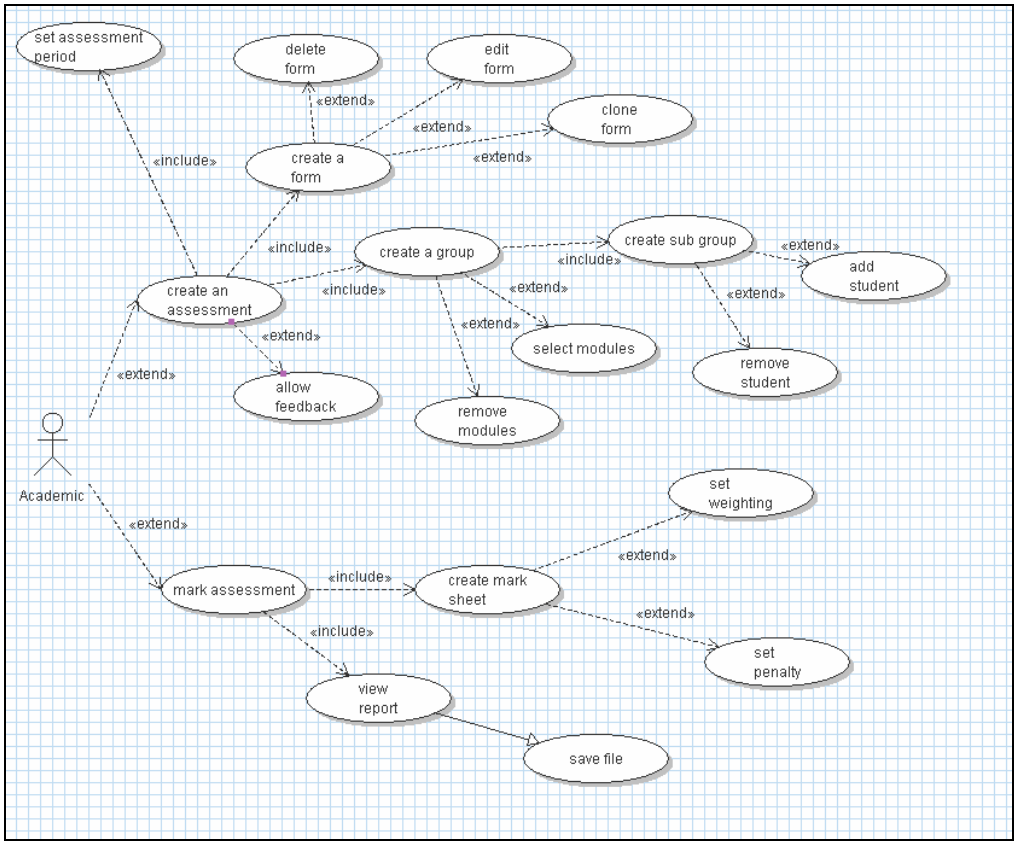

Figure 3 - Use case diagram from an academic perspective

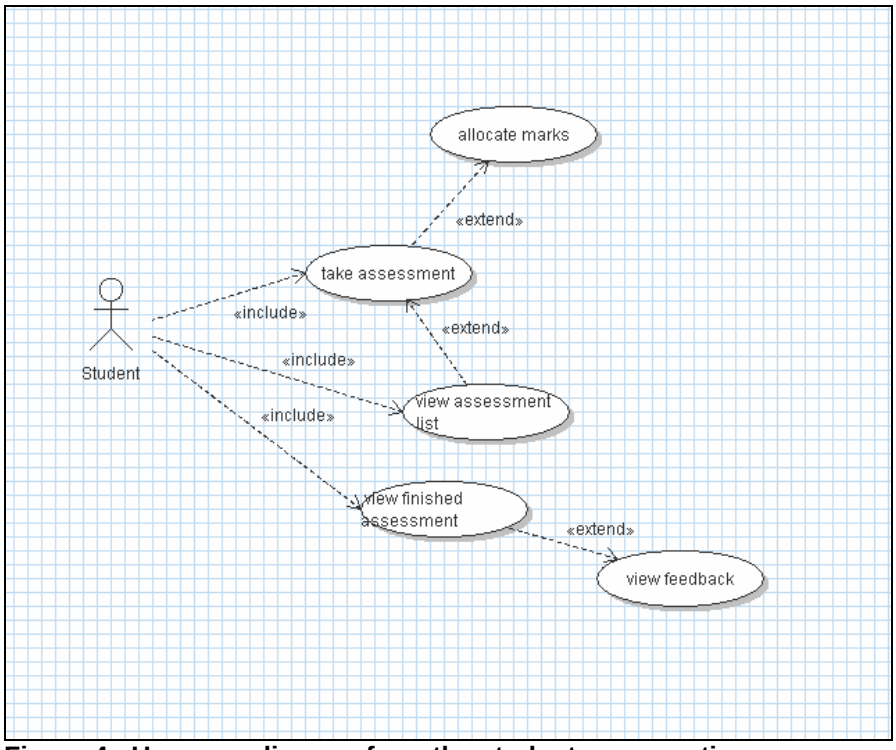

Figure 4 - Use case diagram from the students perspective

Structural View

There are two ways of interpreting what a structural view is. The first is the UML method, where the modules defined in the 'Functional View' are

represented as classes. In order to show this break down a class diagram has been developed and is shown in Figure 5. The full break down of the classes within the system can be found in Appendix 1, where the attributes, methods and inheritance are fully explained.

