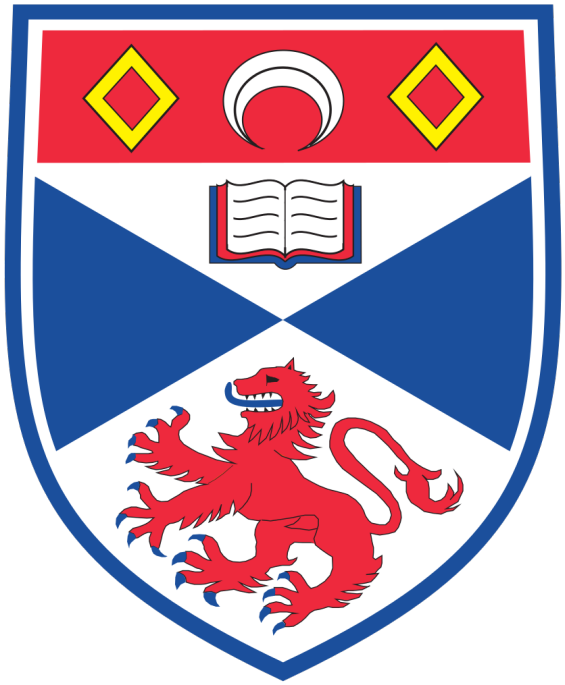
Tracker: Adaptable Curriculum Analysis



University of St Andrews
April 4th, 2016

Ryan Hamilton
120008008

Supervisor: Graham Kirby

Abstract

University degree curricula are often built with modularity and flexibility in mind. While this offers students a greater freedom when embarking upon their chosen paths, it presents a challenge for the school in terms of information management. It is imperative that a school maintains a consistent understanding of the skills it delivers and the capabilities of its students, but a wide array of module options often results in a school with an incredibly diverse skillset.

Tracker is a system that takes in user-provided data regarding skill associations and student module selections and presents schools with an extensive collection of information and visualisations which are dynamically updated with each change to the underlying database. The system is fully adaptable; very few data attributes are pre-set, allowing the tool to be configured for use by any school that offers module choices.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 9574 words long, including project specification and plan but excluding appendices.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Table of Contents

1. Introduction

One of the chief aims of a university school is to provide the strongest possible education for each of its students, but this rarely equates to providing identical educations to each individual. Offering a diverse set of module choices allows students to explore multiple diverging paths before specializing, thus giving schools the opportunity to produce experienced graduates who are familiar with the broad spectrum of their chosen field.

There is a key distinction to be made between data and information. Data is raw and unorganized, lacking in context and focus. Information is the result of taking this mere collection of facts and molding it into something that provides meaningful insight.

Academic institutions thrive on this kind of information. They have to make frequent decisions about how best to guide students from their first day to graduation, a feat made even more challenging by a field – like Computer Science – which evolves so rapidly every year. Decisions like introducing new modules have to be made with consideration given to a number of factors: student capability, student interest, relevance in the current field, space in the curriculum, and many more.

The motivation for this project was to take the substantial amount of data that university schools have at their disposal and turn it into valuable information that allows them to make more informed decisions about what is best for them and their students. Specifically, we wanted to focus on tracking the extent to which skills are taught by a school, and subsequently how student module choices affect the distribution of skills among graduates Through extensive refinement and evaluation, which will be discussed in detail later in this report, I believe the project was a success.

1.1. Objectives and Successes

More specifically, the formal objectives set out at the beginning of this project – as part of the Description, Objectives, Ethics and Resources (DOER) document – were as follows.

1.1.1. Primary Objectives

- Design and implement a system which manages data relevant to student skills and degree paths as a result of the modules available from the University of St Andrews Computer Science school.
- Beyond skill constraints, also allow the user to access visualisations relevant to class-wide statistics such as attendance, coursework submissions, and grades.

1.1.2. Secondary Objectives

- Build the system to be dynamically adaptable for use by any university Computer Science school. This will be done by allowing the user to create their own degree curricula and module entries as a basis for their system.
- Allow the system to analyse the information it is managing to detect any potential oversights, such as a student not gaining an essential skill by a particular point or a module choice which cannot pass due to prerequisites having not been met, etc. Inform the user of such oversights as soon as they are detected.
- Make the system available through the web, and accessible to the user without special software (excluding a modern web browser).
- Provide an appropriate level of security by specifying different access levels for different types of users, so as to guarantee that they can only examine the data of specific students if they are cleared to do so by an administrator.

