Sequencing Content in Intelligent Tutoring Systems

A Major Qualifying Project Report
Submitted to the faculty
of the
Worcester Polytechnic Institute
in partial fulfillment of the requirements of the
Degree of the Bachelor of Science
By

Derek B. Radtke
Date: March 31, 2007

Professor Neil Heffernan, Advisor

Abstract

An improved problem scheduling system for the Assistments Intelligent Tutoring System is described and examples of its applications given.

Acknowledgments

Jimmy Schementi for implementing the design for the sequence builder.

Darren Torpey for his introduction to the system and leading me around in my first days there.

Michael Purcaro for his managing of the lab during the important parts of this project.

My advisor Neil Heffernan.

Table of Contents

Introduction

The Assistments system is a web based intelligent tutoring system with a special purpose. Not only is it designed to assist students in learning but it also assesses them while they learn, allowing their time to be used more efficiently with teachers not being forced to choose how to allocate their time to between teaching and testing.

Its primary target is the MCAS (Massachusetts Comprehensive Assessment System) tests for middle and high school students. To this end it is broken up into two parts on the tutoring side. The first part performs tutoring on the individual “Assistments”, which generally start with problems taken from the MCAS tests themselves augmented with some teaching components. Here the use of problems directly from the MCAS tests helps in two ways, it directly prepares students for their MCAS test and it allows more accurate and direct assessment. The first teaching component is “buggy messages” these messages hint at the problem with an incorrect answer or suggest a correction to the student. A sequence of hints helps to lead students to figuring out the problem on their own and a sequence of sub problems break down the work into a sequence of simpler steps for the student. What type of tutoring to provide has been studied extensively in Razzaq, L., Heffernan 2006 and 2007.

Assistments are grouped for assignment to students by teachers in “sequences”. Formerly there were three operations one could use arrange to compose Assistments in sequences: Linearly (one after another in a deterministic order), Randomly (one after another in a randomized order) or by Condition (one set or the other). This was enough to allow some basic experiments to be performed such as the Hints vs. Scaffolding study of Razzaq, L., Heffernan 2006, but limited the types of experiments that could be formed readily and only provided the most basic assessment and assistance. Other systems such as Mastering Physics and Study Island have even simpler linear or random only assignments without the nested “sections” (for example, a pair of linear groupings inside of a random ordering) present in Assistments. Others, such as MAPS use Computer Adaptive Testing directly and exclusively.

As such problem scheduling is much less developed and much less studied in Assistments and similar systems. The goal of these additions to the assistments system was to increase the types of experiments that can be described and run along with increasing the quality of tutoring and assessing that can be performed. A new arrangement of sections allows resuming where students left off, more section types provide a more general descriptive language and the addition of Computer Adaptive Testing components allows more personalized assessment and assistance.

Design

In the past, a simple hierarchal tree of 4 functional groups was used, these where the internal node types Linear, Random and ChooseCondition (Then called Experimental) along with the leaf node type Problem Section. The internal nodes directed the arrangement of the problem sections when queried for the next problem.

The redesign kept a similar structure, loosening several constraints to allow more complex designs and improving the methods of communication to improve the descriptive power.

The first major change was converting to a Directed Acyclic Graph (called a DAG for short) with a single root node. This rooted directed acyclic graph's primary function is that is provides a simple way to describe the resumption of progress at a section without the need for a concept like pointers or references in the language while, along with other elements of the design, still guaranty termination at each step if none of the sections have a programming bug that cause them to not terminate. This design consideration is most important given the range of people who may be designing sequences. While some will be designed by WPI researchers, others are intended to be designed by middle school and high school teachers with no background in computer programming.

Useful operations like the logical operations ‘and’ and ‘or’ follow naturally by arranging operations (“sections”) in series or parallel and should be intuitively understood without training in math, logic or computer programming. Though some complexity may be found in the multiple parents which while well formed and rigorously defined may lead to some confusion on operation. It is believed that its other qualities, discussed above, outweigh a simpler but more limited design or the switch to a more traditional language. Proper graphical treatment may alleviate this and some work has been put forth in this effort. Several programming languages/environments such as LabVIEW have successfully used visual programming means to present such networks such as to limit the needed training. Over the web, such an environment is harder to achieve due to implement difficulties but a modified approach was designed by me and later implemented by another party at the lab (Jimmy Schementi).

The second is instead of returning a single problem; the communication between sections is now done by returning “Assistment Sequences”, which are mixed list of problems and assistment sequences. This allows problems to be associated with each other and allows the system to be presented in a uniform manner even when problem groupings are being worked with. In addition to and ID each entry in an assistment sequence can hold arbitrary Meta data on that ID or grouping of IDs. In addition to problem specific Meta data retrieved from the database, it can store runtime options such as “No Scaffolding”. Hopefully this can greatly reduce the amount of wrote entry by future experimenters and teachers.

In addition more descriptive elements where added so that not only could parameters be described for each section instance, but parameters could also describe the relationship between sections. This allows many additions, such as weighting the random sections or conditions and handling more than two conditions well.

A large amount of Meta data was added to allow introspection by utilities like the builder to ease maintenance. Node and edge variables are described in each section type with their internal name, a descriptive, human readable name, a description to inform people how to use it, a programmatic description of how to present the variable and a place for a verification function to check that enter values are valid and consistent. In addition the position that a section can take is also programmatically documented. While not part of the functional redesign, this infrastructure allows the addition of new section types without the need for changes in other parts of the system and allows the immediate use of new sections in the builder and possibly other tools without the need for changing that code. Along with this change came the naming of sections so as to remove some of the memorization magic. Now sequence builders can name sections, for example one might name the linear section at the top of an experiment “My Scaffolding vs. Hints Experiment, 2007”, the first child “Pretest”, the last child “Post test” with other logical names in between.

Finally, a large group of addition section types where added allowing a greater range of concepts to be encoded. These additional sections include the RandomInterator section, the NoRepeates section, the GroupAssistments section, the temporal and numeric limit sections along with the three parameter computer adaptive testing section and its partner the computer adaptive testing termination section. As computer adaptive testing is of a significantly different nature, it is treated separately in later sections.

Section Types

The most basic section types are the linear section, the Random section and the Problem section. These are use to form the primary functionality and are used somewhere in almost all sequences.

ProblemSection

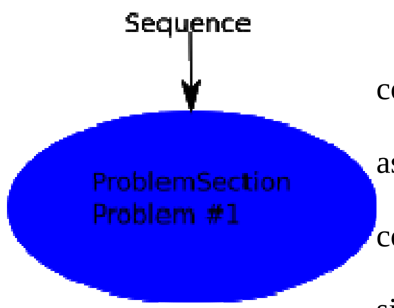

Illustration 1: This trivial section assigns assistment 1 to a student and is completed after that assistment is answered.

Problem sections are the main leaf node type in the system. They contain a reference to a single Assistment and when queried for an assistment sequence the first time, return an assistment sequence containing just that one problem and after that signal finished if queried. A single problem section forms the simplest useful sequence.

As the system matures, this section type can add Meta data that they system stores on each problem such as knowledge components and IRT parameters for other sections to use to the assistment sequence it returns. Logically, the Meta data it would add would be pulled directly from the database, though sequence designers could be given a section variable for Meta data to attach if the need arises, in the expected minor cases, the AttachMetadata section, though intended to attach Meta data to a stream of sequences coming up from a larger section, will work presently. Having programmatic assistment information could greatly simplify the sequence designer's work by saving them data entry in many cases, especially with computer adaptive testing sections.

LinearSection

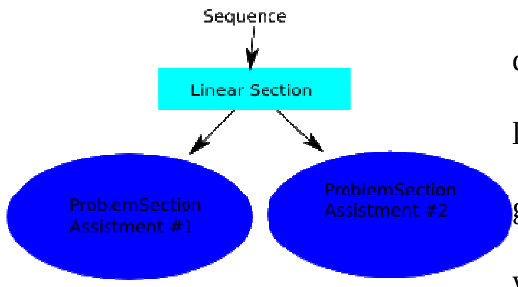

Illustration 2: This sequence contains two problems with a linear section ordering them. Assistment 1 will be pressented first followed by assistment 2 after which the sequence will be finished.

Linear sections allow a progression of problems to be described. It is an interior node which n children. Building logically from the problem section, a linear section could group a set of problem sections in order. Such a sequence would present the first problem followed by the second problem, proceeding until it had presented the last problem and then would signal finished. Linear sections need not contain only problem sections though; its children can be any other section type.

In this way, it can for an “or” clause in addition to holding predetermined orderings of problems and as such forms the backbone of most sequences created and is likely to continue to be the most common section after ProblemSections and the most common at the user level (Users are often, and need not be, aware of ProblemSections as they can consider these references to be the assistments themselves). As people will be most familiar with these sections, unspecified behavior in sections or secondary handling of data unassociated directly with a sections designed purpose defaults to acting as linear section even if they could be modeled, for example, as only taking one child. This simplifies handling of special cases in code using the sequences and sections such as the builder by eliminating a large class of them and provides less “surprise” to users when using an unfamiliar section type.

RandomOrderSection

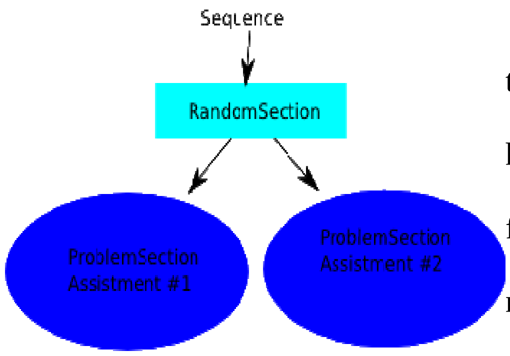

Illustration 3: This sequence is similar to the sequence in figure 2, only the linear section is replaced by a random order section. Whether assistment 1 or assistment 2 is assigned first is random and equally probable.

Random Order sections are like linear sections in that they have a set of n child sections of any type. They do not however preserve the ordering and will assign all sequences from a child before moving on but will select which child to assign assistment sequences from randomly after each child from the set of children that have not yet signaled finished. When there are no children left to assign assistment sequences from, the random section will signal finished. By default, all children are weighted equally, but optionally, children can be given specific weights. This would mean that if a group of 3 children where weighted with weights 4, 2 and 2 the one weighted 4 would come first half of the time with the ones weighted 2 each coming first one fourth of the time. In the case the weight 4 comes first, each of the weight 2s would come second half of the time and in the case one of the weight 2s came first, the weight 4 would come second two thirds of the time with the weight 2 coming second one third of the time.

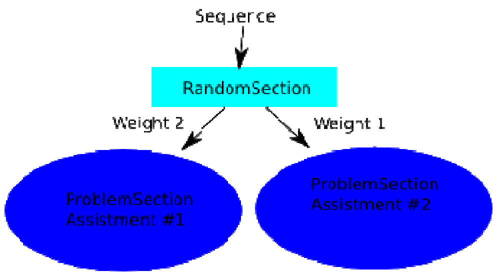

Illustration 4: As in figure 3, wit the addition of weightings. If this where assigned to 3 students it would be expected that 2 students would first be presented with assistment 1 then assistment 2 and ending with the other student being presented assistment 2 first and proceeding to 1 before completing the sequence.

ChooseConditionSection

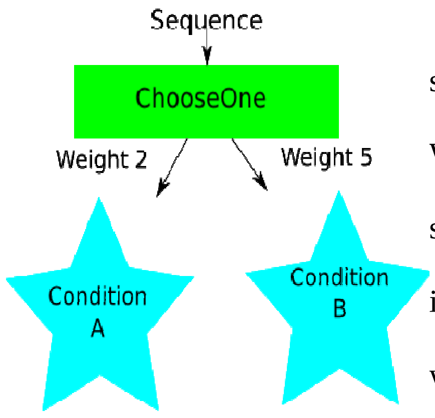

Illustration 5: A simple experimental setup with 2/7th in condition A and 5/7th in condition B.

Choose Condition sections pass on the sequences returned by a single child unlike the above which return them from all children. Which child is selected randomly with the same edge weighting system as Random Order sections uses. This has the minor issue that it is unlikely to split the students in a class exactly into groups of the weighting's relative sizes, in fact, with the small groups that this system is usually used with a splitting close to these ratios is statistically unlikely.

Due to this inequality, an additional section, perhaps called an EqualConditions section, has been conceptualized. It would look at how the other students have already been assigned by examining the working memory of the other student's progress files, and seeing the constraints needed to keep the ratios correct, would assign it's self to the correct condition. Such a section is currently in planning for being added to the system. This is a simple addition which will be made quickly after some possible different cases about reassigning a sequence are handled. The performance impact is limited to pulling all the relevant progress files from the database once upon entry into that section (proper balance can only be assured by waiting until a decision must be made about which condition to enter). While potentially intensive in larger classes, the memory use should be limited by only holding a small number in memory at a time and overall the impact should be acceptable even at larger scales.

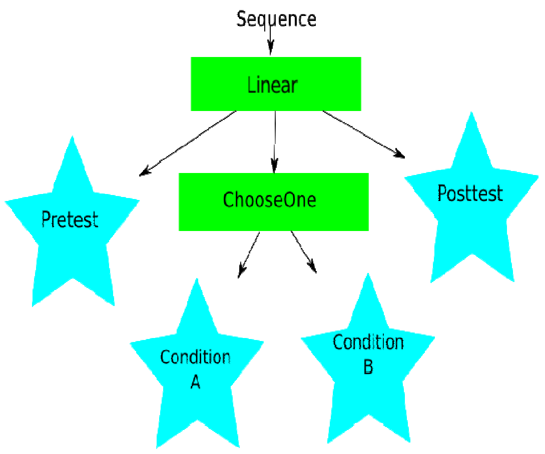

Illustration 6: A Pretest-Experiment-Posttest design as often implimented in the Assistments system.

Limit Sections

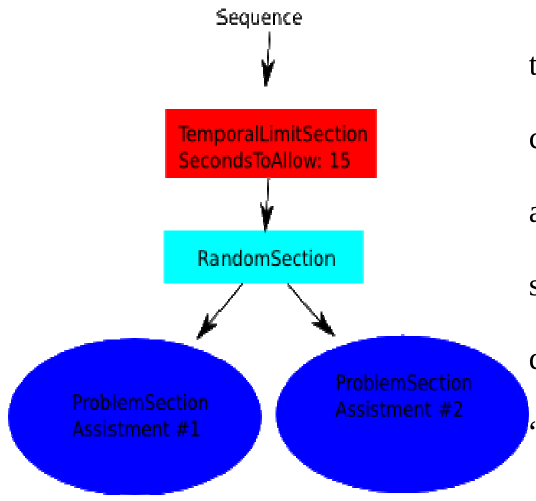

Illustration 7: This sequence describes a randomly ordered timed test of two problems for which 15 seconds total is alloted.

There are currently two types of basic limit sections, the temporal and the numeric limits. Temporal limits start counting time at the first instance that they are asked for an assistment sequence. While still active, the TemporalLimit section acts the same as a Linear section but after ether the defined number of seconds (the section parameter is called “SecondsToAllow”) or when none of its children have assistment sequences left to offer, the temporal limit signals finished. This can allow timed tests, or using the DAG properties of the language, can allow the interjection of other activities into the problem flow.

NumericLimit sections are like TemporalLimits but have two parameters: NumToDisplay, the count of how many assistments or assistment sequences to let through before terminating; and CountAssistments, which determines whether to count the assistments inside each sequences or just the number of sequences total (By default it counts the number of assistment sequences).

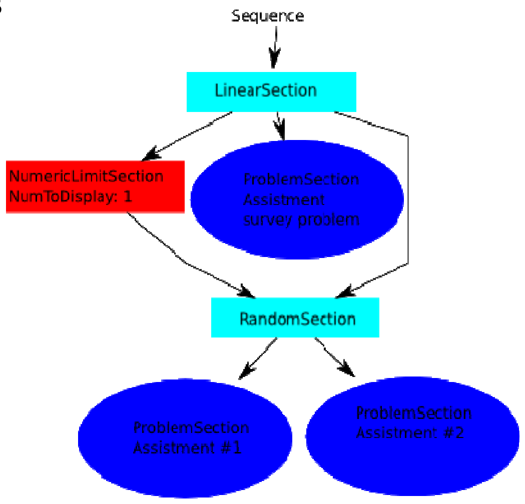

Illustration 8: This demonstrates how a limit sections can be combined with the DAG properties of the language to interject content. The Linear section first gets an assistment sequence from the NumericLimit section which gets ether assistment 1 or assistment 2 from the Random section, since the NumericLimit section is set to only return a single problem, the linear section get the next problem from the problem section returnign a survey question and then proceds to finish the sequence with whichever assistment has not yet been used in the Random section.