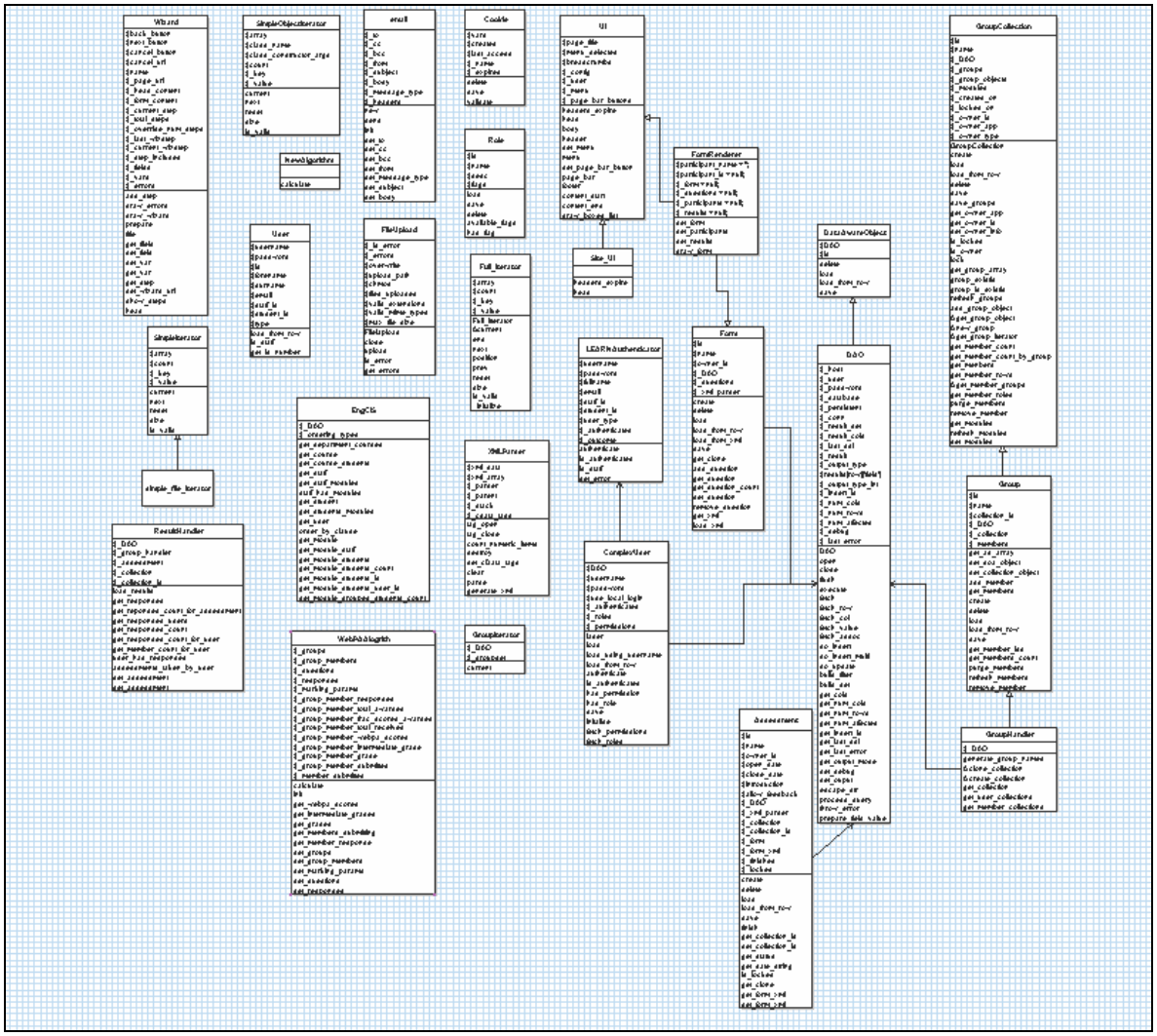

Figure 5 - Class diagram for the WebPA system

The second interpretation of the structural view describes the break down of the system into the files, directories and libraries. This view allows the file structure of the software to be viewed. This representation of the structural view is not explored as part of this documentation, however, an example is shown in Figure 6.

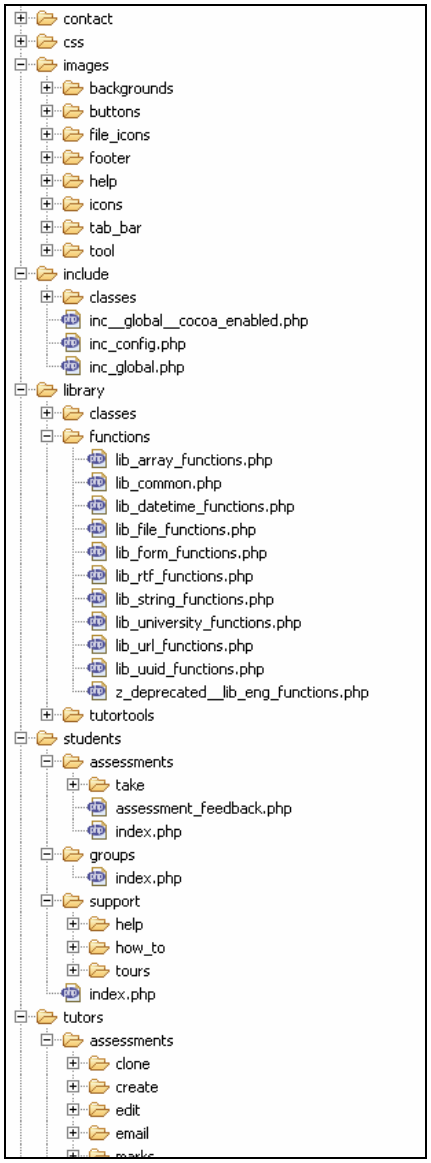

Figure 6 - Physical structure of the WebPA system

Concurrency View

The concurrency view allows for the data flow and the control flow of the software architecture to be documented. The data flow shows the units or modules of the programme and the data that is transmitted. Originally this information was presented in a data-flow-diagram (DFD) and was integral to the SSADM[1] methodology. Within the UML methodology data flow is shown using activity diagrams.

The control flow documents the process where by a module of the software will activate a functional behaviour. This control flow is predominately used to document the timing and ordering of operations that occur in the software. Within this documentation will only briefly use this flow of information to explain the process for the completion of a behaviour the system pertains to.

[1] SSADM (Structured Systems Analysis and Design Methodology)

The representational diagram for the control flow within UML is the state chart diagram as shown in Figure 7.