1.1.3. Tertiary Objectives

- Extend the system so that it may be used as an information management/ visualisation tool for other subject areas, beyond Computer Science.
- Evaluate the effectiveness of the system, both in terms of functionality and aesthetics, using human subjects and surveys.
- Use the results from the survey to design and implement improvements for the system.

2. Context Survey

2.1. Introduction to Information Visualisation

It is important to distinguish between traditional visualisation and the computer-based subset of that field, commonly referred to as information visualisation. While the history of visualisation in general stretches back as far as cave paintings, the field of information visualisation, that is "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition" (Card et al., 1999) is still relatively young.

The virtue of information visualisation is not that computers are more capable of exploring raw data than we are, but that the most effective explorations arise from instances of humans and machines functioning symbiotically. Munzer (2014) states that information visualisation was born out of a "need to augment human capabilities, rather than replacing people with computational decision-making methods". This becomes evident as one begins to explore the potential design space for visualising even the most straightforward possible datasets.

Ortiz (2012) illustrates this point by visualising only two different quantities, 75 and 37, using as many unique techniques as possible. Figure 2.1 shows twelve of the forty-five techniques he employed. Ortiz' exercise perfectly encapsulates the uniquely human touch required to effectively transform raw data into the correct kind of insight. Each of his visualisations gives rise to a slightly different perspective in such a simple set of data, which underlines how truly massive the design space is for a complex dataset.

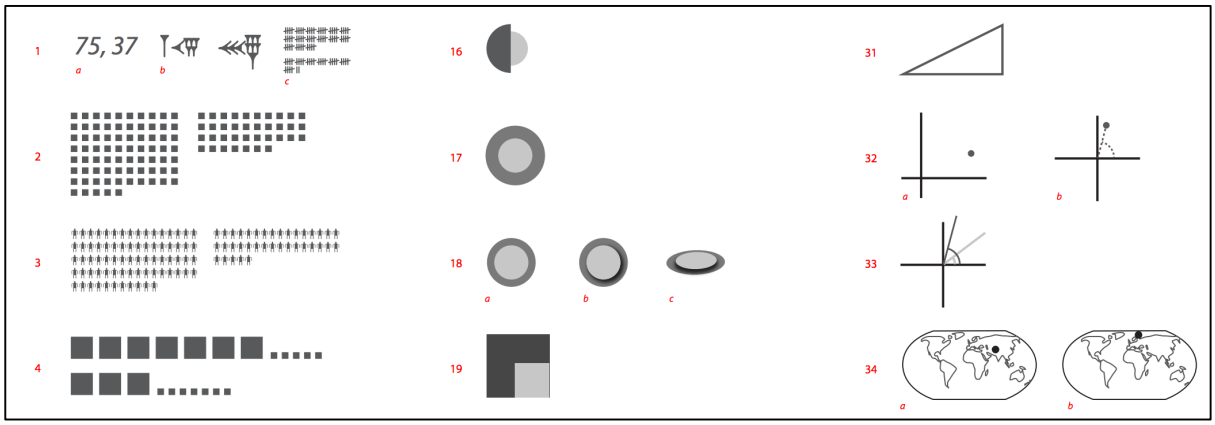


Figure 2.1
Mathematician and educator Santiago Ortiz shows just a few of the many possible ways of visualising two different numbers.

2.2. Types of Visualisation

The role that a visualisation plays can usually be categorised as either presentation or exploration. Presentation-based visualisations attempt to communicate predetermined facts about a set of data. The designer has a specific narrative in mind, and the visualisation is employed to convey that narrative to the audience in as clear, informative, and interesting fashion as possible. One of the most famous and well-regarded examples of presentation-based visualisation is Charles Joseph Minard's graphic which cartographically documents Napoleon Bonaparte's march on Moscow (Figure 2.2).

The ingenuity of Minard's visualisation lies in his ability to precisely convey three complex dimensions of the march. The routes that the soldiers take during their approach and subsequent retreat are distinguished by colour, and geographical points are marked with temperature labels to convey the challenging Russian Winter. But perhaps the most impactful insight Minard shows is the devastating loss of life as the march goes on. The number of troops at each point in time is illustrated by the thickness of the route lines, so the visualisation can show the stark contrast between the start and end of the expedition.