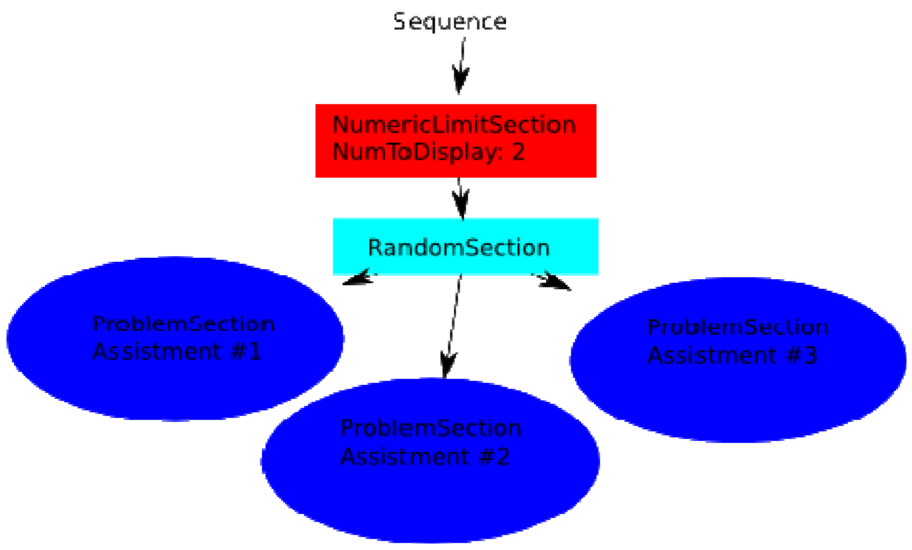

Illustration 9: This sequence demonstrates making a test containing several but not all of a set of problems. A student attempting this sequence would be presented two of the three problems at random and the extra problem would never be shown.

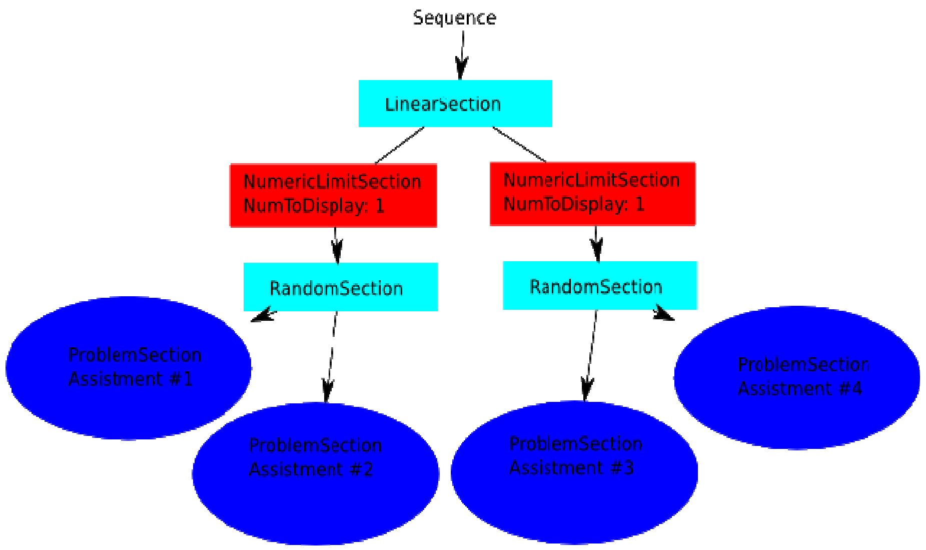

Illustration 10: This demonstrates how components can be built up. Such a structure as this might be used for when one wishes to give a test on multiple topics, including several random questions from each.

RandomIteratorSection

The RandomIterator section acts like a RandomOrderSection with one major difference. Instead of choosing a child and excusing that child's supply of assistment sequences before continuing on, it randomly selects a single child each time it is queried and selects a single assistment sequence from it. Using edge parameters, certain children can be marked as critical with the “TermOnEmpty” parameter. In this case, if the RandomIterator looks at that child for an assistment sequence and finds it empty, the RandomIterator declares its self empty also. This makes sense in the cases such that there is some “primary” functionality of the section with extras thrown in. Take for example the 90/10 sequences currently in use for longitudinal assessment. These are composed of a RandomIterator containing two children, one being the primary goal, say algebra and the other containing all problems that might be useful for that student to study. The algebra problems are assigned 90% of the time and the others assigned 10% of the time using the same edge weightings as the RandomOrder and ChooseCondition, clearly the algebra problems will run out first. It does not make sense for the student to continue through all the extra problems though but instead it is more efficient for the student to move on to the next targeted sequence, hence the algebra child would be marked TermOnEmpty. If via a chance of extremely low probability they completed the 10% first, they would just be assigned only algebra problems until they completed all the problems in that branch. In addition, if all children are empty the section also terminates.

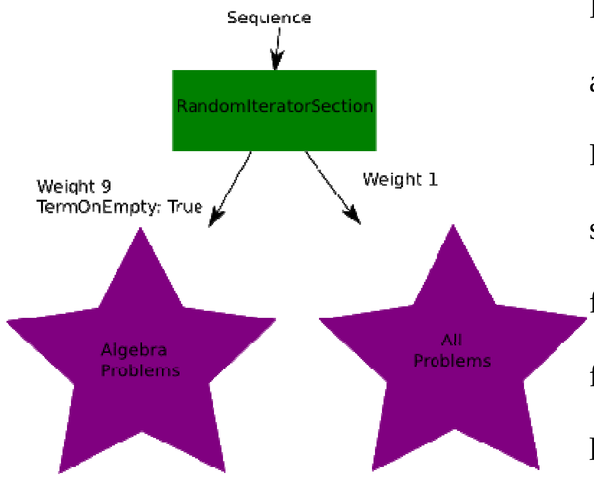

Illustration 11: The outline of a Longitudinal Assessment sequence as described in the surrounding text. This can be used similarly to intersperse survey questions into a problem set. In such a case, the lower weighted child(ren) would most likely be significantly smaller then the heavier weighted “primary” section and the more complete termination rules would come into

NoRepeatesSection

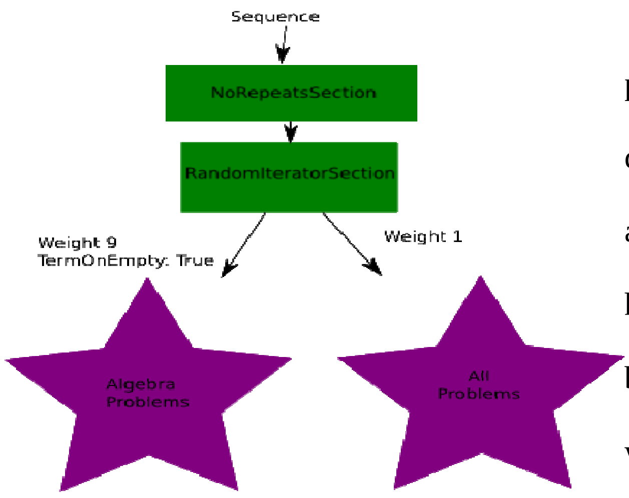

Illustration 12: This sequence trivially patches the duplicate problems the sequence design of figure 9 promotes.

In the Latitudinal Assessment sequence a problem came up, since the primary assistments where duplicated in the secondary branch, the same assistment could be assigned twice. This actually happened quite often due to a combination of the birthday paradox and the pigeon hole principal. Even without this concern though, sequences can contain problems from other sequences and sometime you wish only original problems to be assigned, thus the inclusion of the NoRepeats section. This section can operate in two modes, the first is it takes a look at the assistment sequences passing through it and only allows each to pass once, with the option "AllowRedo" to present problems that where gotten wrong again. The effects in this mode are localized to the paths of the sequences that pass through the specific NoRepeats instance. The other mode looks at all the problems the student has ever done and only allows truly new problems though (again though with the AllowRedo option). A third mode looking at the problems done globally in a sequence may be found to be of use and will be trivially implemented at that time by an extension of the student model.

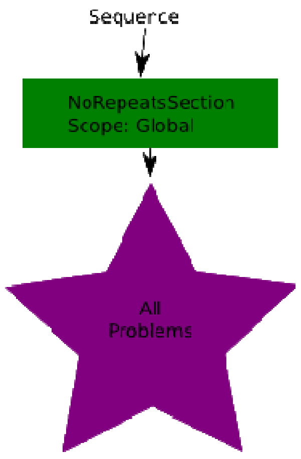

Illustration 13: This may be the design of a sequence meant to catch student up on any tutoring they missed during the year as cleanup before the MCAS tests.

Computer Adaptive Testing

Useful for quickly judging a student's ability is computer adaptive testing. Computer adaptive tests differ greatly from classical tests. While a full discussion is out of the scope of this paper a brief introduction seems necessary for the casual reader's understanding; for more detailed information and to develop a proper understanding, please see the literature.

Of the standard models there are three types the one, two and three parameter models. The one parameter model takes into account only the difficult of the problem. That is, it keeps track of the inflection point of the difficulty curve. The two parameter model accounts for both the difficulty and the discrimination of the problem. Not only does it track the inflection point, it also tracks the slope at the inflection point. Items with a steeper slope provide a greater different in the probability of the correctness of response for abilities slightly to either side of the inflection point and are hence better at determining which side a test taker lies.

The most general, and mathematicly complicated, of the standard models is the three parameter model which includes guessing on top of the two parameter models model. This is done with a third parameter that encodes the lower asymptote of the log likelihood function. Estimating Theta (the test take’s ability) becomes complicated. Modifications of the standard iterative maximum likelyhood estimation exist using Newton’s method exist but others are usually found to be more usefull for a specific application.

Computer adaptive tests, which base their estimates on a statistical model of response correctness across ability levels, select problems at each iteration such as to maximize the information gained from an answer to that problem. In the standard one and two parameter models that means that problems are picked which students have approximately an even chance of answering correctly or incorrectly. The three parameter model used selects problems with a bit over 50% generally since it includes a model of guessing which pushes up the point of maximum information. At each step, the test taker's ability (called theta) is estimated (this estimate is theta hat) so that the most correct problem can be assigned given the current knowledge of the test taker's ability. In practice the best is not always selects, for example, one of the best 5 can be selected to cause divergence. For one this prevents cheating in that two students answering together will not get the same test.

The other half of computer adaptive tests is deciding when to terminate. Some common termination points are a number of problems, a score confidence or a time limit. Any of these can be easily implemented with the already built sections. In previous sections both the numeric and temporal limit sections where described.

Current Implementation of the IRT 3 Parameter Sections

As it currently stands, the computer adaptive test section it's self currently is implemented as a section with 3 link parameters a, b, and c (a being discrimination, b being difficulty and c being guessing; as the literature uses). This is not the optimal solution as it encourages only having problem sections below a CAT section and requires manual entry

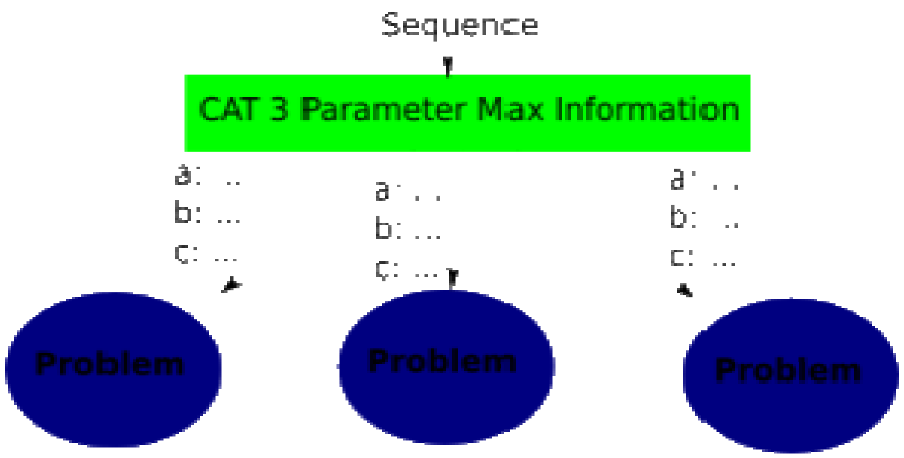

Illustration 14: This CAT section is of "Max Information" type meaning it tries to select be best item as far as estiamting theta goes as the next problem. The parameters for each problem are not defined here and just their possition denoted. One will note that this sequence contains no type of limit section and this sequence will continute perfoming a CAT test untill the CAT section runs out of children to assign. Since in practice all data is best collected in a single session, one might usually add a temporal limit to such a design, not to explicitely limit the student's work but to implicitely constrain it to a single period even when the goal was to make the best possible assesment.

of each parameter. In the future once the system is ready for peripheral work (The Ruby system is just being finalized now, it is a from-scratch rewrite of the generation 2 Java system) the ability to retrieve item parameters from the database directly would be convenient and encourage the much greater use of computer adaptive sections. Normal computer adaptive sections try to select the most informative problem; this exact selection technique has been factored out in the Assistments' implementation in favor of interchangeable selection methods. The most informative technique is referred to as “max information” and the CAT section using it is referred to as a CatThreePMaxInformationSection.

Other types of computer adaptive sections of interest are ones that select on different algorithms. For one, to encourage more attention, lower gaming and increase happiness with learning with the intent of increasing learning selects problems that a student has a certain percentage change of getting right, say 20%. This of course will not work well with assessment as all the problems will be “below” their ability level and hence the theta estimate from that set would have a high error. As such, it would be best to use such a design with a pre-estimated theta.

Termination was split out into a separate section for three main reasons. First it fits the style of the frame work paralleling the existing Numeric and Temporal limits. Furthermore it provides more flexibility. The Theta limit section need not occur at the computer adaptive section it's self. It can control other problems related such as assigning further problems if students test too low for example. This flexibility also allows

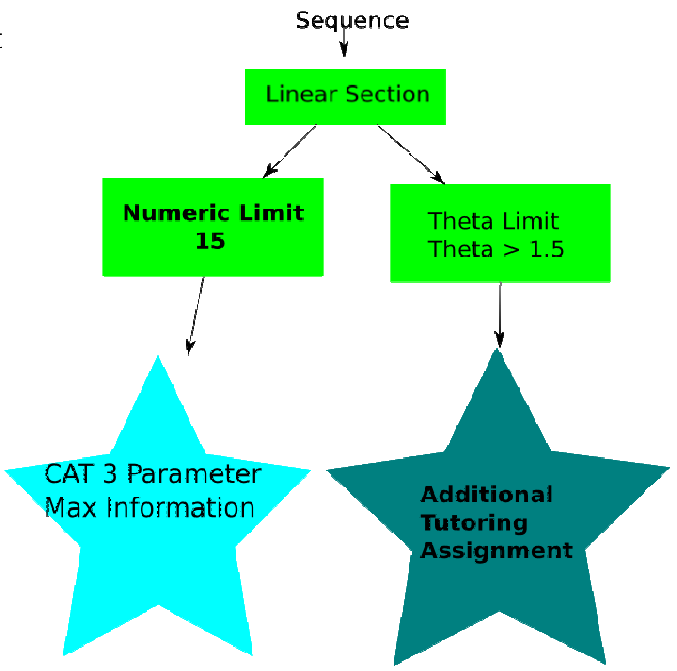
Sequence
Linear Section
Numeric Limit 15
Theta Limit Theta > 1.5
CAT 3 Parameter Max Information
Additional Tutoring Assignment

Illustration 15: This design shows a sequence that first assigns a 15 problem long computer adaptive test and then if the student does not have a high enoguh skill assigns additional tutoring in that area.

you to perform logical operations with Theta limits and Standard Error limits. This last example is the third reason, it allows the type of termination to be exchange without having to present overly numerous options to the users, they can design their sequence as they see fit without every type of combination having to have been already defined. Along with this, standardized names mean that each limit section type could work with one and two parameter computer adaptive sections also if they were released or even other types of adaptive sections.

Improvements

The current system finds the theta that best predicts the data, while this is a decent theta estimate and the standard generally used, there are other options. For use of predicting MCAS scores, a primary goal of the assistments system, this maximum likelihood estimation is a strong predictor. Others are better at handling some random answers such as formed by guessing without thinking, gamming the system or some else wise defined criteria. Examples are biweight, expected a posteriori (EAP), Maximum a posteriori (MAP or Bayes model estimate).

Future Work

The amount of difficult associated with optimal learning would form an interesting study and may be conducted given the computer adaptive sections implemented in this project.