Assessment Process Triggers

The assessment process within the WebPA system is one of the most crucial components. There are a number of conditions that have to be met through the process to enable the next stage to be completed. There are also a number of conditions that have to be met before the process can begin, these include; the creation of the assessment form and the creation of the groups. Once the pre-requisites are met then the creation of the assessment can take place, this in turn triggers the first state that the system reaches. The first state is assessment *pending*, where the assessment has not yet reached the date where it is accessible by students. Once the start date is met then the state of the assessment changes to *open*. The assessment is then in the state of *open* until the end date is reached. When the end date is met, the assessment reaches the state of *closed*. The assessment will remain *closed* and must have reached this state before the next state of *marked* can be reached. Once the *marked* state is reaches the process for the assessment is complete. This means that the assessment will not go through this process again. This whole process is illustrated in Figure 7.

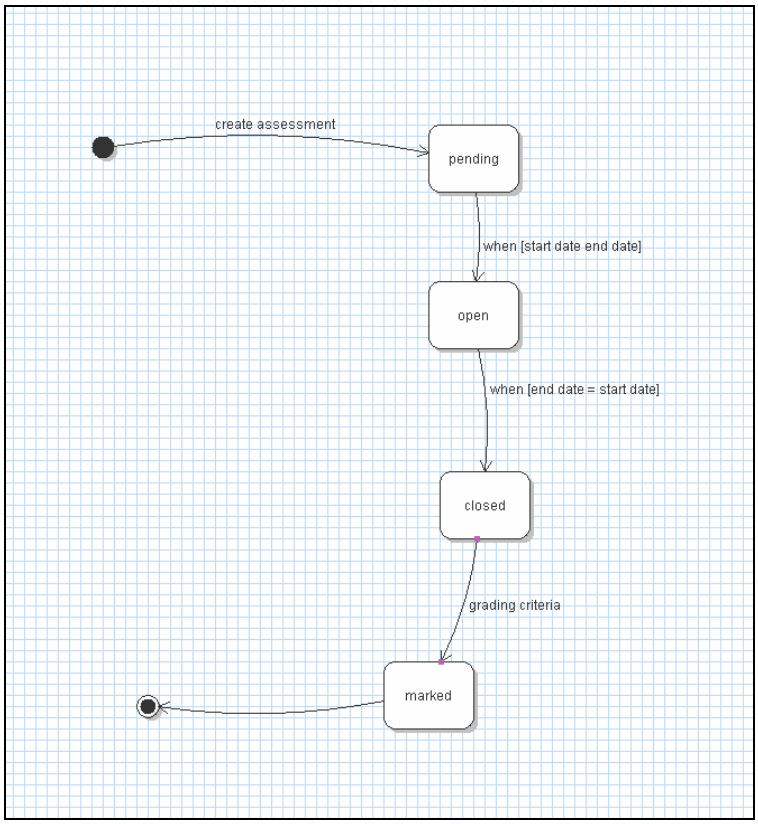

Figure 7 - State chart diagram for assessments

Authentication Process

Authentication is an important aspect of the WebPA system as this identifies the users, to be either staff or student. This in turn controls the view of the WebPA system that they receive. Within WebPA 0.9 all authentication is run

through the class LEARNAuthentication. The process which is carried out is documented in Figure 8. The main actions are that the system needs to authenticate who the user is to ensure the correct view on to the WebPA system is shown, and a flag must be set to identify the user as staff. If the user can not be authenticated by the system then the process reject user is triggered.

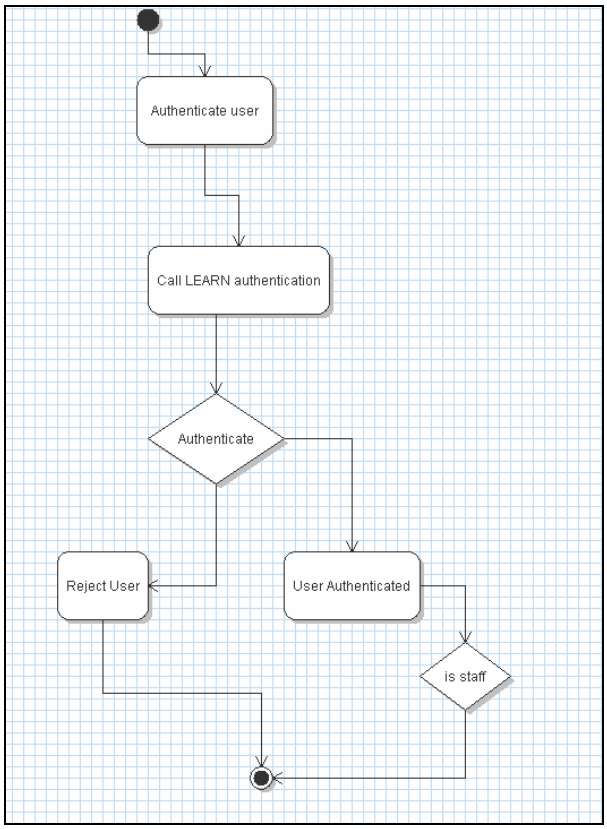

Figure 8 - Activity diagram showing the authentication process

Physical View

The physical view of the software architecture allows the dynamic aspects of the system to be documented. This includes the communication between components of the software as tasks and the operations that are executed. The physical view is often documented as a 'Model View Controller' (MVC) diagrams and in UML are represented as interaction diagrams. In order to document the physical view of the software architecture, elements of the hardware and the application architecture must be examined.

Application Architecture

The application architecture for the WebPA system is a layered architecture. There are two layers within the system, The first layer is the database layer. The second layer is composition of the presentation and the application logic. This two-tier architecture is shown in the diagram in Figure 9, against the more well known three-tier architecture.