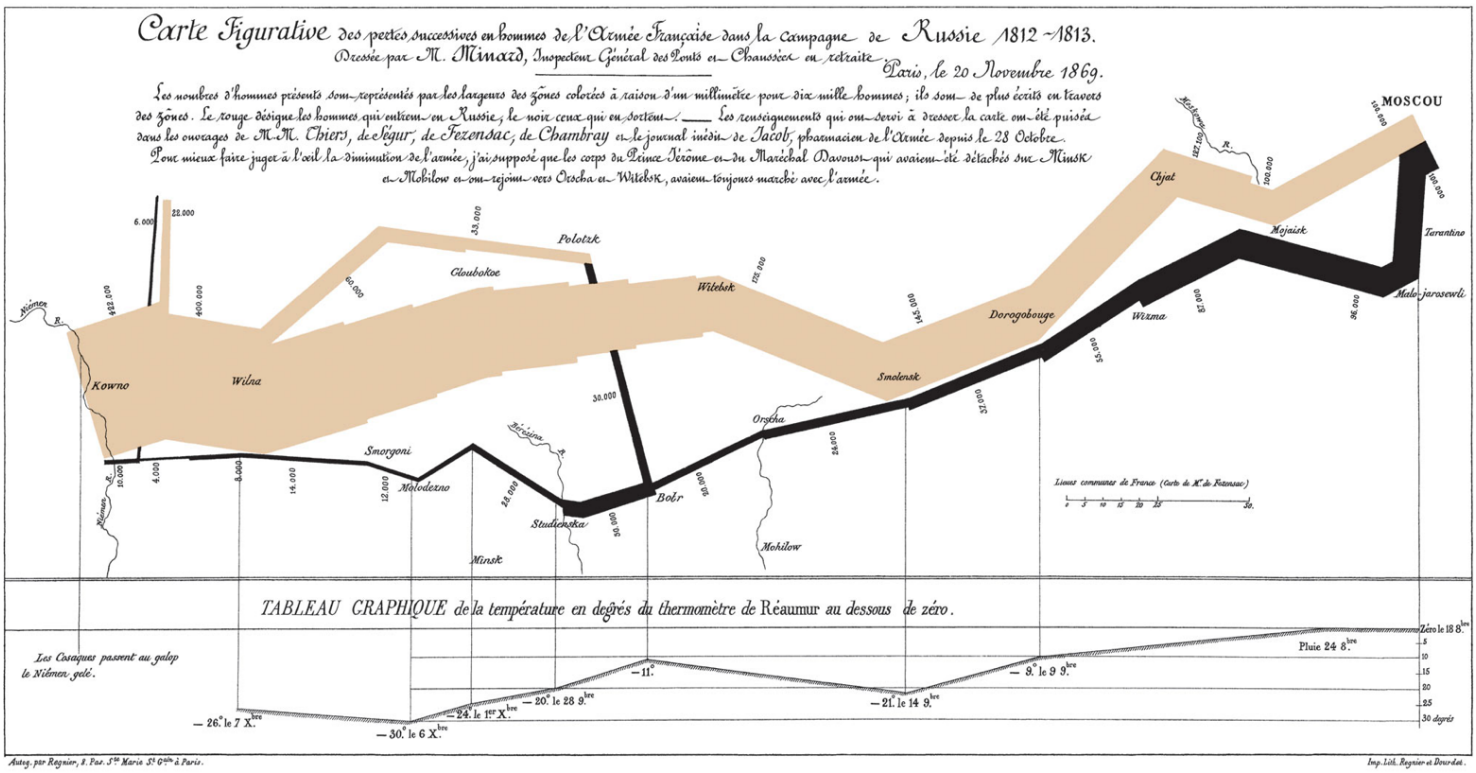


Figure 2.2
"Carte figurative des pertes successives enhommes de l'Armee Francais dans la campagne de Russe 1812-1813" by Charles Joseph Minard. His landmark visualisation effectively illustrates the futility of Napoleon's attempted invasion of Moscow (Corbett, 2001).

The second category, exploration-based visualisations, are created in an attempt to illuminate one or several aspects of a dataset that may not be immediately clear. The designer behind an

exploration-based visualisation does not have a predetermined narrative that they would like to lead the audience through, instead the purpose of these visualisations is to equip the audience with the tools necessary to explore attributes of the data that could not previously be seen. One notable example of this style is the plotting of Anscombe's Quartet, shown in Figure 2.3.

Here we can see four datasets, comprised of eleven (x, y) pairings. Each of the datasets share almost identical results (Anscombe, 1973) for the following:

- Mean of x: 9 (exact)
- Mean of y: 7.50 (to 2 decimal places)
- Sample variance of x: 11 (exact)
- Sample variance of y: 4.122 or 4.127 (to 3 decimal places)
- Correlation between x and y: 0.816 (to 3 decimal places)

Despite these precise similarities, each of these four datasets appear completely different when graphed. These visualisations are effective because they give the viewer the opportunity to explore the underlying differences in the datasets, even when those differences are hidden by the raw data alone, or by statistical similarities.