The utility of assigning only problem sets students have a low estimated theta on may also wish to be studied. This can be accomplished with the current separated limit design and the sequence design is detailed above. The converse, assigning only re-enforcement problem sets on areas students scored reasonable will require duplication of the theta limit section and the obvious minor modification to invert the selection though the same sequence design is used. Of course, with these two sections, the middle ground of problem sets a student is estimated to not be too skilled at but not entirely unskilled at is also possible. This may be an area assistments excels at as it is not prepared to provide the initial teaching and little is to be learned on problems students already are experts at. Especially what range was the best for which type of reinforcement is ripe for study extending upon the former hints vs. scaffolding work to bringing more detail to the proper applications of each.

Less enlightening but important is the continued addition of section types to enable new experiments hither to consider. While this project extended the scope of possible experiments greatly it's lack of Turing completeness assures that some cannot be run without additions, furthermore, in the more immediate and foreseeable future, near the end of this project, many convenience sections where considered that were not strictly needed to achieve the necessary utility but may be wanted. Similarly, as the system is used more strengths and weaknesses will be found, the framework was designed to be flexible enough to encompass the necessary additions for usability but these additions may need to be performed to extend even further or allow use by less trained sequence builders.

Conclusion

This project has successfully completed its goal of writing an assistment scheduling system flexible enough to handle many new experiment types. It has allowed the writing of several new experiment types. The paired analysis that initially motivated it; ten or more 90/10 sections for a 2 Million dollar grant for Making Longitudinal Assessment from the US Department of Education. In addition to including computer adaptive sections to aid in progressive sequence designs. Furthermore; the design showed its self readily able to include the new ideas that arrived near the end of the project.

Less sure is how the user sequence builder design will fair as it is just been mostly completed by Jimmy Schementi. The light use it has gotten so far has been successful, but that is merely anecdotal and not in the problematic groups. This side of the project can only be truly tested when the Ruby version of the Assistments system goes live.

Bibliography

Razzaq, L., Heffernan, N.T. (2006). Scaffolding vs. hints in the Assistment System. In Ikeda, Ashley & Chan (Eds.). Proceedings of the 8th International Conference on Intelligent Tutoring Systems. Springer-Verlag: Berlin. pp. 635-644. 2006.

Razzaq, L., Heffernan, N.T. (2007). **What level of tutor interaction is best?**. In Luckin & Koedinger (Eds) Proceedings of the 13th Conference on Artificial Intelligence in Education. IOS Press.

Lord, Frederick M. (1980). Applications of Item Response Theory to Practical Testing Problems. Hillsdale, NJ: Erlbaum.

De Ayala, R. J., Plake, Barbara S., Impara, James C., Kozmicky, Michelle (2001). The Effect of Omitted Responses on Ability Estimation in IRT. Annual Meeting of the American Educational Research Association

Embretson, Susan E., Reise, Steven P. (2000). Item response Theory for PsychologistsLawrence Erlbaum Associates, Publishers

Baker, Frank (2001). *The Basics of Item Response Theory.* ERIC Clearinghouse on Assessment and Evaluation, University of Maryland, College Park, MD.

Appendix A – Converter to New Java Implimentation

Based on the old Java system, needs XMLNode for proper function.

Converts the old system's curriculums into the intermediate java system's and applies some compression shorthands allowed in that system's storage model.

Available at
svn+ssh://nth2.wpi.edu/home/svn/assistments/branches/OneOffScripts/CurriculumConverter/

cvert.java

```
package CurriculumConverter;


import java.sql.*;
import java.util.*;


class SecStruct {
	String type;
	List<Long> children;
//	XMLNode node;
}


public class cvert {


	private static final String oracleServer="";
	private static final String oracleDatabaseName="";


//	private static final String oracleServer="assistment5.cs.wpi.edu";
//	private static final String oracleDatabaseName="silver";


	private static final String ORACLE_DB_STRING =
"jdbc:oracle:thin:@"+oracleServer+":1521:"+oracleDatabaseName;
	static List<SecStruct> sections;
static public long inflateXML(XMLNode node) throws Exception
{
	SecStruct s = new SecStruct();
	long id = sections.size();
	s.children = new ArrayList<Long>();

	if(!node.getName().equals("isection"))
		throw new Error("Not where we aut.");

	String type = node.getNodeText("type");
	if(type.equals("ExperimentSection")) {
		s.type = "ChooseOne";
	} else if(type.equals("LinearSection") || type.equals("Linear")) {
		s.type = "Linear";
	} else if(type.equals("ProblemSection")) {
		s.type = "Problem";
	} else if(type.equals("RandomSection")) {
		s.type = "RandomOrder";
	} else {
		System.out.println(type);
		throw new Exception("Bad section type!!!");
	}

	XMLNode p = node.getNode("properties").getNode("property");
	if(null!=p && p.getNodeText("key").equals("mProblemID")) {
		return -1*new Long(p.getNodeText("value").trim());
	}

	sections.add(s);

	Iterator children = node.getNode("children").namedChildren("isection");
	while(children.hasNext())
	{
```

```
                XMLNode k = (XMLNode) children.next();
                s.children.add(inflateXML(k));
            }

            return id;
        }

        public static String deflateXML(int id) {
            XMLNode n = new XMLNode("<isection type=\""+sections.get(id).type+"\"/>");

            n.insertChildNode(new XMLNode("<children />"));

            for (Long c : sections.get(id).children) {
                if(0>c) {
                    n.getNode("children").insertChildNode(new XMLNode("<child
type=\"problem\" id=\""+(-1*c)+"\""));
                } else {
                    n.getNode("children").insertChildNode(new XMLNode("<child
type=\"section\" id=\""+c+"\""));
                }
            }

            return n.toString();
        }

    public static XMLNode clobToXML(Clob clob)
    {
      String nodeString = "";

      try {
        nodeString = clob.getSubString(1, (int)clob.length());
      } catch(SQLException sqle){
        System.out.println("XMLSaver (clobToXML): " + sqle.getMessage());
        }

        // return null if the clob is null
        if(nodeString.equals(""))
            return null;

        return new XMLNode(nodeString);
    }

            public static void convert() {
                        XMLNode fake = new XMLNode("<test />"); //Just here to get the japisoft notice at the
ttop and not in our output.

                        long startT, stopT, elapsedT, sizeT=0, numConverted=0, numFailed=0;

                        startT = System.currentTimeMillis();

                        try {

                                    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());

                                    Connection dbS = DriverManager.getConnection (ORACLE_DB_STRING,
"assistment", "assistment");

                                    //Setup our tables
                                    PreparedStatement psTBLS;
                                    try {
                                                psTBLS = dbS.prepareStatement("DROP TABLE
assistment_model.curriculums");
                                                psTBLS.execute();
                                    } catch (Exception e1) {
                                                System.out.println("Fail #" + 1);
                                    }
```

```
			psTBLS = dbS.prepareStatement("CREATE TABLE
assistment_model.curriculums (id INTEGER NOT NULL, status CHAR(4) DEFAULT 1, authorid
INTEGER NOT NULL, creationdate DATE NOT NULL, name VARCHAR2 (64) NOT NULL,
description VARCHAR2 (255), obj_xml CLOB)");
			psTBLS.execute();

			psTBLS = dbS.prepareStatement("CREATE UNIQUE INDEX
assistment_model.sys_c005657 ON assistment_model.curriculums (id)");
			psTBLS.execute();

			try {
				psTBLS = dbS.prepareStatement("DROP TABLE
assistment_model.progresses");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 2);
			}

			try {
				psTBLS = dbS.prepareStatement("CREATE TABLE
assistment_model.progresses (sid INTEGER NOT NULL, cid INTEGER NOT NULL, total_done
INTEGER NOT NULL, complete INTEGER NOT NULL, last_touched TIMESTAMP(6) NOT NULL,
static_data CLOB, dynamic_data CLOB)");
				psTBLS.execute();

				psTBLS = dbS.prepareStatement("CREATE UNIQUE INDEX
assistment_model.usercurriculum ON assistment_model.progresses (sid, cid)");
				psTBLS.execute();

				psTBLS = dbS.prepareStatement("ALTER TABLE
assistment_model.progresses ADD CONSTRAINT usercurriculum PRIMARY KEY (sid, cid)");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 3);
			}
			try {
				psTBLS = dbS.prepareStatement("DROP SEQUENCE
assistment_model.curriculum_seq");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 4);
			}

			psTBLS = dbS.prepareStatement("CREATE SEQUENCE
assistment_model.curriculum_seq START WITH 1 INCREMENT BY 1");
			psTBLS.execute();

			PreparedStatement psDC = dbS.prepareStatement("DELETE FROM
assistment_model.curriculums");
			psDC.executeQuery();
			//PreparedStatement psDP = dbS.prepareStatement("DELETE FROM
assistment_model.progresses");
			//psDP.executeQuery();

			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/jpeg' where mime_type='jpg'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 5);
			}

			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/jpeg' where mime_type='JPG'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 6);
			}
			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/jpeg' where mime_type='jpeg'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 7);
			}

			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/gif' where mime_type='gif'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 8);
			}

			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/gif' where mime_type='GIF'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 9);
			}

			try {
				psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/bmp' where mime_type='bmp'");
				psTBLS.execute();
			} catch (Exception e1) {
				System.out.println("Fail #" + 10);
			}

			try {
					psTBLS = dbS.prepareStatement("update assistment_model.IMedia set
mime_type='image/png' where mime_type='png'");
					psTBLS.execute();
				} catch (Exception e1) {
					System.out.println("Fail #" + 11);
				}


				PreparedStatement psS = dbS.prepareStatement("SELECT * FROM
assistment_model.icurriculum WHERE XML_CLOB like '<ic%' ORDER BY obj_id"); // AND
OBJ_ID=1466");


				ResultSet rsS = psS.executeQuery();


				while(rsS.next()) {
			Connection dbI = DriverManager.getConnection (ORACLE_DB_STRING,
"assistment", "assistment");


					System.out.print("Trying: "+rsS.getLong(9)+" ... ");


					try {
						PreparedStatement psI = dbI.prepareStatement("INSERT INTO
assistment_model.curriculums (id,status,authorid,creationdate,name,description,obj_xml) VALUES
(?,?,?,?,?,?,?)");


						psI.setLong(1, rsS.getLong(9));


						String ssss = rsS.getString(11);
						if(null==ssss) { ssss="1"; }
						psI.setString(2, ssss);


						psI.setLong(3, rsS.getLong(5));


						if(null==rsS.getTimestamp(6)) {
							psI.setTimestamp(4, new
Timestamp(System.currentTimeMillis()));
```

```
        } else {
                psI.setTimestamp(4, rsS.getTimestamp(6));
        }

        if(null==rsS.getString(3)) {
                psI.setString(5, "Unknown");

        } else {
                psI.setString(5, rsS.getString(3));
        }

        if(null==rsS.getString(4)) {
                psI.setString(6, "Unknown");

        } else {
                psI.setString(6, rsS.getString(4));
        }

        long start, stop, elapsed;

        start = System.currentTimeMillis();
        sections = new ArrayList<SecStruct>();

        XMLNode n = clobToXML(rsS.getClob(10));

        long numRoots=0;
        boolean needFake=false;
        Iterator children =
n.getNode("sections").namedChildren("isection");
        if(children.hasNext() && children.hasNext()) {
                //Need fake top linear section
                needFake=true;
                sections.add(new SecStruct());
                sections.get(0).type = "Linear";
```

```
                        sections.get(0).children = new ArrayList<Long>();
                    }

                    children = n.getNode("sections").namedChildren("isection");
                    while(children.hasNext())
                    {
                        System.out.print("@ ");
                        numRoots++;
                        XMLNode k = (XMLNode) children.next();
                        Long tid = inflateXML(k);
                        if(needFake) {
                            sections.get(0).children.add(tid);
                        }
                    }

            String b = "<sections>";
              for(int i=0; i<sections.size(); i++){
                   b += deflateXML(i);
              }
              b += "</sections>";



            stop = System.currentTimeMillis();
            elapsed = stop - start;

            System.out.print((new Long(elapsed)).toString()+"ms, "+b.length()+" bytes ... ");

                    //psI.setString(7, b);

              oracle.sql.CLOB newClob = oracle.sql.CLOB.createTemporary(dbI, false,
oracle.sql.CLOB.DURATION_SESSION);

            newClob.putString(1,b);
```

```
                    psI.setClob(7, newClob);

                                psI.executeQuery();

                      sizeT += b.length();

                      numConverted++;
                                System.out.println("Success! ("+new
Long(numRoots).toString()+"):-)");
                          } catch(Exception e) {
                                numFailed++;
                                System.out.println("Failure! :-(");
                                e.printStackTrace();
                          }
                    }

                    dbS.close();
              } catch(Exception e) {
                    e.printStackTrace();
                    System.out.println("Upper");
              }

    stopT = System.currentTimeMillis();
    elapsedT = stopT - startT;

    System.out.println("Total time taken: "+(new Long(elapsedT)).toString()+"ms ("+(new
Long(elapsedT/1000)).toString()+"s).");
    System.out.println("Total size writen: "+(new Long(sizeT)).toString()+" bytes.");
    System.out.println("Total number converted: "+(new Long(numConverted)).toString());
    System.out.println("Total number failed to convert: "+(new Long(numFailed)).toString());
      }
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		/*
		sections = new ArrayList<SecStruct>();

	   try {
	       BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
	       String str = "";
	System.out.print("> ");
	str = in.readLine();
	XMLNode is = (new XMLNode(str)).getNode("isection");
	inflateXML(is);
	   } catch (IOException e) {
		System.out.println("Bad read");
		return;
	   }

	   System.out.print("<sections>");
	   for(int i=0; i<sections.size(); i++){
		System.out.print(deflateXML(0));
	   }
	   System.out.print("</sections>");
	   */

		convert();
	}

}
```

Appendix B – New Java Implimentation

ICurriculum

```java
package org.assistment.core.curriculum;

import org.assistment.core.section.ISection;


public interface ICurriculum
{
	// Component
	public ISection getHead();

	// Hibernate
	public Long getId();
}
```

Curriclum

```java
package org.assistment.core.curriculum;

import org.assistment.core.section.ISection;

/**
 *
 * There is an implied linear root section, thanks to the list of sections
 * @author Derek Radtke
 * @author Tom Livak
 */

public class Curriculum implements ICurriculum {
	ISection mSection;

	Long mID;

    public Curriculum(Long id, ISection section) {
		mID = id;
		mSection = section;
	}

	public ISection getHead() {
		return mSection;
	}

	public Long getId() {
		return mID;
	}
}
```

IProgress

```
package org.assistment.core.progress;

import java.util.Date;
import java.util.Map;

import org.assistment.core.problem.IProblem;
import org.assistment.core.section.IProblemSequence;
import org.assistment.core.section.SectionDynamicData;
import org.assistment.core.section.SectionStaticData;
import org.assistment.database.harness.SQLHarness;
import org.assistment.datalayer.DataLayerException;
import org.assistment.runtime.logging.RuntimeLogger;

public interface IProgress {

	// Component
	public void saveState( SQLHarness harness ) throws DataLayerException;
	public void updatedTimeStamp(SQLHarness harness);
	public IProblem nextAssistment(RuntimeLogger logger, SQLHarness harness );
	public boolean isComplete();

	//Fuggly compatability datalayer stuff.
	public Long getUserId();

	public String getType();

	public int getTotalNumberDone();

	public boolean getComplete();
	public void setComplete(boolean complete, SQLHarness harness);

	public Date getTimeOfLastProgressSave();

	public Long getCurriculumID();

	public Map<Long, SectionStaticData> getSectionStaticProgress();
	public Map<Long, SectionDynamicData> getSectionDynamicProgress();

	public IProblemSequence getProblemSequence();

}
```