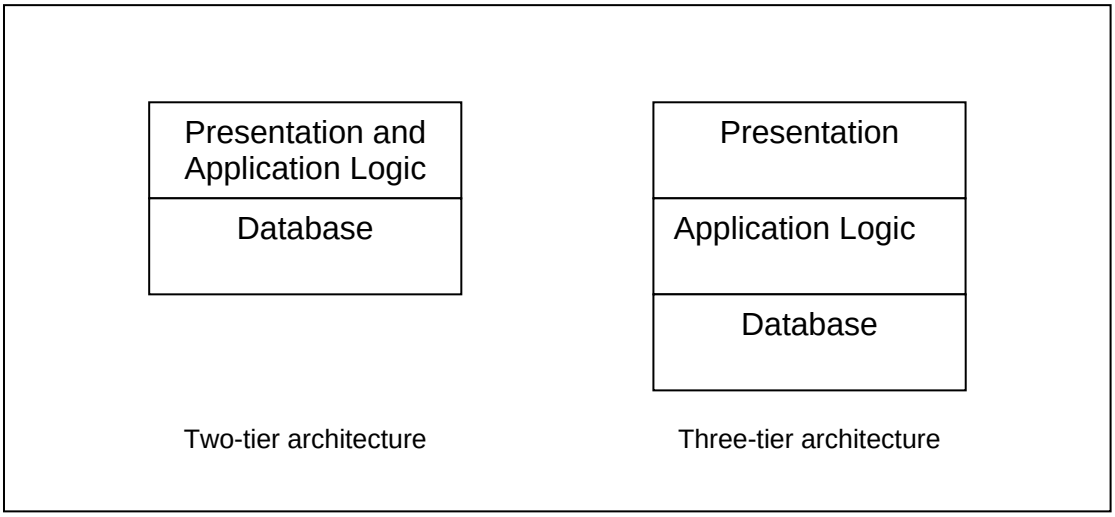

Figure 9 - Application Tier Architectures

Physical Architecture of the WebPA System

In order to properly explain the software architecture of the system the physical architecture must be understood. This will help to clarify 'distributed computing' nature of the system. Within Figure 11, the physical application server model for the WebPA system is shown. There is a second server which is involved, however is this architecture is not shown. This server contains the information related to the student records, and is beyond the scope of this document, and will not be described. However, the reader must be aware of the second system as components will be described as part of the software architecture. A representation of the two systems and their relationship is shown in Figure 10.

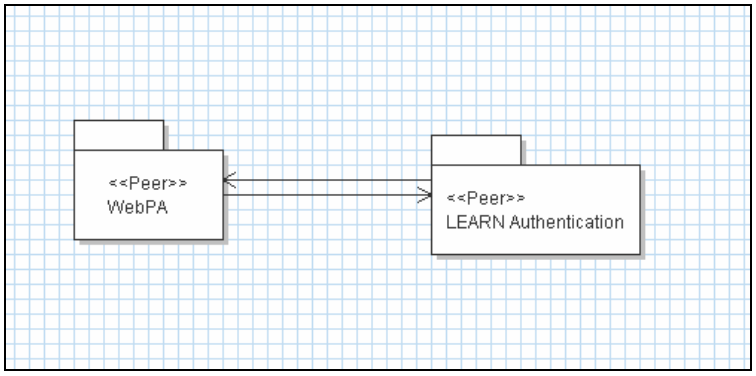

Figure 10 – Peer to Peer Interaction for User Authentication between Servers

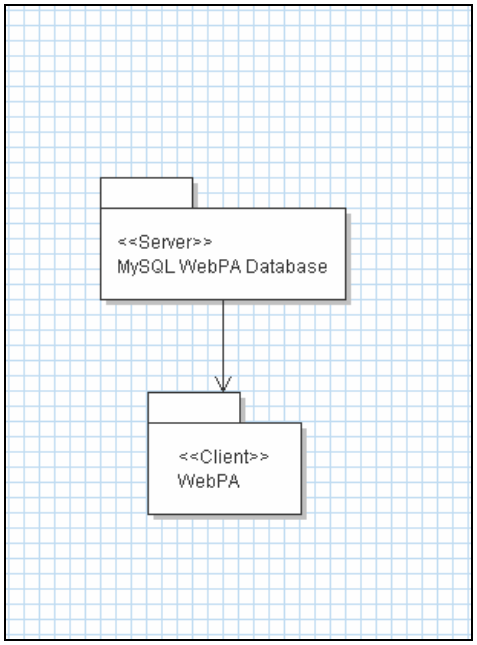

Figure 11 - Client Server interaction between WebPA and the database

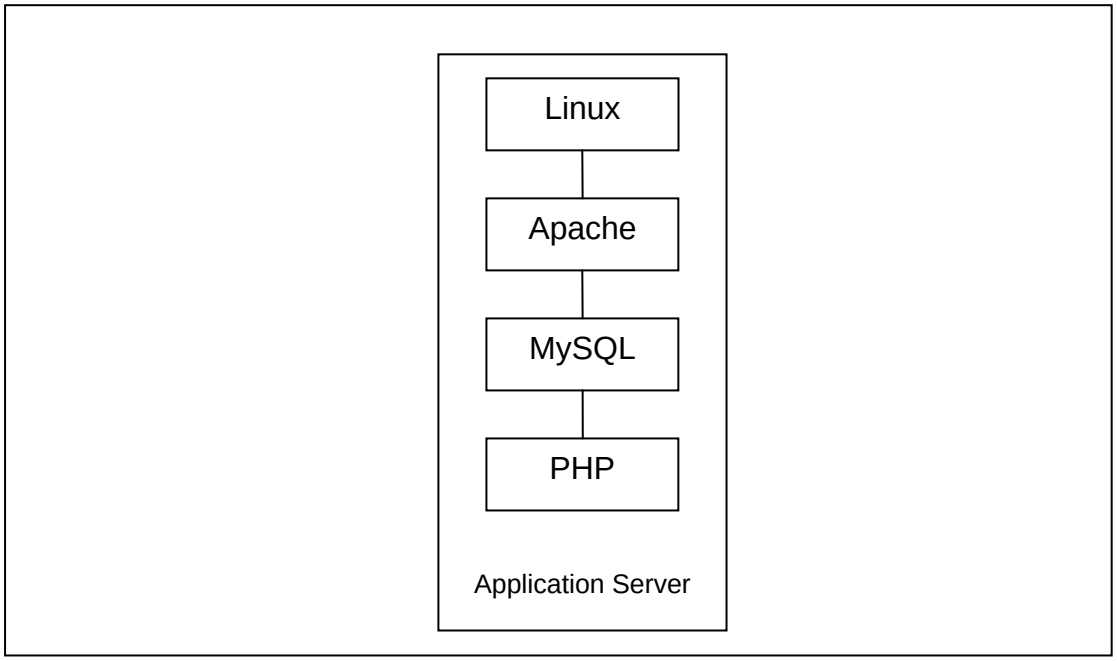

Figure 12 - Hardware Architecture

User Action View

From the academics point of view there are a limited number of actions that they can complete in the system. For each activity, the action is explained briefly and accompanied by an UML activity diagram.