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

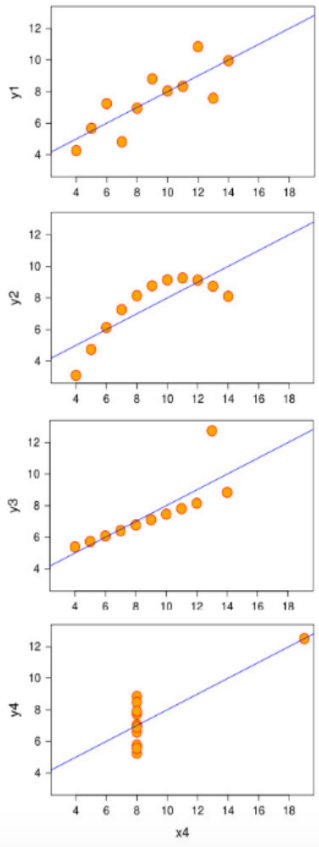


Figure 2.3
Anscombe's Quartet contrasted with their corresponding scatter plots (Hinrichs, 2016).

3. Requirements Specification

After deciding on the official objectives for this project, as described in the DOER document, a subsequent list of requirements was compiled. These requirements were more specific than the objectives and served as the jumping off point for each stage of development. The exact methodology behind the implementation is discussed at length in the next section.

The requirements list was ordered by priority so that the early development cycles would be focused on getting the system and the back-end structure up and running, before implementing additional features and improving the user interface at the later stages.

3.1. High Priority Requirements

- Requirement 1 (Non-functional)

Every feature must be accessible through the web; all major browsers must be able to use the system to its full functionality.

- Requirement 2 (Functional)

There must be a facility for users to specify their own collections of modules that they offer, skills that they would like to track, and students to analyse. For adaptability purposes, none of these records should be hard-coded; users must be able to make alterations to existing records.

- Requirement 3 (Functional)

The system shall provide skill tracking visualisations as a result of analysis performed on the user-provided data. Specifically, these visualisations must illustrate when students are most likely to learn each skill in their academic career, the proportion of students that learn each skill by the end of their four years, and the extent to which a skill has been learned (through repeated encounters).

- Requirement 4 (Non-functional)

The system must dynamically react to changes the user makes within the database. The results of user changes to data must be seen immediately in related visualisations.

3.2. Medium Priority Requirements

- Requirement 5 (Functional)

The system shall present the user with a layered view on the data; allowing access to profiles for each individual class, module, and student.

- Requirement 6 (Functional)

The system shall alert the user in the event of any detected 'oversights', which are defined as inconsistencies in student module entries such as missing compulsory modules or selected modules for which the student is not eligible.

- Requirement 7 (Functional)

For each skill that the user wishes to track, the system shall calculate the availability of this skill (how many modules teach it) and the popularity of the skill (how many students take said modules) and provide a visualisation with this information.

- Requirement 8 (Non-functional)

The process of adding and updating records shall not be time consuming. Even non-technical users shall be able to navigate through and manage records with ease.

- Requirement 9 (Functional)

The system shall offer skills tracking visualisations and analysis specific to both individual students and the entire school, for contrast and comparison.

- Requirement 10 (Functional)

The system shall keep track of historical data for how different classes have interacted with modules over the years in order for the user to be able to identify temporal trends. Additionally, this historical data shall be used to calculate the probability of a new student taking a module.

- Requirement 11 (Functional)

The system shall use historical data to calculate the proportion of the class that has gained a particular skill at the end of each academic year, in order to assist the user in planning modules around the ability levels of each year.

3.3. Low Priority Requirements

- Requirement 12 (Non-functional)

The user interface shall be presented as clear and unobtrusive, and maintain a consistent colour palette throughout.

- Requirement 13 (Functional)

The system shall provide a searching functionality for finding students and modules.

- Requirement 14 (Functional)

The system shall keep a log of recent updates to the database – including the type of change, the details of the change, and the exact time that the change was instigated.

- Requirement 15 (Functional)

The system shall provide a high-level overview visualisation of the curriculum, with branching module paths. This visualisation shall be interactive; the user should be able to expand and retract areas of the tree at will. The tree will be automatically populated using the modules that the user has entered for the skill-tracking features.