Progress

```
package org.assistment.core.progress;

import static org.assistment.apps.utils.FLog.exception;
import static org.assistment.apps.utils.FLog.log;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import org.assistment.core.curriculum.ICurriculum;
import org.assistment.core.problem.IProblem;
import org.assistment.core.section.IProblemSequence;
import org.assistment.core.section.SectionDynamicData;
import org.assistment.core.section.SectionStaticData;
import org.assistment.core.utils.ProblemUtils;
import org.assistment.database.harness.SQLHarness;
import org.assistment.datalayer.DataLayerConnection;
import org.assistment.datalayer.DataLayerException;
import org.assistment.framework.XMLNode;
import org.assistment.runtime.RuntimeSQLUtils;
import org.assistment.runtime.logging.RuntimeLogger;

public class Progress implements IProgress {
	private ICurriculum mGraph;

	private IProblemSequence mSequence;

	// Contains properties needed to properly restore sections
	private Map<Long, SectionStaticData> mSSD;

	Map<Long, SectionDynamicData> mSDD;

	// Persisted Properties
	private long mUID;

	private Long mCurriculumID;

	private int mTotalNumberDone;

	private boolean mComplete;

	private Date mTimeStamp;

	/**
	 * 
	 * 
	 * @param curriculumID
 *            the unique id of the curriculum
 */
public Progress(final long user, final long curriculumID,
              final Date timeStamp) {
       mUID = user;
       mComplete = false;

       mTimeStamp = timeStamp;

       mSequence = null;

       mCurriculumID = curriculumID;
       mTotalNumberDone = 0;
}

/**
 * This will instantiate a progress that was stored into the DB.
 */
public Progress(final long uid, final long cid, final int tnd,
              final boolean cmplt, final Date dt, final IProblemSequence ps,
              final Map<Long, SectionStaticData> ssd,
              final Map<Long, SectionDynamicData> sdd) {
       mUID = uid;
       mCurriculumID = cid;
       mComplete = cmplt;
       mTotalNumberDone = tnd;
       mTimeStamp = dt;
       mSequence = ps;
       mSSD = ssd;
       mSDD = sdd;
}

public Long getUserId() {
       return mUID;
}

/**
 * Save the current progress through the curriculum
 *
 * @param harness
 * @throws DataLayerException
 * @throws DataLayerException
 *
 * @pre mCurrentSection is not null
 */
public void saveState(SQLHarness harness) throws DataLayerException {
       if (null == mGraph) {
              try {
                     loadCurriculum(harness);
              } catch (DataLayerException e) {
                mComplete = true; // No curriculum, so we can't do anything.
                DataLayerConnection.getInstance().saveProgress(this, harness);
                return;
            }
        }

        // Aquire static data once and only once, don't reaqure if we already
        // have it.
        if (null == mSSD) {
            mSSD = new HashMap<Long, SectionStaticData>();
            mGraph.getHead().saveStaticData(mSSD);
        }

        // Get dynamic state that represent the current
        // progress through the current curriculum
        mSDD = new HashMap<Long, SectionDynamicData>();
        mGraph.getHead().saveDynamicData(mSDD);

        DataLayerConnection.getInstance().saveProgress(this, harness);
    }

    private void loadCurriculum(SQLHarness harness) throws DataLayerException {
        try {
            mGraph = DataLayerConnection.getInstance().getCurriculum(
                    mCurriculumID, harness);

            // If we are loading so we can save we will just fill it in with
            // blank.
            loadState();

        } catch (DataLayerException e) {
            exception(this.getClass().getName(), e, e.getMessage());
            throw e;
        }
    }

    private void loadState() {
        // call the load recursive function, we have an issue if we don't have
        // the data to load
        if (null != mSSD && null != mSDD) {
            mGraph.getHead().loadData(mSSD, mSDD);
        } else {
            mGraph.getHead().loadData(new HashMap<Long, SectionStaticData>(),
                    new HashMap<Long, SectionDynamicData>());
        }
    }

    public IProblem nextAssistment(RuntimeLogger logger, SQLHarness harness) {

        if (null == mGraph) {
```

```
        try {
            loadCurriculum(harness);
        } catch (DataLayerException e1) {
            mComplete = true;
            return null;
        }
    }

    try {
        saveState(harness);
    } catch (DataLayerException e) {
        exception(this.getClass().getName(), e, "Could not save  - "
                + e.getMessage());
    }

    IProblem problem = null;

    while (!mComplete) {
        // An empty sequence is fine, but then we need to go back.
        if (null == mSequence || mSequence.isEmpty()) {
            logger.logSectionQueryStart();

            mSequence = mGraph.getHead().getNextSequence(logger,
                    new ArrayList<Long>());

            // This just does logging so we know what happened. (for
            // Debugging)
            logSequenceGotten(logger);
        }

        // We are sure it is ether null or !empty given the above line
        if (null == mSequence) {
            mComplete = true;
            try {
                saveState(harness);
            } catch (DataLayerException e) {
                exception(this.getClass().getName(), e, "Could not save - "
                        + e.getMessage());
            }
            return null;
        }

        Long pid = mSequence.getFirstProblem();
        if (null != pid) {
            problem = ProblemUtils.getProblem(pid, harness);
        }

        if (problem != null) {
            mTotalNumberDone++;
            return problem;
				} else { // We couldn't load the problem, it must be bad problem?
					log(this.getClass().getName(), "We failed to load problem id #"
							+ pid.toString());
				}
			}

			return null;
		}

		private void logSequenceGotten(RuntimeLogger logger) {
			if (null != mSequence) {
				String Seq = "";
				for (XMLNode nd : mSequence.getSubs()) {
					if (Seq.length() > 0) {
						Seq += " ";
					}
					Seq += nd.getAttribute("problem");
				}
				logger.logSectionQueryResult(Seq);
			} else {
				logger.logSectionQueryResult("Completed");
			}
		}

		public boolean isComplete() {
			return mComplete;
		}

		public String getType() {
			return "Progress";
		}

		public Long getCurriculumID() {
			return mCurriculumID;
		}

		public int getTotalNumberDone() {
			return mTotalNumberDone;
		}

		public boolean getComplete() {
			return mComplete;
		}

		public void setComplete(final boolean complete, SQLHarness harness) {
			mComplete = complete;
			try {
				saveState(harness);
			} catch (DataLayerException e) {
				exception(this.getClass().getName(), e, "Could not save - "
                                    + e.getMessage());
                }
        }

        public Date getTimeOfLastProgressSave() {
                return mTimeStamp;
        }

        public Map<Long, SectionStaticData> getSectionStaticProgress() {
                return mSSD;
        }

        public Map<Long, SectionDynamicData> getSectionDynamicProgress() {
                return mSDD;
        }

        public IProblemSequence getProblemSequence() {
                return mSequence;
        }

        public void updatedTimeStamp(SQLHarness harness) {
                RuntimeSQLUtils
                                .updatedTimeStamp(this.mUID, this.mCurriculumID, harness);
        }
}
```

Isection

```
package org.assistment.core.section;

import java.util.List;
import java.util.Map;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.runtime.logging.RuntimeLogger;

/**
 * To add a new section, Create said section. Update SectionUtils.
 *
 * You are now done.
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public interface ISection {
        /**
         * Save's a section's unchanging static data into a hash map at its section
         * ID, recursively saving all of its children's data also.
         *
         * This is called before any requests are made of the section.
         *
         * @param SDM
         *            The Static Data Map, mapping a section's ID number to the
         *            static data a section cares to save.
         */
        public void saveStaticData(Map<Long, SectionStaticData> SDM);

        /**
         * Save's a section's changing dynamic data into a hash map at its section
         * ID, recursively saving all of its children's data also.
         *
         * @param DDM
         *            The Dynamic Data Map, mapping a section's ID number to the
         *            dynamic data a section cares to save.
         */
        public void saveDynamicData(Map<Long, SectionDynamicData> DDM);

        /**
         * Recursively called into the section graph to populate sections with their
         * saved data.
         *
         * @param SDM
         * @param DDM
         */
        public void loadData(Map<Long, SectionStaticData> SDM,
                    Map<Long, SectionDynamicData> DDM);

        /**
         * Asks a section for the next sequence of problems is has to return. It may
         * ask its children for sequences and use them to create it's sequence, and
         * so on recursively.
         *
         * @param logger
         * @param path
         *            Path from root to current section; this is only used for
         *            logging.
         * @return The IProblemSequence to use or null if that section is out of
         *         IProblemSequences.
         */
        public IProblemSequence getNextSequence(RuntimeLogger logger,
                    List<Long> path);

        /**
         * TODO: Refactor to "getTypeName"
         *
         * @return A string representing the type of the section.
         */
```

```
        public SectionType getType();
}
```

IProblemSequence

```
package org.assistment.core.section;

import java.util.List;

import org.assistment.framework.XMLNode;

public interface IProblemSequence {
        public List<IProblemSequence> getSubSequences();

        public boolean isEmpty();

        public Long getFirstProblem();

        public void appendSequence(IProblemSequence sq);

        public void insertSequence(IProblemSequence sq);

        public XMLNode getXMLrep();

        List<XMLNode> getSubs();
}
```

ProblemSequence

```
package org.assistment.core.section;

import java.util.ArrayList;
import java.util.List;

import org.assistment.framework.XMLNode;

/**
 * I'm sorry, I know I screwed this one up. Please fix it?
 *
 * @author Derek Radtke (davean@wpi.edu)
 *
 */
public class ProblemSequence implements IProblemSequence {
        // Ether we store data or we store children.
        Long mIDNum;

        List<IProblemSequence> mSequences;

        public ProblemSequence() {
                mIDNum = null;
```

```
		mSequences = null;
	}

	public ProblemSequence(long id) {
		mIDNum = id;
		mSequences = null;
	}

	// Acts as merge.
	public ProblemSequence(IProblemSequence sqA, IProblemSequence sqB) {
		mIDNum = null;
		mSequences = new ArrayList<IProblemSequence>();
		mSequences.addAll(sqA.getSubSequences());
		mSequences.addAll(sqB.getSubSequences());
	}

	public ProblemSequence(@SuppressWarnings("unused")
	String attribute) {
		// TODO Auto-generated constructor stub
	}

	public String toString() {
		// TODO actually serialize
		return "";
	}

	public List<IProblemSequence> getSubSequences() {
		if (null == mIDNum && null != mSequences) // We aren't a leaf and we
													// have
data
		{
			return mSequences;
		} else if (null != mIDNum) {
			List<IProblemSequence> sq = new ArrayList<IProblemSequence>();
			sq.add(this);
			return sq;
		} else {
			List<IProblemSequence> sq = new ArrayList<IProblemSequence>();
			return sq;
		}
	}

	public boolean isEmpty() {
		return (null == mIDNum && (null == mSequences || mSequences.size() == 0));
	}

	public Long getFirstProblem() {
		if (null != mIDNum) // leaf
		{
			Long id = mIDNum;
```

```
            mIDNum = null;
            return id;
        } else if (null != mSequences) {
            Long id = null;

            while (null == id && !mSequences.isEmpty()) {
                id = mSequences.get(0).getFirstProblem();
                if (mSequences.get(0).isEmpty()) {
                    mSequences.remove(0);
                }
            }

            if (mSequences.size() <= 0) // Clean up if we are out.
            {
                mSequences = null;
            }

            return id;
        } else {
            return null;
        }
    }

    public void appendSequence(IProblemSequence sq) {
        if (null == mSequences) {
            mSequences = new ArrayList<IProblemSequence>();
        }

        if (null != mIDNum) { // we were leaf, fix that
            mSequences.add(new ProblemSequence(mIDNum));
            mIDNum = null;
        }

        // We are now an internal node no matter how we started.
        mSequences.add(sq);
    }

    public void insertSequence(IProblemSequence sq) {
        if (null == mSequences) // We already have children
        {
            mSequences = new ArrayList<IProblemSequence>();
        } else if (null != mIDNum) // we are a leaf
        {
            mSequences = new ArrayList<IProblemSequence>();
            mSequences.add(new ProblemSequence(mIDNum));
            mIDNum = null;
        }
        // else we already have subSequences.
        mSequences.add(sq);
    }
```

```
	// Good enough for progress' purposes ...
	public List<XMLNode> getSubs() {
		List<XMLNode> nodes = new ArrayList<XMLNode>();

		if (null != mIDNum) // leaf
		{
			Long id = mIDNum;
			XMLNode probNode = new XMLNode("<subSeq />");
			probNode.setAttribute("problem", id.toString());
			nodes.add(probNode);
		} else if (null != mSequences) { // No, this is not an else Darren,
		                                                        // don't refactor it
			for (IProblemSequence ps : mSequences) {
				nodes.addAll(ps.getSubs());
			}
		}

		return nodes;
	}

	public XMLNode getXMLrep() {
		XMLNode seq = new XMLNode("<ProblemSequence />");

		for (XMLNode nd : getSubs()) {
			seq.insertChildNode(nd);
		}

		return seq;
	}
}
```

AnnotatedChildSection

```
package org.assistment.core.section;

import java.util.Map;

import org.assistment.framework.TypedValue.ITypedValue;

public class AnnotatedChildSection {
	ISection mSection;

	Map<String, ITypedValue> mProperties;

	public AnnotatedChildSection(ISection s, Map<String, ITypedValue> p) {
		mSection = s;
		mProperties = p;
	}
	public ISection getSection() {
		return mSection;
	}

	public void setProperties(Map<String, ITypedValue> props) {
		mProperties = props;
	}

	public Map<String, ITypedValue> getProperties() {
		return mProperties;
	}

	public void setProperty(String name, ITypedValue data) {
		mProperties.put(name, data);
	}

	public ITypedValue getProperty(String name) {
		return mProperties.get(name);
	}

	public boolean hasProperty(String name) {
		return mProperties.containsKey(name);
	}
}
```

SectionDynamicData

```
package org.assistment.core.section;

import java.util.HashMap;
import java.util.Map;

import org.assistment.framework.TypedValue.ITypedValue;

/**
 * A class to hold together any dynamic data that a section needs to store while
 * it is out of core memory. When paired with SectionStaticData, a section's
 * complete state should be able to be repoduced.
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public class SectionDynamicData {
	int mIndex;

	IProblemSequence mSequence;

	Map<String, ITypedValue> mProperties;
```

```
public SectionDynamicData() {
	mProperties = new HashMap<String, ITypedValue>();
	mIndex = 0;
	mSequence = null;
}

public void setIndex(int idx) {
	mIndex = idx;
}

public int getIndex() {
	return mIndex;
}

public void setSequence(IProblemSequence ps) {
	mSequence = ps;
}

public IProblemSequence getSequence() {
	return mSequence;
}

public void setProperties(Map<String, ITypedValue> props) {
	mProperties = props;
}

public Map<String, ITypedValue> getProperties() {
	if (null == mProperties) {
		mProperties = new HashMap<String, ITypedValue>();
	}
	return mProperties;
}

public void setProperty(String name, ITypedValue data) {
	if (null == mProperties) {
		mProperties = new HashMap<String, ITypedValue>();
	}
	mProperties.put(name, data);
}

public ITypedValue getProperty(String name) {
	if (null == mProperties) {
		mProperties = new HashMap<String, ITypedValue>();
	}
	return mProperties.get(name);
}

public boolean hasProperty(String name) {
	if (null == mProperties) {
			mProperties = new HashMap<String, ITypedValue>();
		}
		return mProperties.containsKey(name);
	}
}
```

SectionStaticData

```
package org.assistment.core.section;

import java.util.HashMap;
import java.util.Map;

import org.assistment.framework.TypedValue.ITypedValue;

/**
 * A class to hold together any static data that a section needs to store while
 * it is out of core memory. When paired with SectionDynamicData, a section's
 * complete state should be able to be repoduced.
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */

public class SectionStaticData {
	Map<String, ITypedValue> mProperties;

	public SectionStaticData() {
		mProperties = new HashMap<String, ITypedValue>();
	}

	public void setProperties(Map<String, ITypedValue> props) {
		mProperties = props;
	}

	public Map<String, ITypedValue> getProperties() {
		return mProperties;
	}

	public void setProperty(String name, ITypedValue data) {
		mProperties.put(name, data);
	}

	public ITypedValue getProperty(String name) {
		return mProperties.get(name);
	}

	public boolean hasProperty(String name) {
		return mProperties.containsKey(name);
	}
}
```