Create forms

The academic using the system needs to be able to create forms that they will use as the assessment. Within the current WebPA system there are two ways of creating forms. The first method is to create a new form from scratch, the

second is to clone a form. In each case once the form is created, criterions can be added before the form is considered complete.

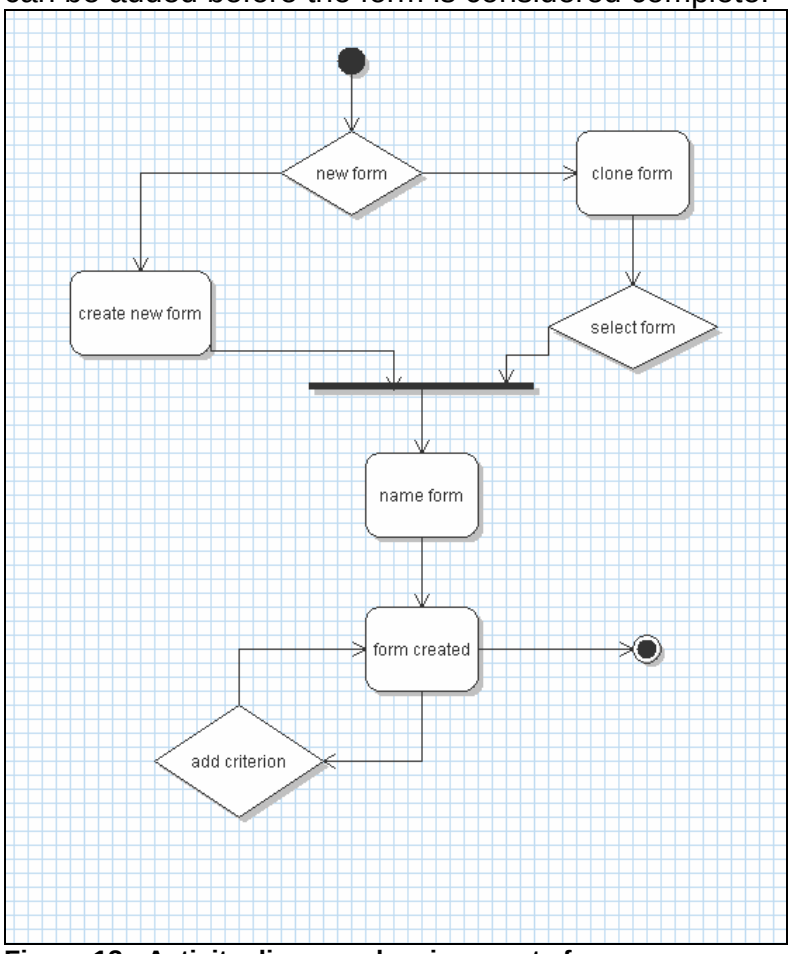

Figure 13 - Activity diagram showing create form

Create groups

Academics need to be able to create groups within the cohorts of students they teach. The groups are formed out of the students who are members of a module. The academic must select to either create a new group or to clone an existing group.

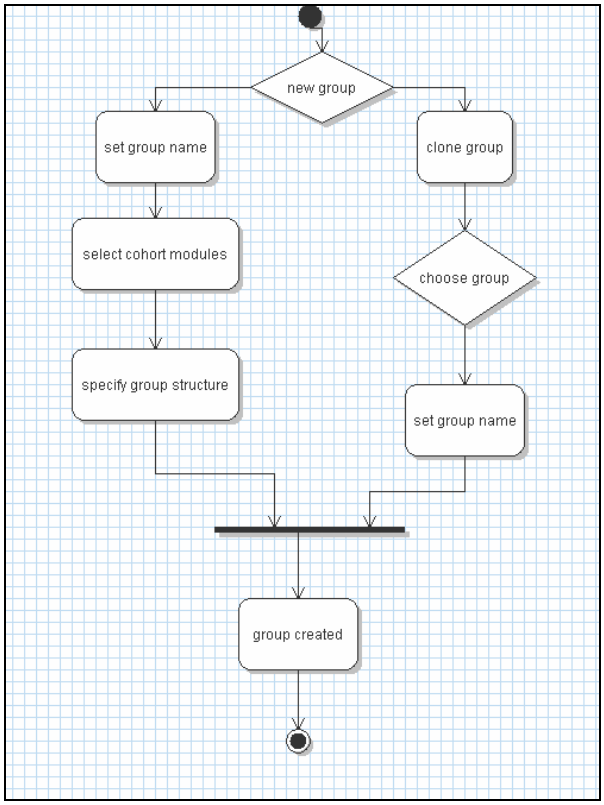

Figure 14 - Activity diagram for creating groups

Create assessment

Academics must be able to create an assessment. The assessment brings together the group information and the forms that have been created. As part of this activity the academic must chose the dates between which the assessment will run, as well as if any feedback to the user will be presented.