4. Software Engineering Process

4.1. Methodology

The nature of this project presented a challenge for myself and my advisor when it came to deciding on a development strategy. It was tempting to view the objectives as a list of static requirements to be fulfilled in a linear fashion, but one of the most important aspects of the project was investigating which visualisations and analysis techniques were most effective, an endeavor which lends itself to a more research-based approach.

As discussed in the Context Review, the potential design space for visual analysis is unfathomably large, particularly with a complex database structure like this with multiple associative relationships. For this reason, I decided upon an agile development strategy focused on iterative work sequences and continuous evaluation.

Each sprint was exactly one week in length, in accordance with my weekly supervisor meetings. During these meetings, I would begin by outlining everything I had achieved in the previous week. Usually, this was accompanied by a practical demonstration, which allowed my supervisor and I to discuss the new features from the perspective of a user. Following the demonstration, we would decide on the best way to progress for the next sprint. The most challenging aspect of this stage was time management. Often, in the early stages of development, I was overly ambitious, setting unrealistic goals, but over the course of this project my ability to estimate development times improved. Usually this new sprint was focused on building upon the work of the previous one by selecting the next logical tasks that remained from the requirements list, but sometimes we would decide to dedicate part of the sprint to exploring alternative solutions to what I had just demonstrated.

This strategy of iteratively developing the tool, and investigating multiple visualisations was deliberate. It allowed us to carefully refine the tool over time until we were ready for the next development stage in February/March. This was the user testing phase, for which we wanted to have a fully working system so that feedback could be collected on as many features as possible. I had spent the first five months of development iteratively implementing various alternative solutions so that I could present the subjects with the visualisations and approaches that myself and my supervisor were most confident about. The details and results of the user tests and evaluations are discussed at length in a later section.

I decided to leave some time between the user testing phase and the submission date so that I could collect the results, discuss them with my supervisor, and subsequently spend the last remaining development sprints improving the system based on the suggestions and feedback given by the subjects.

4.2. Tools & Technologies

4.2.1. Ruby on Rails

Ruby is a dynamic, general purpose programming language. It is object-oriented in the 'complete' sense as every value is treated as an object. Ruby is often lauded for its clean syntactic features and extensive library support. It is derivative of Perl, Lisp, and Smalltalk, but makes use of a grammar style that allows easy transition from C or Java (Flanagan and Matsumoto, 2008).

Rails is an open-source web development framework written by and for the Ruby programming language. Ruby on Rails was my main platform for the back-end development of Tracker. It was used to define the structure of the database and the relationships between data models.

The virtue of Rails is that it takes advantage of the flexibility of Ruby to essentially create a domain-specific language for writing web applications. In addition to assisting the developer with the management of front and back-end software, Rails also makes use of 'Gems' which are self-contained pieces of code designed to handle specific problems such as CSV upload or third party compatibility (Hartl, 2015).

4.2.2. JavaScript

For handling front-end events, I used the jQuery JavaScript library which is a collection of functions designed to manage user interface requests and data manipulation across html pages. Outside of aesthetic features, my main use of jQuery was for handling the user-defined input in the search bars for students, modules, and skills.

Another JavaScript library that Tracker makes use of is HighCharts. This is an extensive visualisation library focused on interactivity and cross-platform support. It was through a combination of HighCharts and SQLite data queries that the bar and line chart visualisations were formed.

For one specific visualisation – the curriculum overview – I used D3, a JavaScript library built to create data-driven visualisations. I sought out a more configurable visualisation library than HighCharts for the curriculum overview because I had a very specific vision for how it should look – with branching module paths and expandable nodes – that D3 supported more effectively.

4.2.3. SQLite

SQLite is a relational database management system that is embedded into the end program. I used SQLite with the Rails-powered back-end to create queries on the user-defined data. It forms the basis of all the data-driven algorithms used for analysis in Tracker. I chose SQLite because it is well-supported by the Rails framework. Rails allows SQLite queries to be written in Ruby code and converted for data retrieval. This was a useful approach to take because it allowed for better language consistency throughout the application.

4.2.4. Bootstrap

Bootstrap is an open-source front-end frame work that uses JavaScript and CSS libraries to streamline many user interface features. In Tracker, Bootstrap was used to assist with the visual structure of interface features such as the navigational bar and data panels.

4.2.5. Git

Git is a command line tool for source code management. I used Git for version control during the development of Tracker because of the many database-centric features. Often testing such features demands that major changes to the database take place, so Git was used to maintain a stable build of the system in the case of any accidental data loss incidents. In addition to Git, GitHub – a web interface for Git – was used to keep a visual log of development for quick reference.