BasicSection

```
package org.assistment.core.section;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.runtime.logging.RuntimeLogger;

/**
 * This provides some base common functions many sections use and hooks to
 * override how it works. Its only purpose is to consolidate code.
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public abstract class BasicSection implements ISection {
	long mID; // Who we are.

	Map<String, ITypedValue> mProperties;

	List<AnnotatedChildSection> mChildren;

	int mIndex;

	BasicSection() {
		mIndex = 0;
	}

	/**
	 * If a section has static data to save, it should override this function.
	 *
	 * @return The data to save.
	 */
	protected SectionStaticData getStaticData() {
		return null;
	}

	public void saveStaticData(Map<Long, SectionStaticData> SDM) {
		SectionStaticData toSave = getStaticData();

		if (null != toSave) {
			SDM.put(mID, toSave);
		}

		for (AnnotatedChildSection achild : mChildren) {
			achild.getSection().saveStaticData(SDM);
		}
	}

	/**
	 * If a section has dynamic data to save, it should override this function.
	 *
	 * @return The data to save.
	 */
	protected SectionDynamicData getDynamicData() {
		return null;
	}

	public void saveDynamicData(Map<Long, SectionDynamicData> DDM) {
		SectionDynamicData toSave = getDynamicData();

		if (null != toSave) {
			DDM.put(mID, toSave);
		}

		for (AnnotatedChildSection achild : mChildren) {
			achild.getSection().saveDynamicData(DDM);
		}
	}

	public void loadData(Map<Long,SectionStaticData> SDM, Map<Long, SectionDynamicData> DDM)
	{
		if(null!=SDM.get(mID))
		{
			loadStaticData(SDM.get(mID));
		}

		if(null!=DDM.get(mID))
		{
			loadDynamicData(DDM.get(mID));
		}

		for(AnnotatedChildSection achild : mChildren)
		{
			achild.getSection().loadData(SDM, DDM);
		}
	}

	//clearly, if this are ever called, it should have an implimentation ...
	protected void loadDynamicData(@SuppressWarnings("unused")
	SectionDynamicData data)
```

```
        {
        }

        //clearly, if this are ever called, it should have an implimentation ...
        protected void loadStaticData(@SuppressWarnings("unused")
        SectionStaticData data)
        {
        }

        /**
         * By default this acts as the identity, pulling a single sequence and returning it.
         * @param achild The child (with metadata) to get the sequence from.
         * @param runtime The runtime you are running under.
         * @return A sequence to return.
         */
        protected IProblemSequence getSequenceFromChild(RuntimeLogger logger, List<Long> path,
AnnotatedChildSection achild) {
                return achild.getSection().getNextSequence(logger, path);
        }

        /**
         * A function so that sections can mark themselves as explicitely done.
         * By default always not finished to utilitise normal implicite finished only.
         *
         * @return Truth of our not-yet-finished-ness.
         */
        protected boolean notFinished() {
                return true;
        }

        public IProblemSequence getNextSequence(RuntimeLogger logger, List<Long> path) {
                while(notFinished()) {
                        if(mChildren.size()<=mIndex)
                                return null;

                        List<Long> subPath = new ArrayList<Long>(path);
                        subPath.add(mID);
                        IProblemSequence seq = getSequenceFromChild(logger, subPath,
mChildren.get(mIndex));

                        if(null!=seq)
                                return seq;
                        else
                                mIndex++; //Try the next child -- just loop around.
                }

                return null;
        }
}
```

ProblemSection

```
package org.assistment.core.section;

import static org.assistment.apps.utils.FLog.log;

import java.util.List;
import java.util.Map;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.LongValue;
import org.assistment.runtime.logging.RuntimeLogger;

/**
 * This section holds a single problem ID and returns it when requested. it is
 * responcible for this sole ID and nothing more.
 *
 * State 0 (not yet used) | v State 1 (Problem returned) |
 * +-----------------------+ | | V V State 2 (Problem right) State 3 (Problem
 * Wrong)
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public class ProblemSection implements ISection {
       long mID; // This is who we are.

       int mState; // This is deterministic state machine, this says where we are.

       long mProblemID; // Who we represent.

       public ProblemSection(long id, long pid) {
              mID = id;
              mState = 0;
              mProblemID = pid;
       }

       public ProblemSection(long id, Map<String, ITypedValue> props,
                     @SuppressWarnings("unused")
                     List<AnnotatedChildSection> k) {
              mID = id;
              mState = 0; // start out nowhere.
              if (null != props.get("ProblemID"))
                     mProblemID = ((LongValue) props.get("ProblemID")).getLong();
              else {
                     log(ProblemSection.class.getName(),
                                   "A ProblemSection did not have a problem id!!!");
```

```
                mState = 1; // Unknown result state.
            }
        }

        public void saveStaticData(Map<Long, SectionStaticData> SDM) {
            // empty methiod, never has children and never has static data.
        }

        public void saveDynamicData(Map<Long, SectionDynamicData> DDM) {
            SectionDynamicData SDD = new SectionDynamicData();

            SDD.setIndex(mState);

            DDM.put(mID, SDD);
        }

        public void loadData(Map<Long, SectionStaticData> SDM,
                Map<Long, SectionDynamicData> DDM) {
            if (null != DDM.get(mID))
                mState = DDM.get(mID).getIndex();
            else
                mState = 0;
        }

        public IProblemSequence getNextSequence(RuntimeLogger logger,
                List<Long> path) {
            path.add(mID);
            if (0 == mState) {
                mState = 1;
                IProblemSequence ps = new ProblemSequence(mProblemID);
                logger.logSequenceCreation(path, ps);
                return ps;
            } else
                return null;
        }

        public SectionType getType() {
            return SectionType.PROBLEM;
        }
}
```

LinearSection

```
package org.assistment.core.section;

import java.util.List;
import java.util.Map;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.ITypedValue;
```

```
/**
 * A LinearSection returns sequences as they are gotten from the children,
 * iterating in order over the children.
 *
 * Jun 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public class LinearSection extends BasicSection {
	public LinearSection(long id, Map<String, ITypedValue> p,
			List<AnnotatedChildSection> k) {
		mIndex = 0;
		mID = id;
		mProperties = p;
		mChildren = k;
	}

	@Override
	protected SectionDynamicData getDynamicData() {
		SectionDynamicData data = new SectionDynamicData();

		data.setIndex(mIndex);

		return data;
	}

	@Override
	protected void loadDynamicData(SectionDynamicData data) {
		mIndex = data.getIndex();
	}

	public SectionType getType() {
		return SectionType.LINEAR;
	}
}
```

RandomOrderSection

```
package org.assistment.core.section;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Random;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.IntValue;
```

```
/**
 * A RandomOrderSection returns sequences as they are gotten from the children,
 * iterating in over the children in a random order.
 *
 * Jan 2006 rewrite.
 *
 * @author Derek Radtke davean@wpi.edu, if I've left WPI try
 *         davean@isomerica.net or davean@sciesnet.net
 */
public class RandomOrderSection extends BasicSection {
       private int mSeed;

       public RandomOrderSection(long id, Map<String, ITypedValue> p,
                     List<AnnotatedChildSection> k) {
              mIndex = 0;
              mID = id;
              mProperties = p;
              mChildren = k;
       }

       @Override
       protected SectionStaticData getStaticData() {
              // Generate our static data.
              mSeed = new Random().nextInt();
              reorder();

              // Save it.
              SectionStaticData sd = new SectionStaticData();

              IntValue p = new IntValue();
              p.setInt(mSeed);

              sd.setProperty("Seed", p);

              return sd;
       }

       @Override
       protected void loadStaticData(SectionStaticData data) {
              // Get our seed back.
              mSeed = ((IntValue) data.mProperties.get("Seed")).getInt();

              // Repermute our order.
              reorder();
       }

       @Override
       protected SectionDynamicData getDynamicData() {
              SectionDynamicData dd = new SectionDynamicData();
```

```
		dd.setIndex(mIndex);

		return dd;
	}

	@Override
	protected void loadDynamicData(SectionDynamicData data) {
		mIndex = data.getIndex();
	}

	/**
	 * Reorders the section by the class seed.
	 */
	private void reorder() {
		Random random;
		List<AnnotatedChildSection> tmp;
		int index;

		random = new Random(mSeed);
		tmp = mChildren;
		mChildren = new ArrayList<AnnotatedChildSection>();

		while (tmp.size() > 0) {
			index = random.nextInt(tmp.size());

			mChildren.add(tmp.remove(index));
		}
	}

	public SectionType getType() {
		return SectionType.RANDOMORDER;
	}
}
```

RandomIteratorSection

```
package org.assistment.core.section;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Random;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.BoolValue;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.IntValue;
import org.assistment.framework.TypedValue.RandomValue;
import org.assistment.runtime.logging.RuntimeLogger;

public class RandomIteratorSection extends BasicSection {

	private boolean mCompleted;

	private Random mRand;

	private int mMaxNumber;

	public RandomIteratorSection(long id, Map<String, ITypedValue> p,
			List<AnnotatedChildSection> k) {
		mRand = new Random();
		mCompleted = false;
		mID = id;
		mProperties = p;
		mChildren = k;

		// Find the largest value in the children
		mMaxNumber = -1;
		for (AnnotatedChildSection section : mChildren) {
			IntValue v = (IntValue) section.getProperty("FromWhenBelow");

			if (null == v || null == v.getInt()) {
				continue;
			}

			int number = v.getInt();
			if (mMaxNumber < number) {
				mMaxNumber = number;
			}
		}

		if (mMaxNumber <= 0) {
			// No numbers assigned, assume all the same
			mMaxNumber = mChildren.size();

			// Assign temp values in memory so that there is no special case
			// behavior
			int count = 1;
			for (AnnotatedChildSection section : mChildren) {
				section.setProperty("FromWhenBelow", new IntValue(count));
				count++;
			}

			mMaxNumber = count - 1; // Fill in what our highest number is now
		}
	}

	/*
 * Choose a random number, go through sections untill you are less then the
 * number it mentions. If we try to pull from an EndIfEmpty section, and it
 * returns null, we are finished.
 */
@Override
public IProblemSequence getNextSequence(RuntimeLogger logger,
            List<Long> path) {
      while (notFinished()) {
            int trialNumber = mRand.nextInt(mMaxNumber);

            // We are looking for the first section with a number higher then
            // the one we choose
            for (Iterator<AnnotatedChildSection> iter = mChildren.iterator(); iter
                        .hasNext();) {
                  AnnotatedChildSection section = iter.next();

                  IntValue whenBelow = (IntValue) section
                              .getProperty("FromWhenBelow");

                  if (null == whenBelow || null == whenBelow.getInt()) {
                        iter.remove();
                        continue;
                  } // bad curriculum builder!

                  if (trialNumber < whenBelow.getInt()) {
                        List<Long> subPath = new ArrayList<Long>(path);
                        subPath.add(mID);
                        IProblemSequence seq = getSequenceFromChild(logger,
                                    subPath, section);

                        if (null != seq) {
                              return seq;
                        } else {
                              // Does this mean we are done?
                              BoolValue val = (BoolValue) section
                                          .getProperty("TerminateOnEmpty");
                              if (null != val && null != val.getBool()
                                          && true == val.getbool()) {
                                    mCompleted = true;
                              }

                              // One way or another remove it, we don't want to see it
                              // again.
                              iter.remove();

                              break;
                        }
                  }
            }
      }
```

```
			return null;
		}

		@Override
		protected boolean notFinished() {
			return !(mCompleted || mChildren.isEmpty());
		}

		@Override
		protected SectionDynamicData getDynamicData() {
			SectionDynamicData data = new SectionDynamicData();

			data.setProperty("RandomObject", new RandomValue(mRand));
			data.setProperty("Completed", new BoolValue(mCompleted));

			return data;
		}

		@Override
		protected void loadDynamicData(SectionDynamicData data) {
			BoolValue compVal = (BoolValue) data.mProperties.get("Completed");
			if (null == compVal) {
				mCompleted = false;
			} else {
				mCompleted = compVal.getbool();
			}

			RandomValue randVal = (RandomValue) data.mProperties
					.get("RandomObject");
			if (null != randVal && randVal.getRandom() != null) {
				mRand = randVal.getRandom();
			}
		}

		public SectionType getType() {
			return SectionType.RANDOMITERATOR;
		}
}
```

GroupNItemsSection

```
package org.assistment.core.section;

import java.util.List;
import java.util.Map;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.BoolValue;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.IntValue;
import org.assistment.runtime.logging.RuntimeLogger;

/**
 * GroupNItems takes
 */
public class GroupNItemsSection extends BasicSection {
	private int mNumToCombine;

	private boolean mDiscardRunts;

	public GroupNItemsSection(long id, Map<String, ITypedValue> p,
			List<AnnotatedChildSection> k) {
		mIndex = 0;
		mID = id;
		mProperties = p;
		mChildren = k;

		BoolValue runts = (BoolValue) mProperties.get("DiscardRunts");
		if (null != runts && null != runts.getBool()) {
			mDiscardRunts = runts.getBool();
		} else {
			mDiscardRunts = true;
		}

		IntValue nc = (IntValue) mProperties.get("NumToCombine");
		if (null != nc && null != nc.getInt()) {
			mNumToCombine = nc.getInt();
		} else {
			mNumToCombine = 1;
		}
	}

	@Override
	protected IProblemSequence getSequenceFromChild(RuntimeLogger logger,
			List<Long> path, AnnotatedChildSection achild) {
		IProblemSequence ps = new ProblemSequence();
		for (int bundled = 0; bundled < mNumToCombine; bundled++) {
			IProblemSequence newPS = achild.getSection().getNextSequence(
					logger, path);
			if (null == newPS) {
				if (mDiscardRunts || ps.isEmpty()) { // We don't care or it

	// doesn't really have

	// anything
					return null;
				} else {
					logger.logSequenceCreation(path, ps);
					return ps;
				}
			}

			ps.appendSequence(newPS);
		}

		logger.logSequenceCreation(path, ps);
		return ps;
	}

	protected SectionDynamicData getDynamicData() {
		SectionDynamicData data = new SectionDynamicData();

		data.setIndex(mIndex);

		return data;
	}

	@Override
	protected void loadDynamicData(SectionDynamicData data) {
		mIndex = data.getIndex();
	}

	public SectionType getType() {
		return SectionType.GROUP_N_ITEMS;
	}
}
```

ChooseOneSection

```
package org.assistment.core.section;

import java.util.List;
import java.util.Map;
import java.util.Random;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.IntValue;
import org.assistment.runtime.logging.RuntimeLogger;

public class ChooseOneSection extends BasicSection {
	@Override
	protected SectionStaticData getStaticData() {
		mIndex = new Random().nextInt(mChildren.size());

		SectionStaticData sd = new SectionStaticData();
		IntValue p = new IntValue();
		p.setInt(mIndex);
		sd.setProperty("WhichChild", p);
		return sd;
	}

	@Override
	protected void loadStaticData(SectionStaticData data) {
		mIndex = ((IntValue) data.getProperty("WhichChild")).getInt();
	}

	public ChooseOneSection(long id, Map<String, ITypedValue> p,
			List<AnnotatedChildSection> k) {
		mIndex = -1;
		mID = id;
		mProperties = p;
		mChildren = k;
	}

	@Override
	public IProblemSequence getNextSequence(RuntimeLogger logger,
			List<Long> path) {
		path.add(mID);
		return getSequenceFromChild(logger, path, mChildren.get(mIndex));
	}

	public SectionType getType() {
		return SectionType.CHOOSEONE;
	}
}
```