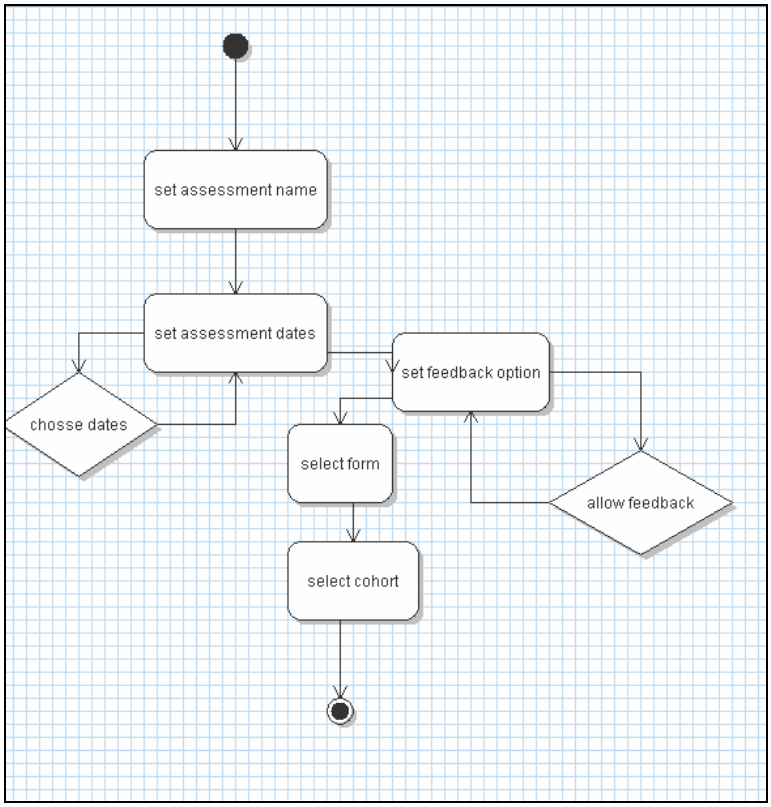

Figure 15 - Activity diagram for creating an assessment

Edit form

Once a form has been created then there needs to be the opportunity for the academic to edit the form, to either add more criterions, or to remove the form completely. In Figure 15 the basic actions for the form are shown. There are no constraints on the forms when they are part of the assessment.

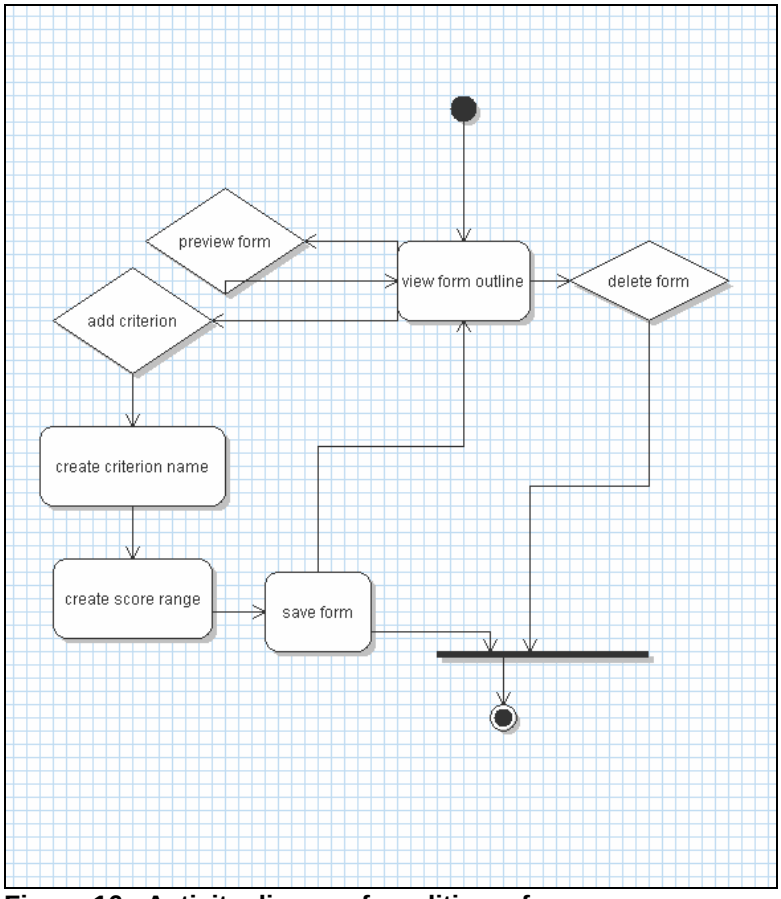

Figure 16 - Activity diagram for editing a form

Edit groups

An academic can edit the overall group that they have created from modules. They are also able to view each of the sub groups and alter the students that are comprised within the group. These actions are shown within the activity diagram - Activity diagram showing editing groups.

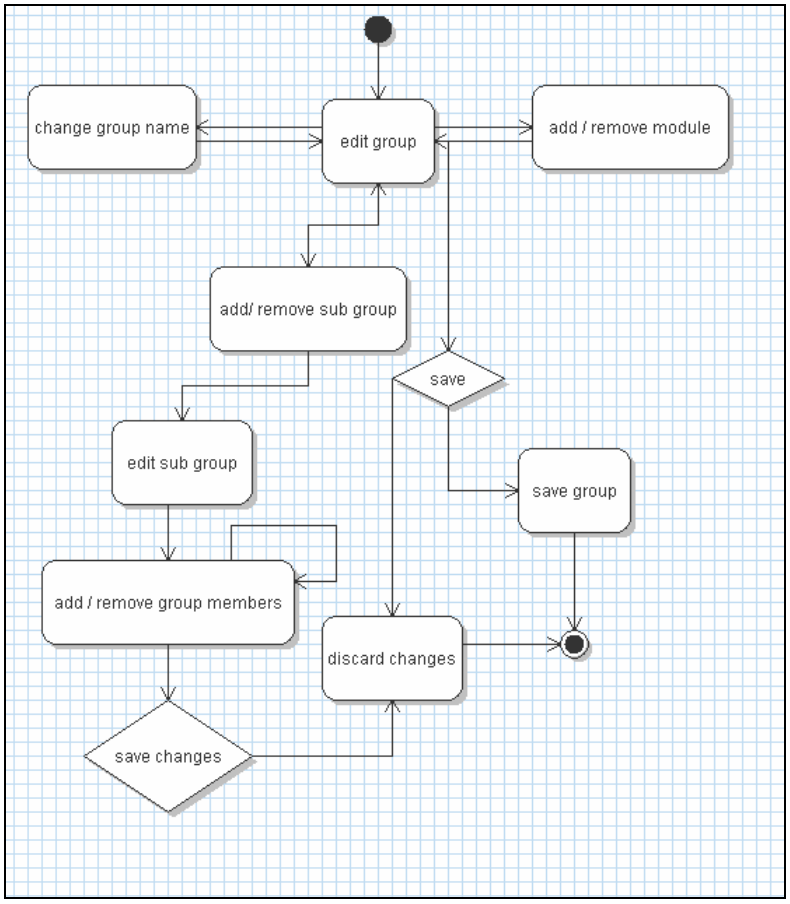

Figure 17 - Activity diagram showing editing groups

Conclusions

It is impossible to show every point of view and process, that can be carried out in the existing system, due to project constraints. However, this document covers the software architecture to a detailed enough level to allow the reader to understand the system. It has not been possible within this document to record the decision processes for the software architecture and it is accepted that as a consequence there may be missing or misinterpreted information within this document.