5. Ethics

All data used during development and testing was fictional, either the product of random variable generation or imagination, no student data was collected by the system. Therefore, there are no ethical considerations with regards to the gathering of personal data. At the beginning of this project, the Preliminary Ethics Form was submitted stating that human subjects would be used for the user testing phase.

All information gathered during the user testing phase was stored in a secure and anonymous form. In the case of electronic storage, the files were kept in an encrypted location. Each participant was given a standard ethical briefing and signed a consent form.

6. Design

The main idea behind the visual structure of Tracker was giving the user 'levelled perspectives' on the school. By this, I mean that I wanted the user to be able to take as specific or as general a view on the data as they desired. For example, the user may want to explore the skill distribution of the entire school, which would be the most general perspective on that data. But after that, they may want to narrow their view to just an individual class, then an individual module, right down to the most specific perspective – an individual student.

This idea of perspective levels is evident in the final design, as seen in Figure 6.1. The home page of Tracker immediately presents the user with a quantitative overview of the school, and from that point they have the option to get as specific as they like. Every class, module, student, and skill has a corresponding page allocated to them as soon as they are entered into the database. For example, each class page contains information about which modules are currently being taken by this class, the proportion of the class taking each module, and the skill distribution of this class given all their previous module choices. Profile pages for each individual record in the database allow the user to select exactly the perspective they would like on the data.

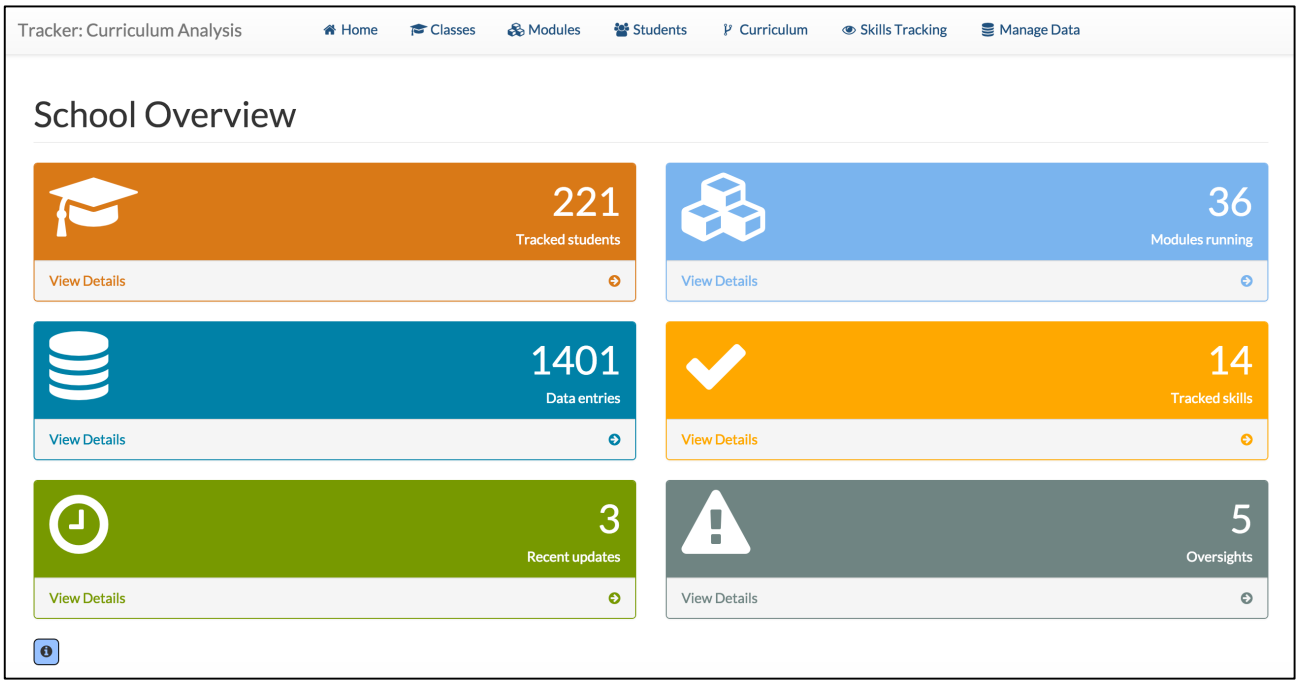
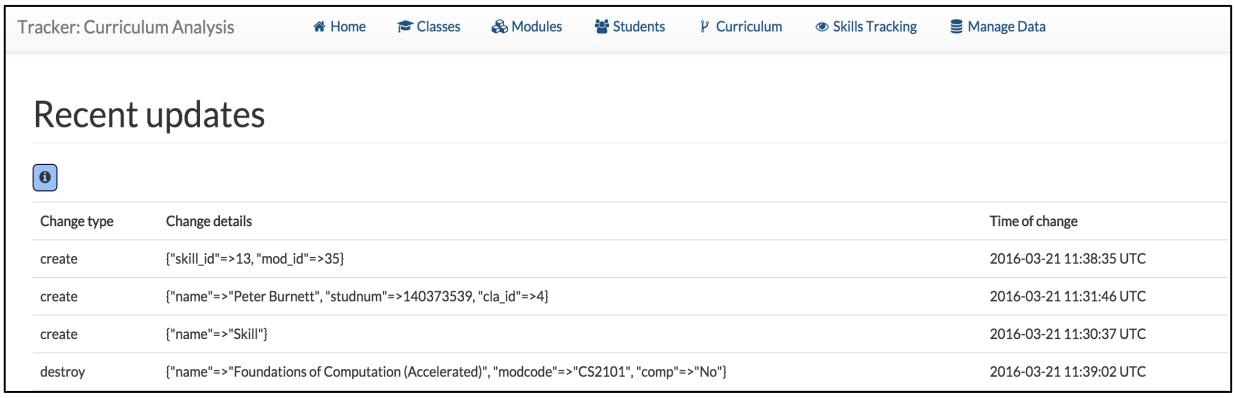