ConstraintNumeric

```
package org.assistment.core.section;

import java.util.List;
import java.util.Map;

import org.assistment.core.utils.SectionUtils.SectionType;
import org.assistment.framework.TypedValue.ITypedValue;
import org.assistment.framework.TypedValue.IntValue;
import org.assistment.runtime.logging.RuntimeLogger;

public class ConstraintNumericSection extends BasicSection {

	private int mNumSentOn;

	private int mNumToSendOn;

	public ConstraintNumericSection(long id, Map<String, ITypedValue> p,
			List<AnnotatedChildSection> k) {
		IntValue ntso = (IntValue) p.get("NumberToSendOn");
		if (null != ntso) {
			mNumToSendOn = ntso.getInt();
		} else {
			mNumToSendOn = 0;
			// TODO: LOG THIS.
		}

		mNumSentOn = 0;
		mIndex = 0;
		mID = id;
		mProperties = p;
		mChildren = k;
	}

	public SectionType getType() {
		return SectionType.CONSTRANT_NUMERIC;
	}

	@Override
	protected SectionDynamicData getDynamicData() {
		SectionDynamicData data = new SectionDynamicData();

		data.setIndex(mIndex);
		data.setProperty("NumSentOn", new IntValue(mNumSentOn));

		return data;
	}

	@Override
	protected void loadDynamicData(SectionDynamicData data) {
		mIndex = data.getIndex();
		IntValue nso = (IntValue) data.getProperty("NumSentOn");
		if (null == nso) {
			mNumSentOn = 0;
			// TODO: Log this!
		} else {
			mNumSentOn = nso.getInt();
		}
	}

	@Override
	protected IProblemSequence getSequenceFromChild(RuntimeLogger logger,
			List<Long> path, AnnotatedChildSection achild) {
		IProblemSequence ps = achild.getSection().getNextSequence(logger, path);
		if (null != ps) {
			mNumSentOn++;
		}

		return ps;
	}
	@Override
	protected boolean notFinished() {
		return (mNumToSendOn > mNumSentOn);
	}
}
```

assistment_sequence

```
#
# A class to store a heiercal sequence of Assistments and
# optional metadata associated with each assistment.
# This only gets saved into serialized data.
#
class AssistmentSequence

  attr_reader :sequence
  attr_writer :sequence

  def initialize(id=nil, meta={})
    @sequence = []
    if(id.class == Fixnum)
      @sequence.push({ "id"=>id, "meta"=>meta })
    elsif(id.class == AssistmentSequence)
      @sequence.push(id)
    elsif(id.nil?)
      #done
    else
      raise "Unhandled type in initialize (" << id.class.to_s << ")."
    end
  end

  # -----------------------------------------------------------------
  # Concatiante vs. Append gets horribly pedantic with definitions.
  # Tries to follow Ruby "Array" usage of the words.
  # -----------------------------------------------------------------

  # Takes the other sequence and attaches it sequenctialy making one
  # longer sequence both at the same level.
  # [a, b] concat [c, d] -> [a, b, c, d]
  def concatinate(addition)
    if(addition.class == Numeric or addition.class == Fixnum)
      addition = AssistmentSequence.new(addition)
    elsif(addition.class == AssistmentSequence)
      # already in correct form
    else
      raise "Problem appending unknown type to AssistmentSequence (" << addition.class.to_s << ")."
    end
    addition.sequence.each { |s| @sequence << s }
    return self
  end

  #Takes the sequence and adds it inside
  #[a, b] append [c, d] -> [a, b, [c, d]]
  def append(addition)
    if(addition.class == Numeric or addition.class == Fixnum)
      addition = AssistmentSequence.new(addition)
    elsif(addition.class == AssistmentSequence)
      # already in correct form
    else
      raise "Problem appending unknown type to AssistmentSequence (" << addition.class.to_s << ")."
    end
    @sequence << addition
    return self
  end

  # Takes this sequence and the other and makes them both sequenctial sequences in another
sequence.
  # [a, b] join [c, d] -> [[a, b], [c, d]]
  def join(second)
    if(second.class == Numeric)
      second = AssistmentSequence.new(second)
    elsif(second.class == AssistmentSequence)
      #already in correct form
    else
      raise "Problem appending unknown type to AssistmentSequence."
    end
    first = AssistmentSequence.new
    first.sequence = @sequence
    @sequence = []
    @sequence.push(first)
    @sequence.push(second)
    return self
  end

  def peak()
    if(0==@sequence.length)
      return nil
    elsif(@sequence[0].class == Hash)
      data = @sequence[0]
      id = data['id']
      return [id, data]
    elsif(@sequence[0].class == AssistmentSequence)
      id_t = @sequence[0].peak
      return id_t
    end
  end

  def pop()
    if(0 == @sequence.length)
      return nil
    elsif(@sequence.first.class == Hash)
      data = @sequence.shift
      id = data['id']
      return [id, data]
    elsif(@sequence.first.class == AssistmentSequence)
      id_t = @sequence.first.pop
      if(0 == @sequence.first.size)
        @sequence.shift
      end
      return id_t
    end
  end

  def to_s
    str = "("
    first_pass = true
    @sequence.each do |s|
      if(not first_pass)
        str << ", "
      else
        first_pass = false
      end

      if(s.class == Hash)
        str << '{' << s['id'].to_s << '}'
      elsif(s.class == AssistmentSequence)
        str << s.to_s
      else
        raise "Error!"
      end
    end
    str << ")"
  end

  # Returns how many subsequences there are.
  def size()
    return @sequence.length
  end

  # Counts the number of assistments in the sequence, the 'run length' of it.
  def length()
    count = 0
    @sequence.each do |s|
      if(s.class == Hash)
        # hashs contain only one Assistment
        count += 1
      elsif(s.class == AssistmentSequence)
        count += s.length

      else
        raise "Problem with an AssistmentSequence, bad type in array (" << s.class.to_s << ")."

      end
    end
    return count
  end
end
```

sequence

```
class Sequence < ActiveRecord::Base

  has_many :sections, :dependent => :destroy, :foreign_key => 'sequence_id'
  has_many :progresses, :dependent => :destroy

  belongs_to :base_section, :class_name => "Section", :foreign_key => 'head_section_id'

  attr_reader :created_at
  attr_reader :updated_at

  # Creates a progress for this student associated  with the sequence
  # and returns is
  def initProgress(student)
    # TODO: Start create, crash, restart invalid. should have a "finished init" flag? Better yet, a
proper transaction
    progress = Progress.new(:student_id => student.id, :sequence_id => self.id).set_defaults
    begin
      self.base_section.initProgress(progress)
    rescue SystemStackError # no sensical result, we are DONE here.
      problem_id = nil
    end
    # Not saving here since it will be saved as soon as it is something more then just a blank
progress.
    return progress
  end

  def progress(student)
       return student.progresses.find(:first, :conditions => ["sequence_id=?", self.id])
  end

  def nextProblem(student)
    progress = self.progress student
    if progress.nil? # no progress yet; initialize it.
      progress = self.initProgress(student)
    end

    # We have a progress here! And it is ready to use.
    # Get our sequence, ether from progress or get a new one
    if progress.state[:sequence]['sequence'].nil?
      begin
        sequence = self.base_section.nextSequence(progress)
      rescue SystemStackError
        # no sensical result, we are DONE here.
        problem_id = nil
        progress.is_done = true
      end
    else
      sequence = progress.state[:sequence]['sequence']
    end

    # Figure out what our next problem is
    unless sequence.nil?
      problem_id = sequence.pop
    else
      problem_id = nil
      progress.is_done = true
    end

    # Store our sequence if it has anything left in it so we can reuse it.
    if(sequence.nil? or 0 == sequence.length)
      progress.state[:sequence]['sequence'] = nil
    else
      progress.state[:sequence]['sequence'] = sequence
    end

    # We have now changed state, save this to the database.
    # Throw an exception on save failure as we want to state in the same state if we can't record
the change
    progress.save!

    return problem_id
  end

end

class Sequence < ActiveRecord::Base
  include SequenceBuilderUtils
end
```

section

```
class Progress < ActiveRecord::Base
  belongs_to :sequence
  belongs_to :student

  # A hash of SectionIDs to a hash of strings to types
  serialize :state

  validates_presence_of :sequence_id, :student_id
  def set_defaults
    self.is_done = false
    self.state = Hash.new
    self.state[:section] = Hash.new
    self.state[:sequence] = Hash.new
    self.state[:answer_history] = Hash.new # ProblemID => bool(correct)
    self
  end

end
```

section_link

```
class SectionLink < ActiveRecord::Base
  # this is here so that children have a predictable ordering
  acts_as_list :scope => :parent

  belongs_to :parent_section, :class_name => "Section", :foreign_key => 'parent_id'
  belongs_to :child_section, :class_name => "Section", :foreign_key => 'child_id'

  serialize :parameters

  before_create :set_defaults

  def after_find
   if self.parameters.nil?
    self.parameters = Hash.new
   end
  end

  private
  def set_defaults
   if self.parameters.nil?
    self.parameters = Hash.new
   end

   return true
  end
end
```

progress

```
class Progress < ActiveRecord::Base
  belongs_to :sequence
  belongs_to :student
  # A hash of SectionIDs to a hash of strings to types
  serialize :state

  validates_presence_of :sequence_id, :student_id

  def set_defaults
    self.is_done = false
    self.state = Hash.new
    self.state[:section] = Hash.new
    self.state[:sequence] = Hash.new
    self.state[:answer_history] = Hash.new # ProblemID => bool(correct)
    self
  end

end
```

problem_section

```
# This section type stores the actual referance to an assistment.
# It returns the assistment only once, and on subsiquent requests,
# returns that it is empty.
# Use this section type to include each assistment into a sequence.
class ProblemSection < Section
  SECTION_VARIABLES = {
    "AssistmentID" => {
      :display_name=>"Assistment",
      :widget=>"numeric",
      :default=>nil,
      :description=>"The ID number of the assistment that this section represents."
    },
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = true

  def name()
    return assistment.name
  end

  def assistment
    Assistment.find(self.getParam('AssistmentID'))
  end

  def initProgress(progress)
    super progress
    progress.state[:section][self.id]['Used']=false
  end

  def nextSequence(progress)
    if(!progress.state[:section][self.id]['Used'])
      progress.state[:section][self.id]['Used']=true
      return AssistmentSequence.new(self.getParam('AssistmentID'))
    else
      return nil
    end
  end

end
```

linear_section

```
# Returns problems from each child in order, only moving on
# when the current child reports it is finished.
class LinearSection < Section
  SECTION_VARIABLES = {
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::GetDirectFromChild
  include SectionMixin::IterateLinear
end
```

random_iterate_section

```
# Randomly reselects which unfinished child section to pull the next sequence from each time.
class RandomIterateSection < Section
  SECTION_VARIABLES = {
  }

  LINK_VARIABLES = {
    "Weight" => {
      :display_name=>"Weight",
      :widget=>"numeric",
      :default=>1,
      :description=>"The weight to give this child section when selecting which section to use.
Each child has a probability of (its weight/total weights)."
    },
    # Todo: Write test.
    "TermOnEmpty" => {
      :display_name=>"Finish when Finished",
      :widget=>"checkbox",
      :default=>false,
      :description=>"If when this child section is empty the parent section should finish or, if
false, continue with other sections that have problems remaining."
    },
  }

  IS_LEAF = false

  include SectionMixin::GetDirectFromChild
  include SectionMixin::IterateRandomIter
end
```

random_child_order_section

```
# Pulls a complete set of problems from each child,
# but when a child reports empty (and when selecting the first child)
# make a random choice of which previously unused child to pull from.
class RandomChildOrderSection < Section
  SECTION_VARIABLES = {
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::GetDirectFromChild
  include SectionMixin::IterateRandomNoRepeat
end
```

choose_one_condition_section

```
# Randomly selects one child from to pull Assistments from and only pulls from that one.
class ChooseConditionSection < Section
  SECTION_VARIABLES = {
  }

  LINK_VARIABLES = {
    "Weight" => {
      :display_name=>"Weight",
      :widget=>"numeric",
      :default=>1,
      :description=>"The weight to give this child section when selecting which section to choose.
Each child has a probability of (its weight/total weights)."
    },
  }

  IS_LEAF = false
  include SectionMixin::GetDirectFromChild
  include SectionMixin::IterateChooseCondition
end
```

no_repeats_section

```
# Section generaly acts like linear except dups are squashed and forgotten about.
class NoRepeatsSection < Section
  SECTION_VARIABLES = {
    #TODO: Impliment AllowRedo
    "AllowRedo" => {
      :display_name=>"Redo Wrong Assistments",
      :widget=>"checkbox",
      :default=>true,
      :description=>"Only suppress duplicate assistments that the student answered correctly."
    },
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::IterateLinear

  def getSequenceFromChild(progress, child)
    begin
      sequence = child.nextSequence(progress)
      if(sequence and !progress.state[:section][self.id]['returned'][sequence.to_s])
        progress.state[:section][self.id]['returned'][sequence.to_s]=true;
        return sequence
      end
    end until sequence.nil?
    return nil
  end

  def initProgress(progress)
    super progress
    # want the hash to default to returnign false because if then the trueth of
    # and element is if we sent it before.
    progress.state[:section][self.id]['returned']= Hash.new(anObject=false)
  end
end
```

numeric_limit_section

```
# Acts linear, but only returns a specified number of problems from
# all it's children combined.
```

```
class NumericLimitSection < Section
  SECTION_VARIABLES = {
    "NumToDisplay" =>{
      :display_name=>"Number of Assistments",
      :widget=>"numeric",
      :default=>0,
      :description=>"The number of sequences or assistments to send up through this section."
    },
    #TODO: Write a test for this
    "CountAssistments" => {
      :display_name=>"Count Assistments",
      :widget=>"checkbox",
      :default=>false,
      :description=>"Wether this section looks inside sequences and coutns the assistments
themselves or just counts the sequences."
    },
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::IterateLinear

  def getSequenceFromChild(progress, child)
    sequence = child.nextSequence(progress)

    if(nil!=sequence and
      progress.state[:section][self.id]['returned'] < self.getParam("NumToDisplay"))
      if(self.getParam("CountAssistments"))
        progress.state[:section][self.id]['returned'] += sequence.length;
      else
        progress.state[:section][self.id]['returned'] += 1;
      end
      return sequence
    else
      return nil
    end
  end

  def initProgress(progress)
    super progress
    progress.state[:section][self.id]['returned']=0
  end
end
```

temporal_limit_section

```
class TemporalLimitSection < Section
  SECTION_VARIABLES = {
    "SecondsToAllow" => {
      :display_name=>"Seconds to allow",
      :widget=>"numeric",
      :default=>0,
      :description=>"The number of Seconds to allow assistments to be pulled from this sectionb
once testing starts."
    },
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::IterateLinear

  def getSequenceFromChild(progress, child)
    if(!progress.state[:section][self.id]['start_seconds'])
      progress.state[:section][self.id]['start_seconds'] = Time.new.to_i
    end

    if((progress.state[:section][self.id]['start_seconds']+self.getParam("SecondsToAllow")) >
Time.new.to_i)
      return child.nextSequence(progress)
    else
      return nil
    end
  end
end
```

cat_three_p_term_confidence

```
class CatThreePTermConfidenceSection < Section
  SECTION_VARIABLES = {
    "TargetCAT3P" => {
      :display_name=>"CAT3P Section",
      :widget=>"numeric",
      :default=>0,
      :description=>"The section ID of the CAT test to work of of."
    },
    "DesiredSE" => {
      :display_name=>"Terminal Standard Error",
      :widget=>"numeric",
      :default=>0,
      :description=>"The standard error we need to get under to terminate."
    }
  }

  LINK_VARIABLES = {
  }

  IS_LEAF = false

  include SectionMixin::GetDirectFromChild
  include SectionMixin::IterateLinear

  def terminate(progress)
    return (self.SE(progress) <= self.getParam("DesiredSE"))
  end

end
```

cat_three_p_max_information

```
#Only works on single problem sets
class CatThreePMaxInformationSection < CatSections::CATBaseSection
  SECTION_VARIABLES = {
  }

  LINK_VARIABLES = {
    "a" => {
      :display_name=>"discrimination",
      :widget=>"numeric",
      :default=>0.0,
      :description=>""
    },
    "b" => {
      :display_name=>"difficulty",
      :widget=>"numeric",
      :default=>0.5,
      :description=>""
    },
    "c" => {
      :display_name=>"guessing",
      :widget=>"numeric",
      :default=>0.0,
      :description=>""
    },
  }

  IS_LEAF = false

  include SectionMixin::GetDirectFromChild
  include CatSections::CATPrioritise_IRT3parm
  include CatSections::CATSelect_best

  def nextSequence(progress)
    #keep trying until we run out
    while(!self.terminate(progress))
      next_index = self.select_next(progress)

      #there was nothing else to offer
      if(nil==next_index)
        return nil
      end

      child = self.child_sections[next_index]
      n = getSequenceFromChild(progress, child)

      if n.nil?
        progress.state[:section][self.id]['done_children'].add(next_index)
      else
        progress.state[:section][self.id]['assigned_problems'][n.peak[0]] = next_index
        return n
      end
    end

    return nil
  end
end
```

iterate_choose_condition

```
# This mixin selects a single child and only ever pulls from THE CHOOSEN ONE.
module SectionMixin::IterateChooseCondition
  def init_progress_iterate(progress)
    available = []
    # populate with all
    (0..(self.child_sections.size-1)).each { |id|
      # push weight copies in
      (1..self.getLinkParam(id,'Weight')).each{ available.push(id) }
    }

    choosen = available[rand(available.length)]

    progress.state[:section][self.id]['iterater_index']=choosen
  end

  def nextSequence(progress)
    return getSequenceFromChild(progress,
self.child_sections[progress.state[:section][self.id]['iterater_index']])
  end
end
```

iterate_random_no_repeate

```
# Makes a random choice of which previously unused child to pull from.
# Should the mapping be created each time or stored?
module SectionMixin::IterateRandomNoRepeat
  # the mapping is generated here and stored into the progress
  # only for coding expediance. It would be more efficient to have it generated
  # from a seed each time in the nextSequence probable (untested)
  def init_progress_iterate(progress)
    progress.state[:section][self.id]['iterater_index']=0
    index_list = Array.new
    (1..(self.child_sections.size)).each {|i| index_list << (i-1) }
    mapping = Array.new
    until index_list.empty?
      mapping << index_list[rand(index_list.length)]
      index_list.delete mapping.last
    end
    progress.state[:section][self.id]['iterater_mapping']=mapping
  end

  def nextSequence(progress)
    mapping = progress.state[:section][self.id]['iterater_mapping']
    while( self.child_sections[progress.state[:section][self.id]['iterater_index']])
      # looks up which child to use through the mapping.
      result = getSequenceFromChild(progress,
self.child_sections[mapping[progress.state[:section][self.id]['iterater_index']]])
      if(nil==result)
        # move on if this child is out; ++ doesn't work?
        progress.state[:section][self.id]['iterater_index'] += 1
      else
        return result
      end
    end
    # fall back if we run out of children, then we are out.
    return nil
  end
end
```