References

Brad J. Cox, Andrew J. Novobilski: Object-Oriented Programming: An Evolutionary Approach. 2nd ed. Addison-Wesley, Reading 1991 ISBN 0-201-54834-8

Bertrand Meyer: Object-Oriented Software Construction. 2nd ed. Prentice Hall, 1997.

Clemens Szyperski: Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Professional, Boston 2002 ISBN 0-201-74572-0

George T. Heineman, William T. Councill, Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Professional, Reading 2001 ISBN 0-201-70485-4

IEEE, *IEEE Recommended Practice for Architecture Description of Software-Intensive Systems*, IEEE Recommended practice for architectural description of software-intensive systems*, E-ISBN 0-7381-2519-9, ISBN 0-7381-2518-0,*

Dale Dougherty (January 26, 2001). LAMP: The Open Source Web Platform., [Online] Available at: http://www.onlamp.com/pub/a/onlamp/2001/01/25/lamp.html

Carnegie Mellon University, 2007, Published Software Architecture Definitions, [Online] Available at: http://www.sei.cmu.edu/architecture/published_definitions.html

Carnegie Mellon University, 2007, The Acme Project, [Online] Available at: http://www.cs.cmu.edu/~acme/

University of Waterloo (2006). A Very Brief History of Computer Science., [Online] Available at: http://www.cs.uwaterloo.ca/%7Eshallit/Courses/134/history.html

IEEE Transactions on Software Engineering (2006). Introduction to the Special Issue on Software Architecture. [Online] Available at: http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/trans/ts/&toc=comp/trans/ts/1995/04/e4toc.xml&DOI=10.1109/TSE.1995.10003

SEI (2006). How do you define Software Architecture?. [Online] Available at: http://www.sei.cmu.edu/architecture/definitions.html

SoftwareArchitectures.com (2006). Intro to Software Quality Attributes. [Online] Available at: http://www.softwarearchitectures.com/one/Designing+Architecture/78.aspx

SEI (2006). Origins of Software Architecture Study. [Online] Available at: http://www.sei.cmu.edu/architecture/roots.html

Garlan & Shaw (1994). An Introduction to Software Architecture. [Online] Available at: http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

Philippe Kruchten, 1995, Architectural Blueprints—The “4+1” View Model of Software Architecture, [Online] Available at: http://www.win.tue.nl/~mchaudro/sa2004/Kruchten4+1.pdf

J. Han. Specifying the structural properties of software documents. Journal of Computing and Information, 1:1333--1351, 1994., [Online] Available at: http://citeseer.ist.psu.edu/cache/papers/cs/2499/http:zSzzSzwww.wi.euv-frankfurt-o.dezSzicci94zSzpaperszSze12.pdf/han95specifying.pdf

Störrle, Harald, 2004, Semantics and Verification of Data Flow in UML 2.0 Activities, [Online] Available at: http://www.pst.informatik.uni-muenchen.de/personen/stoerrle/V/AD2b-DataFlow.pdf

Medvidovic. N, et al., 2006, Understanding the past, improving the present, and mapping out the future of software architecture, [Online] Available at: http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V0N-4M21STJ-1-1&_cdi=5651&_user=122878&_orig=search&_coverDate=12%2F31%2F2006&_sk=999209987&view=c&wchp=dGLzVzz-zSkWb&md5=979d20282a5db68946b7259fd8f757b5&ie=/sdarticle.pdf

Pahl, Claus., 2006, Semantic model-driven architecting of service-based software systems [Online] Available at: http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V0B-4M93BM1-1-C&_cdi=5642&_user=122878&_orig=search&_coverDate=11%2F07%2F2006&_sk=999999999&view=c&wchp=dGLzVzz-zSkWb&md5=34186ebbdd754edc03e9badfb61888ec&ie=/sdarticle.pdf

Appendices

Appendix 1

<Wizard>	Gives a formal characterisation of the letter.
Attributes	$back_button $next_button $cancel_button $cancel_url $name $_page_url $_head_content $_form_content $_current_step $_total_steps $_override_num_steps $_last_wizstep $_current_wizstep $_step_includes $_fields $_vars $_errors
Methods	add_step draw_errors draw_wizard prepare title get_fiield set_field set_var get_var get_step set_wizard_url show_steps head
Parents	
Children	

<SimpleIterator>	Gives a formal characterisation of the letter.
Attributes	$array $count $_key $_value
Methods	current next reset size

	is_valid
Parents	
Children	

<simple_file_iterator>	Gives a formal characterisation of the letter.
Attributes	
Methods	
Parents	
Children	

<simple_file_iterator>	Gives a formal characterisation of the letter.
Attributes	$_DAO $_group_handler $_assessment $_collection $_collection_id
Methods	$_DAO $_group_handler $_assessment $_collection $_collection_id
Parents	
Children	

<SimpleObjectIterator>	Gives a formal characterisation of the letter.
Attributes	$array $class_name $class_constructor_args $count $_key $_value
Methods	current next reset size is_valid
Parents	
Children	

<NewAlgorithm>	Gives a formal characterisation of the letter.
Attributes	
Methods	calculate
Parents	
Children	

<User>	Gives a formal characterisation of the letter.
Attributes	$username $password $id $forename $surname $email $staff_id $student_id $type
Methods	load_from_row is_staff get_id_number
Parents	
Children	

<email>	Gives a formal characterisation of the letter.
Attributes	$_to $_cc $_bcc $_from $_subject $_body $_message_type $_headers
Methods	new send init set_to set_cc set_bcc set_from set_message_type set_subject set_body
Parents	
Children	