Figure 6.1
The home page for Tracker, which provides the user with a snapshot of the school from a quantitative point of view and the option to further explore the attributes.

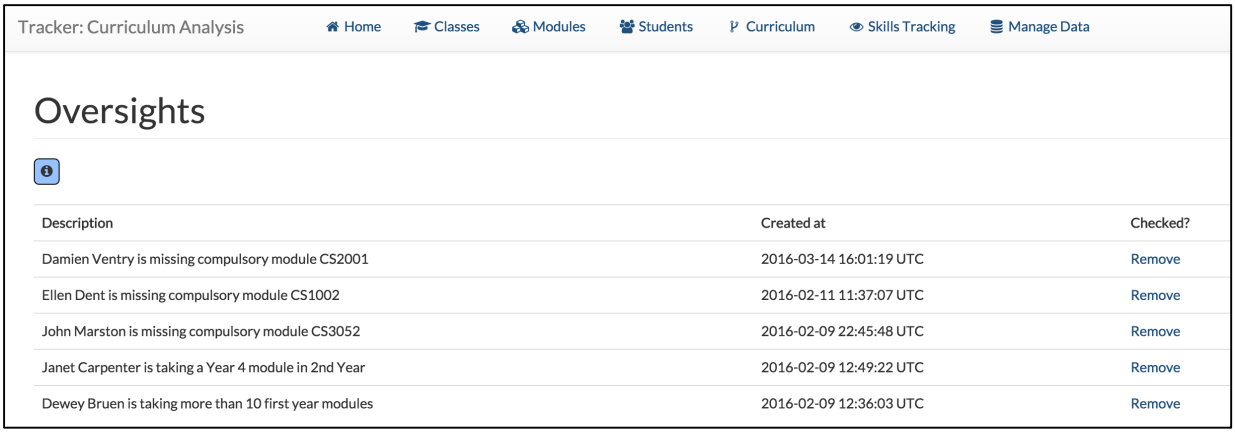
6.1. Homepage/School Overview

The homepage is the first piece of content the user sees when they access Tracker. For this reason, it was built to function as a snapshot of the school in its current state. It not only informs the user of the quantities of active students/modules/skills that are currently being tracked, but also the number of recent updates and oversights detected within the database.



Change type	Change details	Time of change
create	{"skill_id"=>13, "mod_id"=>35}	2016-03-21 11:38:35 UTC
create	{"name"=>"Peter Burnett", "studnum"=>140373539, "cla_id"=>4}	2016-03-21 11:31:46 UTC
create	{"name"=>"Skill"}	2016-03-21 11:30:37 UTC
destroy	{"name"=>"Foundations of Computation (Accelerated)", "modcode"=>"CS2101", "comp"=>"No"}	2016-03-21 11:39:02 UTC

Here is an example of what the audit log might look like to the user. The first column refers to the type of change, whether it be create, destroy, or edit. The second column shows the details of the records that were involved in the change. For example, we can see in the second row that a student named 'Peter Burnett' was created with a student ID of '140373539' and belongs in the class with ID '4'. The third column refers to the time at which the change was initiated. Twenty-four hours after the time of initiation, the change will be removed from the audit log.