iterate_random_iter

```
# This module chooses each subsiquent sequence from a randomly choosen child.
# Some children can be marked such that gets are stopped if that one comes up empty.
# (Link variable Weight, TermOnEmpty)
module SectionMixin::IterateRandomIter
  def init_progress_iterate(progress)
    progress.state[:section][self.id]['iterater_done']=false
    progress.state[:section][self.id]['iterater_children_finished']={}
  end
  def nextSequence(progress)
    # get an array of valid keys (remove the invalidated ones)
    # These are the ones eligable to pull sequences from
    available = []
    # populate with all
    (0..(self.child_sections.size-1)).each { |id|
      # push weight copies in
      (1..self.getLinkParam(id,'Weight')).each{ available.push(id) }
    }
    # remove invalid
    progress.state[:section][self.id]['iterater_children_finished'].keys.each { |k|
      available.delete(k)
    }

    begin
      # die if we are done
      if(progress.state[:section][self.id]['iterater_done'])
        return nil
      else

        # nothing to pull from, finish
        if(0==available.size)
          progress.state[:section][self.id]['iterater_done'] = true
          return nil
        end

        # pull
        idx = available[rand(available.length)]
        s = getSequenceFromChild(progress, self.child_sections[idx])

        # did we get one? if not remove this one as a possability
        if(nil==s)
          progress.state[:section][self.id]['iterater_children_finished'][idx] = true
          # remove it from our current consideration
          available.delete(idx)
          if(self.getLinkParam(idx,'TermOnEmpty'))
            progress.state[:section][self.id]['iterater_done'] = true
            return nil
          end
        else
          return s
        end
      end
      # keep trying till we get a conclusive result (a sequence or we are done)
    end while true
  end

end
```

iterate_linear

```
# This mixin provides a simple iterator that returns all problems
# in the section's children in order from each section sequentialy,
# in the order that the child links occure.
module SectionMixin::IterateLinear
  def init_progress_iterate(progress)
    progress.state[:section][self.id]['iterater_index']=0
  end

  def nextSequence(progress)
    while( self.child_sections[progress.state[:section][self.id]['iterater_index']])
      result = getSequenceFromChild(progress,
self.child_sections[progress.state[:section][self.id]['iterater_index']])
      if(nil==result)
        # move on if this child is out; ++ doesn't work?
        progress.state[:section][self.id]['iterater_index'] += 1
      else
        return result
      end
    end
    # fall back if we run out of children, then we are out.
    return nil
  end
end
```

get_dirrect_from_child

```
# This mixin impliments the simplist method of getting a problem from a child.
# It mearly grabs a single problem and returns it dirrectly.
module SectionMixin::GetDirectFromChild

  def getSequenceFromChild(progress, child)
    if self.terminate(progress)
      return nil
    else
      return child.nextSequence(progress)
    end
  end

end
```

cat_sections

```
module CatSections

  #Provides hooks for te more complicated CAT sections to ease developement.
  class CATBaseSection < Section
  def initProgress(progress)
    super progress
    init_progress_prioritise(progress)
    init_progress_select(progress)
  end
end

#Developed from http://edres.org/scripts/cat/
module CATPrioritise_IRT3parm
  def init_progress_prioritise(progress)
    progress.state[:section][self.id]['assigned_problems']= Hash.new
    progress.state[:section][self.id]['done_children']= Set.new
  end

  def sorted(progress)
    #get an array of valid keys (remove the invalidated ones)
    #These are the ones eligable to pull sequences from
    available = []
    #populate with all
    (0..(self.child_sections.size-1)).each { |id|
      available.push(id)
    }
    #remove invalid
    progress.state[:section][self.id]['done_children'].each { |k|
      available.delete(k)
    }

    theta_hat = self.theta_hat(progress)
    available.sort! {|b,a| self.I(a,theta_hat) <=> self.I(b,theta_hat) }

    #print "¥nsorted:¥n"
    #available.each { |cid|
    #  print cid.to_s+": "+self.I(cid,theta_hat).to_s+"¥n"
    #}
    #print "¥n"

    return available
  end

  def P(i,progress)
    theta = self.theta_hat(progress)
    return self.Pof(i,theta)
  end

  def Pof(i, theta)
    a = self.getLinkParam(i,'a')
    b = self.getLinkParam(i,'b')
    c = self.getLinkParam(i,'c')
    return c + (1-c)/(1+Math.exp(-1.7*a*(theta-b)))
  end
```

```
    def I(i, theta_hat)
      a = self.getLinkParam(i,'a')
      b = self.getLinkParam(i,'b')
      c = self.getLinkParam(i,'c')
      p = self.Pof(i,theta_hat)
      return ((a*a)*((1-p)/p)) * (((p-c)**2)/((1-c)**2))
    end

    #TODO: Replace with analytical solution
    #Uses numerical methods solution ATM due to the desire to avoid errors when writing first
version
    def dPof(i,theta)
      e = 0.0000001
      return (self.Pof(i,theta+e)-self.Pof(i,theta-e))/(2*e)
    end

    def fitness(hat, progress)
          sum = 0
          progress.state[:section][self.id]['assigned_problems'].keys.each { |iOrig|
            i = progress.state[:section][self.id]['assigned_problems'][iOrig]
            #make sure that it was answered
            if(progress.state[:answer_history].has_key?(iOrig))
              if(progress.state[:answer_history][iOrig])
                correctness = 1
              else
                correctness = 0
              end
              sum = sum + ((correctness - self.Pof(i, hat))*(self.dPof(i, hat)/(self.Pof(i,hat)*(1-
self.Pof(i,hat)))))
            end
          }

      return sum
    end

    #Brute force estiamtion is bad. Don't do it.
    def theta_hat(progress)
      #print "Calculating theta_hat¥n"
      best_hat = 0
      fit = 99999
      if 0 < progress.state[:section][self.id]['assigned_problems'].length
        hat = -3
        while hat <= 3 do
          val = self.fitness(hat,progress)

          #print hat.to_s+","+val.to_s+"¥n"

          if(val.abs < fit)
           best_hat = hat
           fit = val.abs
          end

          #print "Hat: "+hat.to_s+" sum: "+sum.to_s+"¥n"

          hat = hat + 0.01
        end
      end

      #clamp hat to the range (-3, 3)

      return best_hat
    end

    #Standard Error
    def SE(progress)
      irr = 0
      hat = self.theta_hat(progress)
      #Use a set to avoid dups
      Set.new(progress.state[:section][self.id]['assigned_problems'].keys).each { |iOrig|
        #find out what it is in our selves
        i = progress.state[:section][self.id]['assigned_problems'][iOrig]
        #make sure that it was answered
        if(progress.state[:answer_history].has_key?(i))
          #They did part of this test! Let us learn something about them.
          irr = irr + self.I(i,hat)
        end
      }

      #Give a failback that will probably be an unsatisfactory SE if things are undefined.
      if(0==irr)
        irr = 0.001
      end

      se = 1/Math.sqrt(irr)

      return se
    end
  end

  module CATSelect_best
    def init_progress_select(progress)
    end

    #Just return the best one
    def select_next(progress)
      sorted = self.sorted(progress)

      if(0<sorted.length)
        return sorted[0]
      else
        return nil
      end
    end
  end

end
```

section_tests

```
require File.dirname(__FILE__) + '/../test_helper'

def withing_small_margin(a,b)
 e = 0.08
 return ((a+e)>b) && ((a-e)<b)
end

class SectionTest < Test::Unit::TestCase
  fixtures :sections
  fixtures :section_links

  # Replace this with your real tests.
  def test_CheckCRUD
   s = LinearSection.create! :sequence_id => 1
  end

#  def test_CAT3_proper_estimator
#   s = sections(:CAT_CV)
#
#   print "Start CAT proper¥n"
#
#   (-3..3).each{ |i|
#     print "For i="+i.to_s
#     p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
#     s.initProgress(p)
#     while (r = s.nextSequence(p))
#       assert(1==r.size)
#       prob = r.pop[0]
#       if s.Pof(p.state[:section][s.id]['assigned_problems'][prob],1)> 0.50
#         p.state[:answer_history][prob] = true
#       else
#         p.state[:answer_history][prob] = false
#       end
#       #print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
#       #print "standard error = " + s.SE(p).to_s+"¥n"
#     end
#
#   print " we got "+s.theta_hat(p).to_s+".¥n"
#   assert(withing_small_margin(i,s.theta_hat(p)))
#  }
#  end

  #Sample run: (Target z=0.5)
  # #: responce, estimate, standard error
  # _: _______,     0.5
  # 1: correct,     3.0, 6.0
  # 2: incorrect, -3.0, 1.65
  # 3: correct,    1.06, 1.34
  # 4: incorrect, 0.67, 0.74
  # 5: incorrect, 0.31, 0.48
  # 6: correct,    0.42, 0.37
  # 7: correct,    0.56, 0.30
  #
  def test_CAT3_perfect
   s = sections(:CAT3)
   p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
   s.initProgress(p)

   print "Start CAT perfect¥n"

   assert(withing_small_margin(0,s.theta_hat(p)))
   #print s.I(0,0).to_s+"¥n"

   r = s.nextSequence(p)
   assert(1==r.size)
   assert(1==r.pop[0])
   p.state[:answer_history][1] = true
   print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
   assert(withing_small_margin(3,s.theta_hat(p)))
   #print "standard error = " + s.SE(p).to_s+"¥n"
   #assert(withing_small_margin(6,s.SE(p)))

   r = s.nextSequence(p)
   assert(1==r.size)
   assert(2==r.pop[0])
   p.state[:answer_history][2] = false
   print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
   assert(withing_small_margin(-3,s.theta_hat(p)))
   #print "standard error = " + s.SE(p).to_s+"¥n"
   #assert(withing_small_margin(1.65,s.SE(p)))

   r = s.nextSequence(p)
   assert(1==r.size)
   assert(3==r.pop[0])
   p.state[:answer_history][3] = true
   print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
   assert(withing_small_margin(1.06,s.theta_hat(p)))

   r = s.nextSequence(p)
```

```
 assert(1==r.size)
 assert(4==r.pop[0])
 p.state[:answer_history][4] = false
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(0.67,s.theta_hat(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(5==r.pop[0])
 p.state[:answer_history][5] = false
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(0.31,s.theta_hat(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(6==r.pop[0])
 p.state[:answer_history][6] = true
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(0.42,s.theta_hat(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(7==r.pop[0])
 p.state[:answer_history][7] = true
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(0.56,s.theta_hat(p)))

end

# #: responce, estimate, standard error
# _:   _______,    0.5,
# 1:   correct,    3.0, 6.0
# 2:   incorrect, -3.0, 1.65
# 3:   correct,   1.06, 1.34
# 4:   incorrect, 0.67, 0.74
# 5:   incorrect, 0.31, 0.48
# 6:   incorrect, 0.01, 0.46
# (8): correct,   0.10, 0.37
def test_CAT3_deviation
 s = sections(:CAT3)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)

 print "Start CAT deviation¥n"

 assert(withing_small_margin(0,s.theta_hat(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
```

```
p.state[:answer_history][1] = true
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(3,s.theta_hat(p)))
#print "standard error = " + s.SE(p).to_s+"¥n"
#assert(withing_small_margin(6,s.SE(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(2==r.pop[0])
p.state[:answer_history][2] = false
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(-3,s.theta_hat(p)))
#print "standard error = " + s.SE(p).to_s+"¥n"
#assert(withing_small_margin(1.65,s.SE(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(3==r.pop[0])
p.state[:answer_history][3] = true
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(1.06,s.theta_hat(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(4==r.pop[0])
p.state[:answer_history][4] = false
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(0.67,s.theta_hat(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(5==r.pop[0])
p.state[:answer_history][5] = false
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(0.31,s.theta_hat(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(6==r.pop[0])
p.state[:answer_history][6] = false
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(0.01,s.theta_hat(p)))

r = s.nextSequence(p)
assert(1==r.size)
assert(8==r.pop[0])
p.state[:answer_history][8] = true
print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
assert(withing_small_margin(0.10,s.theta_hat(p)))
```

```
end

# #: responce, estimate, standard error
# _:   _______,    0.5,
# 1:   incorrect, -3.0, 6.0
# 3:   correct,   -1.32, 4.09
def test_CAT3_early_deviation
 s = sections(:CAT3)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)

 print "Start CAT early deviation¥n"

 assert(withing_small_margin(0,s.theta_hat(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 p.state[:answer_history][1] = false
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(-3,s.theta_hat(p)))
 #print "standard error = " + s.SE(p).to_s+"¥n"
 #assert(withing_small_margin(6,s.SE(p)))

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 p.state[:answer_history][3] = true
 print "Theta hat = " + s.theta_hat(p).to_s+"¥n"
 assert(withing_small_margin(-1.32,s.theta_hat(p)))

end

def test_ProblemSectionDoesNotReturnID
 s = sections(:zero)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(747==r.pop[0])
 assert(nil==s.nextSequence(p))
end

def test_ProblemSection
 s = sections(:one)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 assert(nil==s.nextSequence(p))
end

def test_LinearSection
 s = sections(:Linear1_2_3)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(2==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 assert(nil==s.nextSequence(p))
end

#this test runs a random curriculum twice,
#each time it runs it, it checks that each number
#came up that was in it
#and that the curriculum was out after that
#and that the curriculum returned the right number of assistments.
#It then, having stored which request each assistment id came up on
#makes sure that the runs where not identical.
#
#Admittedly, there exists no good way to test for randomness.
#This doesn't even make a good attempt at checkign that something
#wasn't a fluke, it DOES check that it doesn't repeat.
#If by chance it does, it doesn't handle that.
def test_RandomOrderSection
 s = sections(:Random1_to_9)

 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 run1 = Hash.new
 (1..9).each do |i|
    r = s.nextSequence(p)
    assert(1==r.size)
    run1[r.pop[0]]=i
 end
 assert(nil==run1[0])
 assert(nil!=run1[1])
 assert(nil!=run1[2])
 assert(nil!=run1[3])
 assert(nil!=run1[4])
 assert(nil!=run1[5])
 assert(nil!=run1[6])
 assert(nil!=run1[7])
 assert(nil!=run1[8])
 assert(nil!=run1[9])
 assert(nil==s.nextSequence(p))

 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 run2 = Hash.new
 (1..9).each do |i|
    r = s.nextSequence(p)
    assert(1==r.size)
    run2[r.pop[0]]=i
 end
 assert(nil==run2[0])
 assert(nil!=run2[1])
 assert(nil!=run2[2])
 assert(nil!=run2[3])
 assert(nil!=run2[4])
 assert(nil!=run2[5])
 assert(nil!=run2[6])
 assert(nil!=run2[7])
 assert(nil!=run2[8])
 assert(nil!=run2[9])
 assert(nil==s.nextSequence(p))

 #make sure they differed
 times_not_equal = 0
 (1..9).each do |i|
    if(run1[i]!=run2[i])
     times_not_equal += 1
    end
 end

 assert(0<times_not_equal)
end

def test_NoRepeatsSection
 s = sections(:NoRepeat0)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(2==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 assert(nil==s.nextSequence(p))
end
```

```
def test_NumericLimitSection
 s = sections(:NumericLimit0)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(2==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 assert(nil==s.nextSequence(p))
end

def test_ChooseConditionSection
 s = sections(:ChooseCondition0)

 values = {}
 #make sure it occasionaly chooses all of them
 (0..15).each { |i|
   p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
   s.initProgress(p)
   r = s.nextSequence(p)
   assert(1==r.size)
   values[r.pop[0]] = 1
   assert(nil==s.nextSequence(p))
 }

 (1..3).each { |i|
   assert(1==values[i])
  }
end

def test_ChooseConditionSectionWeighted
 s = sections(:ChooseConditionW)

 values = []
 values[1] = 0
 values[2] = 0
 #make sure it occasionaly chooses all of them
 (1..1000).each { |i|
   p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
   s.initProgress(p)
   r = s.nextSequence(p)
   assert(1==r.size)
   values[r.pop[0]] += 1
   assert(nil==s.nextSequence(p))
 }

 assert((values[1]>50) && (values[1]<150))
 assert((values[2]>850) && (values[2]<950))
 assert(1000==(values[1]+values[2]))
end

def test_GroupSequencesSectionPartials
 s = sections(:GroupSequencesFromChildPartials)

 #First make sure it returns everything
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)

 r = s.nextSequence(p)
 assert(2==r.length)
 assert(2==r.size)
 assert(1==r.pop[0])
 assert(2==r.pop[0])

 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 assert(nil==s.nextSequence(p))
end

def test_GroupSequencesSectionNoPartials
 s = sections(:GroupSequencesFromChildNoPartials)

 #First make sure it returns everything
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)

 r = s.nextSequence(p)
 assert(2==r.length)
 assert(2==r.size)
 assert(1==r.pop[0])
 assert(2==r.pop[0])

 assert(nil==s.nextSequence(p))
end

def test_TemporalLimitSection
 s = sections(:TemporalLimit0)

 #First make sure it returns everything
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
```

```
 assert(1==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(2==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(3==r.pop[0])
 assert(nil==s.nextSequence(p))

 #Then that it stops after time correctly
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 sleep(1)
 assert(nil==s.nextSequence(p))

 #Then that it always starts
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 sleep(1)
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(1==r.pop[0])
 r = s.nextSequence(p)
 assert(1==r.size)
 assert(2==r.pop[0])
 sleep(1)
 assert(nil==s.nextSequence(p))

end

def test_TestGettingOfDefaultParameter
 s = sections(:NumericLimit1)
 p = Progress.new(:student_id => 0, :sequence_id => 0).set_defaults
 s.initProgress(p)
 assert(nil==s.nextSequence(p))
end

def test_SectionCreation_ChildForLeaf
 s = ProblemSection.new
 s.extend SectionBuilderUtils

 c = LinearSection.new


 errored = false

 begin
```

```
   s.addChild(c)
  rescue
   errored = true
  end

  assert(errored)
 end

 def test_SectionCreation_ChildForNonLeaf
  s = LinearSection.new
  s.extend SectionBuilderUtils

  c = LinearSection.new

  worked = true

  s.addChild(c)

  assert(worked)
 end

 def test_SectionCreation_SectionParameters
  s = GroupSequencesSection.new

  s.extend SectionBuilderUtils

  failed = false
  begin
   s.parameter("NumberToGroup", 2)
  rescue
   failed = true
  end
  assert(failed)

  worked = true
  begin
   s.parameter("NumToGroup", 2)
  rescue
   print $!,"¥n"
   worked = false
  end
  assert(worked)

 end

 #need test for Termiante On Empty
 #Ech! horrible test case!
 #Uses build for performance reasons
 def test_RandomIterSection
  s = sections(:RandomIterNoWeights)
  counts = Hash.new(0)

  #make sure it has the right items and number of them in a valid sequence
  #500 times means it should hit all the paths
  (1..12000).each { |u|
    p = Progress.new(:state=>{:section => {}})
    s.initProgress(p)

    entries = Hash.new(false)
    entries[1] = true
    entries[2] = true
    entries[3] = true
    entries[4] = true
    entries[5] = true

    seq_rec = ""

    (1..5).each { |i|
        r = s.nextSequence(p)
        assert(nil!=r)
        val = r.pop[0]
        assert(entries[val])
        entries[val]=false

        #There was only one there
        assert(nil==r.pop)

        seq_rec << val.to_s
     }
    assert(nil==s.nextSequence(p))
    counts[seq_rec] += 1
   }

   #make sure they are all within a close range of each other
   counts.each{ |seq,count|
     #so large because the ruby random nubmer generator sucks.
     assert((count>=60) && (count<=140), seq+"("+count.to_s+")")
    }
 end

 #need test for Termiante On Empty
 #Ech! horrible test case!
 #Uses build for performance reasons
 def test_RandomIterSectionWeighted
  s = sections(:RandomIterWeights)

  counts = Hash.new(0)

  #make sure it has the right items and number of them in a valid sequence
```

```
   #500 times means it should hit all the paths
   (1..10000).each { |u|
     p = Progress.new(:state=>{:section => {}})
     s.initProgress(p)

     entries = Hash.new(false)
     entries[1] = true
     entries[2] = true
     entries[3] = true
     entries[4] = true
     entries[5] = true

     seq_rec = ""

     (1..2).each { |i|
         r = s.nextSequence(p)
         assert(nil!=r)
         val = r.pop[0]
         assert(entries[val])
         entries[val]=false

         #There was only one there
         assert(nil==r.pop)

         seq_rec << val.to_s
      }
     assert(nil==s.nextSequence(p))
     counts[seq_rec] += 1
    }
    assert((counts["12"]<1250) && (counts["12"]>750))
    assert((counts["21"]<9250) && (counts["21"]>8750))
  end
end
```