<FileUpload>	Gives a formal characterisation of the letter.
Attributes	$_is_error $_errors $overwrite $upload_path $chmod

	$files_uploaded $valid_extensions $valid_mime_types $max_file_size
Methods	FileUpload close upload is_error get_errors
Parents	
Children	

<EngCIS>	**Gives a formal characterisation of the letter.**
Attributes	$_DAO $_ordering_types
Methods	get_department_courses get_course get_course_students get_staff get_staff_modules staff_has_modules get_student get_students_modules get_user order_by_clause get_module get_module_staff get_module_students get_module_students_count get_module_students_id get_module_students_user_id get_module_grouped_students_count
Parents	
Children	

<WebPAAlogrith>	**Gives a formal characterisation of the letter.**
Attributes	$_groups $_group_members $_questions $_responses $_marking_params $_group_member_responses $_group_member_total_awarded $_group_member_frac_scores_awarded $_group_member_total_received $_group_member_webpa_scores $_group_member_intermediate_grade

	$_group_member_grade $_group_member_submitted $_member_submitted
Methods	calculate init get_webpa_scores get_intermediate_grades get_grades get_members_submitting get_member_response set_groups set_group_members set_marking_params set_questions set_responses
Parents	
Children	

<Cookie>	**Gives a formal characterisation of the letter.**
Attributes	$vars $created $last_access $_name $_expires
Methods	delete save validate
Parents	
Children	

<Role>	**Gives a formal characterisation of the letter.**
Attributes	$id $name $desc $flags
Methods	load save delete available_flags has_flag
Parents	
Children	

<Full_Iterator>	**Gives a formal characterisation of the letter.**
Attributes	$array $count

Methods	$_key $_value Full_Itterator &current end next position prev reset size is_valid _initalize
Parents	
Children	

<XMLParser>	**Gives a formal characterisation of the letter.**
Attributes	$xml_data $xml_array $_parser $_parent $_stack $_cdata_tags
Methods	tag_open tag_close count_numeric_items destroy set_cData_tags clear parse generate_xml
Parents	
Children	

<GroupIterator>	**Gives a formal characterisation of the letter.**
Attributes	$_DAO $_groupset
Methods	current
Parents	
Children	

<UI>	**Gives a formal characterisation of the letter.**
Attributes	$page_title $menu_selected $breadcrumbs $_config $_user

Methods	$_menu $_page_bar_buttons headers_expire head body header set_menu menu set_page_bar_button page_bar footer content_start content_end draw_boxed_list
Parents	
Children	

<Site_UI>	Gives a formal characterisation of the letter.
Attributes	
Methods	headers_expire head
Parents	
Children	

<LEARNAuthenticator>	Gives a formal characterisation of the letter.
Attributes	$username $password $fullname $email $staff_id $student_id $user_type $_authenticated $_outcome
Methods	authenticate is_authenticated is_staff get_error
Parents	
Children	

<ComplexUser>	Gives a formal characterisation of the letter.
Attributes	$DAO $username $password $use_local_login

	$_authenticated $_roles $_permissions
Methods	User load load_using_username load_from_row authenticate is_authenticated has_permission has_role save initalise fetch_permissions fetch_roles
Parents	
Children	

<FormRenderer>	**Gives a formal characterisation of the letter.**
Attributes	$participant_name = ''; $participant_id = null; $_form = null; $_questions = null; $_participants = null; $_results = null;
Methods	set_form set_participants set_results draw_form
Parents	
Children	

<Form>	**Gives a formal characterisation of the letter.**
Attributes	$id $name $owner_id $_DAO $_questions $_xml_parser
Methods	create delete load load_from_row load_from_xml save get_clone add_question

	get_question get_question_count set_question remove_question get_xml load_xml
Parents	
Children	

<DataAwareObject>	Gives a formal characterisation of the letter.
Attributes	$DAO $id
Methods	delete load load_from_row save
Parents	
Children	

<DAO>	Gives a formal characterisation of the letter.
Attributes	$_host $_user $_password $_database $_persistent $_conn $_result_set $_result_cols $_last_sql $_result $_output_type $results[row]['field'] $_output_type_int $_insert_id $_num_cols $_num_rows $_num_affected $_debug $_last_error
Methods	DAO open close flush execute fetch fetch_row fetch_col

	fetch_value fetch_assoc do_insert do_insert_multi do_update build_filter build_set get_cols get_num_cols get_num_rows get_num_affected get_insert_id get_last_sql get_last_error get_output_mode set_debug set_ouput escape_str process_query throw_error prepare_field_value
Parents	
Children	

<Assessment>	**Gives a formal characterisation of the letter.**
Attributes	$id $name $owner_id $open_date $close_date $introduction $allow_feedback $_DAO $_xml_parser $_collection $_collection_id $_form $_form_xml $_finished $_locked
Methods	create delete load load_from_row save finish get_collection_id set_collection_id

	get_status get_date_string is_locked get_clone get_form_xml set_form_xml
Parents	
Children	

<GroupCollection>	**Gives a formal characterisation of the letter.**
Attributes	$id $name $_DAO $_groups $_group_objects $_modules $_created_on $_locked_on $_owner_id $_owner_app $_owner_type
Methods	GroupCollection create load load_from_row delete save save_groups get_owner_app get_owner_id set_owner_info is_locked is_owner lock get_group_array group_existis group_id_existis refresh_groups add_group_object & get_group_object & new_group & get_group_iterator get_member_count get_member_count_by_group get_menbers get_member_rows & get_member_groups get_member_roles

	purge_members remove_member get_modules refresh_modules set_modules
Parents	
Children	

<Group>	Gives a formal characterisation of the letter.
Attributes	$id $name $collection_id $_DAO $_collection $_members
Methods	get_as_array set_doa_object set_collection_object add_member get_members create delete load load_from_row save get_member_ids get_members_count purge_members refresh_members remove_member
Parents	
Children	

<GroupHandler>	Gives a formal characterisation of the letter.
Attributes	$_DAO
Methods	generate_group_names & clone_collection & create_collection get_collection get_user_collections get_member_collections
Parents	
Children	