assistment_sequence_test

```
require File.dirname(__FILE__) + '/../test_helper'

class AssistmentSequenceTest < Test::Unit::TestCase
  def test_SingleProblemOperation
    s = AssistmentSequence.new(3)
    assert(1==s.size)
    assert(1==s.length)
    assert(3==s.pop[0])
    assert(nil==s.pop)
  end

  def test_concatination
   s0 = AssistmentSequence.new(1)
```

```
 s0.concatinate(AssistmentSequence.new(2))
 s0.concatinate(3)

 assert(3==s0.size)
 assert(3==s0.length)

 s1 = AssistmentSequence.new(4)
 s1.concatinate(AssistmentSequence.new(5))
 s1.concatinate(s0)

 assert(5==s1.size)
 assert(5==s1.length)

 assert(4==s1.pop[0])
 assert(5==s1.pop[0])
 assert(1==s1.pop[0])
 assert(2==s1.pop[0])
 assert(3==s1.pop[0])
 assert(nil==s1.pop)
end

def test_append
 s0 = AssistmentSequence.new(1)
 s0.append(AssistmentSequence.new(2))
 s0.append(3)

 assert(3==s0.size)
 assert(3==s0.length)

 s1 = AssistmentSequence.new(4)
 s1.concatinate(AssistmentSequence.new(5))
 s1.append(s0)

 assert(3==s1.size)
 assert(5==s1.length)

 assert(4==s1.pop[0])
 assert(5==s1.pop[0])
 assert(1==s1.pop[0])
 assert(2==s1.pop[0])
 assert(3==s1.pop[0])
 assert(nil==s1.pop)
end

def test_join
 s0 = AssistmentSequence.new(1)
 s0.append(AssistmentSequence.new(2))
 s0.append(3)

 s1 = AssistmentSequence.new(4)
```

```
    s1.concatinate(AssistmentSequence.new(5))

    s0.join(s1)

    assert(2==s0.size)
    assert(5==s0.length)

    assert(1==s0.pop[0])
    assert(2==s0.pop[0])
    assert(3==s0.pop[0])
    assert(4==s0.pop[0])
    assert(5==s0.pop[0])
    assert(nil==s0.pop)
  end

  def test_pop
    s = AssistmentSequence.new(AssistmentSequence.new(1))
    assert(1==s.pop[0])
    assert(nil==s.pop)
  end

  def test_length_size
    s =
AssistmentSequence.new(AssistmentSequence.new(AssistmentSequence.new(AssistmentSequence.new(1).conc
atinate(2))))
    assert(2==s.length)
    assert(1==s.size)
  end
end
```

progress_test

```
require File.dirname(__FILE__) + '/../test_helper'

class ProgressTest < Test::Unit::TestCase
  fixtures :progresses

  #How sad that this test is the most important one for any of the models?
  def test_CheckCRUD
    p = Progress.create!
    assert(!p.is_done)
    p.is_done = true
    assert(p.is_done)
    p.save!

    id = p.id
    p = Progress.find(id)
    assert(p.is_done)
    p.destroy
    begin
     p = Progress.find(id)
     assert(false)
    rescue
     assert(true)
    end

    p = Progress.create! :is_done=>true
    p.state[:section][0] = 1
    p.save!

    assert(1 == p.state[:section][0])

    id = p.id
    p = Progress.find(id)
    #print p.to_yaml
    assert(1 == p.state[:section][0])
  end
end
```

sequence_test

```
require File.dirname(__FILE__) + '/../test_helper'

class SequenceTest < Test::Unit::TestCase
  fixtures :sequences
  fixtures :sections
  fixtures :progresses

  # Replace this with your real tests.
  #def test_CheckCRUD
  # s = Sequence.new(:name => "Blank Test Sequence")
  #end

  #make sure a curriculum can find it's head problem,
  #generate a progress and get a problem.
  #This is the very basic functionality.
  def test_SimpleSingleSection
   c = sequences(:problem_section_1)
   s = Student.create!
   assert(1==c.nextProblem(s)[0])
   assert(nil==c.nextProblem(s))
  end

  #This tests what happens when a section contains it's self as a child.
  #This is to make sure that if an error DOES occure, the code handles it
  #and doesn't halt the system
#  def test_SectionLoop
#   c = curriculums(:Loop)
#   assert(nil==c.nextProblem(nil))
#  end

  def test_SequenceSequenceUsage
    c = sequences(:SequencedCurriculum)
    print c.nextProblem(nil)
    print c.nextProblem(nil)
    print c.nextProblem(nil)
    assert(1==c.nextProblem(nil))
    assert(2==c.nextProblem(nil))
    assert(nil==c.nextProblem(nil))
  end

  def test_SequenceSequenceUsage
    c = sequences(:PrettyCuric0)
    c.extend SequenceBuilderUtils
    #print c.DOTfile
    assert(c.DOTfile ==
    "digraph g {¥n"+
    " graph [ rankdir = ¥"LR¥" ];¥n"+
    " node [ fontsize = ¥"16¥" shape = ¥"ellipse¥" ];¥n"+
    " edge [ ];¥n"+
    " section_id_93 [ label = ¥"<fName>Section #93 |<fType>ProblemSection¥" shape = ¥"record¥"
];¥n"+
    " section_id_94 [ label = ¥"<fName>Section #94 |<fType>NumericLimitSection |<f1>¥" shape =
¥"record¥" ];¥n"+
    " section_id_90 [ label = ¥"<fName>Section #90 |<fType>LinearSection |<f1> |<f2> |<f3> |<f4>¥"
shape = ¥"record¥" ];¥n"+
    " section_id_91 [ label = ¥"<fName>Section #91 |<fType>ProblemSection¥" shape = ¥"record¥"
];¥n"+
    " section_id_92 [ label = ¥"<fName>Section #92 |<fType>ProblemSection¥" shape = ¥"record¥"
];¥n"+
    " section_id_94:f1 -> section_id_92:fName [ id = 1 ];¥n"+
    " section_id_90:f1 -> section_id_91:fName [ id = 2 ];¥n"+
    " section_id_90:f2 -> section_id_92:fName [ id = 3 ];¥n"+
    " section_id_90:f3 -> section_id_94:fName [ id = 4 ];¥n"+
    " section_id_90:f4 -> section_id_93:fName [ id = 5 ];¥n"+
    "};");
  end
end
```

Appendix D – Converter from New Java to Ruby

```
$LOAD_PATH << File.expand_path(File.dirname(__FILE__) + "/..")

require 'yaml'
require "rexml/document"

project_config = YAML.load_file('../config/project.yml')
require project_config['rails_project']['path'] + '/config/environment'

require 'models/java/curriculum'

def convert_sequence(old_sequence)
  #keep a mapping for conversion the new global ID space
  #Old ones where implicitely indexed, so guarrentied 0->whatever without holes
  old_ids_new_ids = Array.new

  sections = REXML::Document.new old_sequence.obj_xml

  seq = Sequence.new :create_at => old_sequence.creationdate,
                     :updated_at => DateTime::now(),
                     :name => old_sequence.name,
                     :description => old_sequence.description
  seq.id = old_sequence.id

  seq.save!

  sections.elements.each("sections/isection") { |section|
   #print section.to_s + "\n"

   type = section.attributes['type']

   if    "Problem" == type
    throw "Unexpected existence of ProblemSection!"
    #nsection = ProblemSection.new :sequence_id=> seq.id,
    #                              :parameters => {'AssistmentID' => }
   elsif "ChooseOne" == type
    nsection = ChooseConditionSection.create! :sequence_id => seq.id,
                                              :create_at =>
old_sequence.creationdate
     section.elements.each("TypedProperties/tproperty") { |property|
      print "Section Property: " + property.to_s + "\n"
     }
   elsif "Linear" == type
    nsection = LinearSection.create! :sequence_id => seq.id,
                                     :create_at => old_sequence.creationdate
   #mind you, "Random" as a type wouldn't have actually compiled right in the old
system
   #so it wasn't usable then ... see id 2567.
   elsif "RandomOrder" == type or "Random" == type
    nsection = RandomChildOrderSection.create! :sequence_id => seq.id,
                                               :create_at =>
old_sequence.creationdate
     section.elements.each("TypedProperties/tproperty") { |property|
      print "Section Property: " + property.to_s + "\n"
     }
   elsif "RandomIterator" == type
    nsection = RandomIterateSection.create! :sequence_id => seq.id,
                                            :create_at => old_sequence.creationdate
     section.elements.each("TypedProperties/tproperty") { |property|
      print "Section Property: " + property.to_s + "\n"
     }
   elsif "ConstraintNumeric" == type
    nsection = NumericLimitSection.create! :sequence_id => seq.id,
                                           :create_at => old_sequence.creationdate,
                                           :parameters => {
                                                           "NumToDisplay" =>
section.elements["TypedProperties/tproperty[@name='NumberToSendOn']"].attributes["v
al"].to_i
                                                          }
     section.elements.each("TypedProperties/tproperty") { |property|
      print "Section Property: " + property.to_s + "\n"  unless "NumberToSendOn" ==
property.attributes['name']
     }
   elsif "GroupNItems" == type
    nsection = GroupSequencesSection.create! :sequence_id => seq.id,
                                             :create_at =>
old_sequence.creationdate,
                                             :parameters => {
                                                             "NumToGroup" =>
section.elements["TypedProperties/tproperty[@name='NumToCombine']"].attributes["val
"].to_i
                                                             #"ThrowAwayPartials"
=>
section.elements["TypedProperties/tproperty[@name='NumToCombine']"].attributes["val
"]
                                                            }
     section.elements.each("TypedProperties/tproperty") { |property|
      print "Section Property: " + property.to_s + "\n"  unless "NumToCombine" ==
property.attributes['name']
     }
   else
    throw "Unknown section type!"
   end

   #nsection.save!
   old_ids_new_ids << nsection.id
  }

  index = 0
  sections.elements.each("sections/isection") { |section|
   position = 0
   weight = 0
   section.elements.each("children/child") { |child|
    #print "\n\n\n++++++\n"+child.to_s+"\n++++++\n\n\n"

    #section IDs over 1k are cleary a mistake, they have to be supposed to be
problem IDs, correct
    if "problem" == child.attributes['type'] or child.attributes['id'].to_i > 1000
     #create a new ProblemSection and link to that
     nsection = ProblemSection.create! :sequence_id => seq.id,
                                       :create_at => old_sequence.creationdate,
                                       :parameters => {'AssistmentID' =>
child.attributes['id'].to_i}
     SectionLink.create! :parent_id => old_ids_new_ids[index],
                         :child_id => nsection.id,
                         :position => position
    elsif "section" == child.attributes['type']
     properties = Hash.new

     child.elements.each("TypedProperties/tproperty") { |property|
      if "TerminateOnEmpty" == property.attributes['name']
       properties['TermOnEmpty'] = unless "False" == property.attributes['val']
then true else false end
      elsif "FromWhenBelow" == property.attributes['name']
       w = property.attributes['val'].to_i
       properties['Weight'] = w - weight
       weight = w
      else
       print "Property: " + property.to_s + "\n"
      end
     }

     SectionLink.create! :parent_id => old_ids_new_ids[index],
                         :child_id => old_ids_new_ids[child.attributes['id'].to_i],
                         :position => position,
                         :parameters => properties
    else
     throw "Unkown child type!"
    end

    position = position + 1
   }
   index = index + 1
  }

  #some old curriculums where flawed in that they contained nothing,
  #this handles that cleanly; they continue to be empty.
  #Since a sequence needs a base section though, jsut hrowing in a borring, empty
one.
  if 0==old_ids_new_ids.length
    nsection = LinearSection.create! :sequence_id => seq.id
    old_ids_new_ids << nsection.id
  end

  seq.base_section = Section.find(old_ids_new_ids[0])
  seq.save!

  #print "Maps: " + old_ids_new_ids.join(",") + "\n"
end

def convert_all_sequences
  old_sequences = JavaProject::Curriculum.find :all#, :conditions=>{ :id => 1 }

  print "There are "+old_sequences.length.to_s+" curriculums to convert.\n"

  old_sequences.each do |old_sequence|
   convert_sequence(old_sequence)
  end
```

```
end

begin
 convert_all_sequences
rescue Exception => e
  puts "Exception: #{e.class}: #{e.message}\n\t#{e.backtrace.join("\n\t")}"
  exit 1
end
```