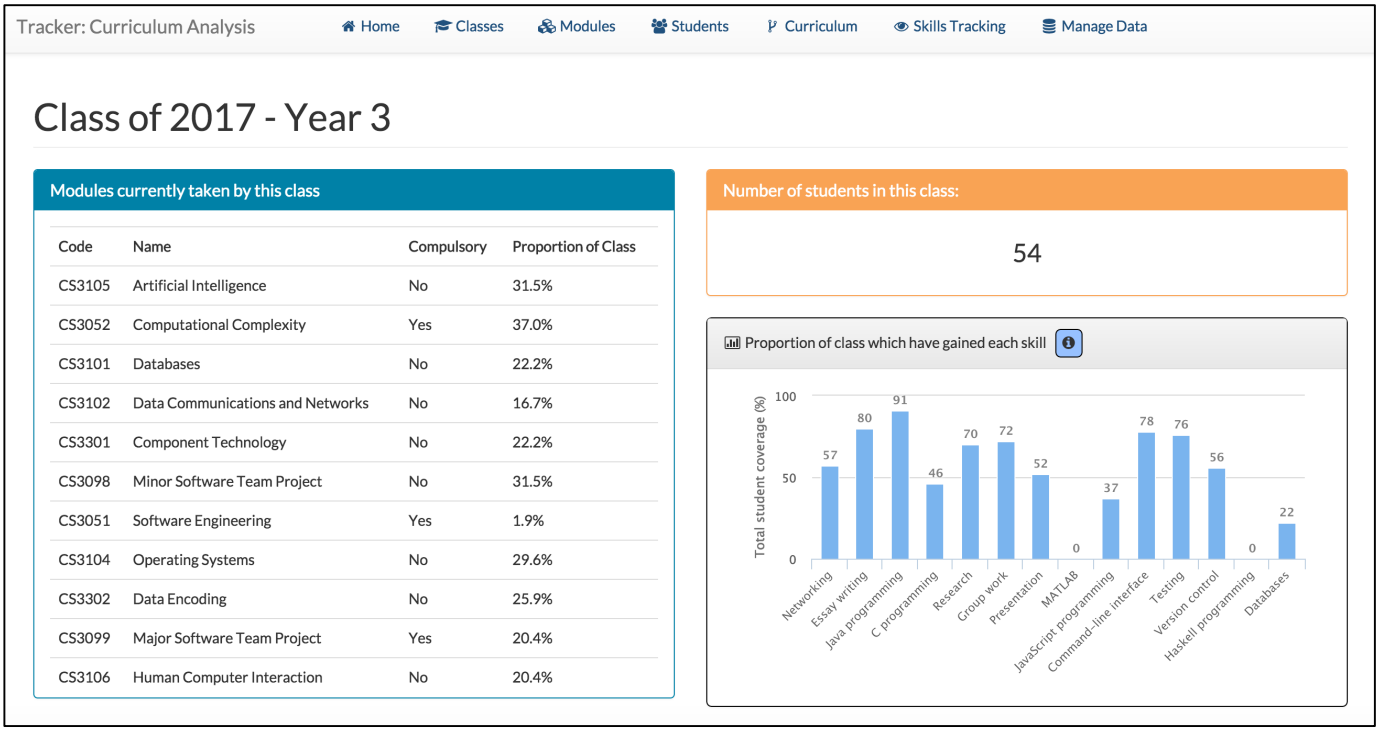


Description	Created at	Checked?
Damien Ventry is missing compulsory module CS2001	2016-03-14 16:01:19 UTC	Remove
Ellen Dent is missing compulsory module CS1002	2016-02-11 11:37:07 UTC	Remove
John Marston is missing compulsory module CS3052	2016-02-09 22:45:48 UTC	Remove
Janet Carpenter is taking a Year 4 module in 2nd Year	2016-02-09 12:49:22 UTC	Remove
Dewey Bruen is taking more than 10 first year modules	2016-02-09 12:36:03 UTC	Remove

Here is an example of what the oversight detection log might look like to the user. Unlike the audit log, the entries here are not cleared after a day, but are checked off manually by the user. There are two reasons for this. Firstly, there may be hundreds of entries in the audit log if the user made a lot of changes, which would be time consuming to remove manually. And second, oversights are more important, and may need to be addressed by the user, regardless of their age. The first column here conveys all the details about the detected oversight; both the name of the student and the type of oversight that has triggered on their record.

6.2. Classes

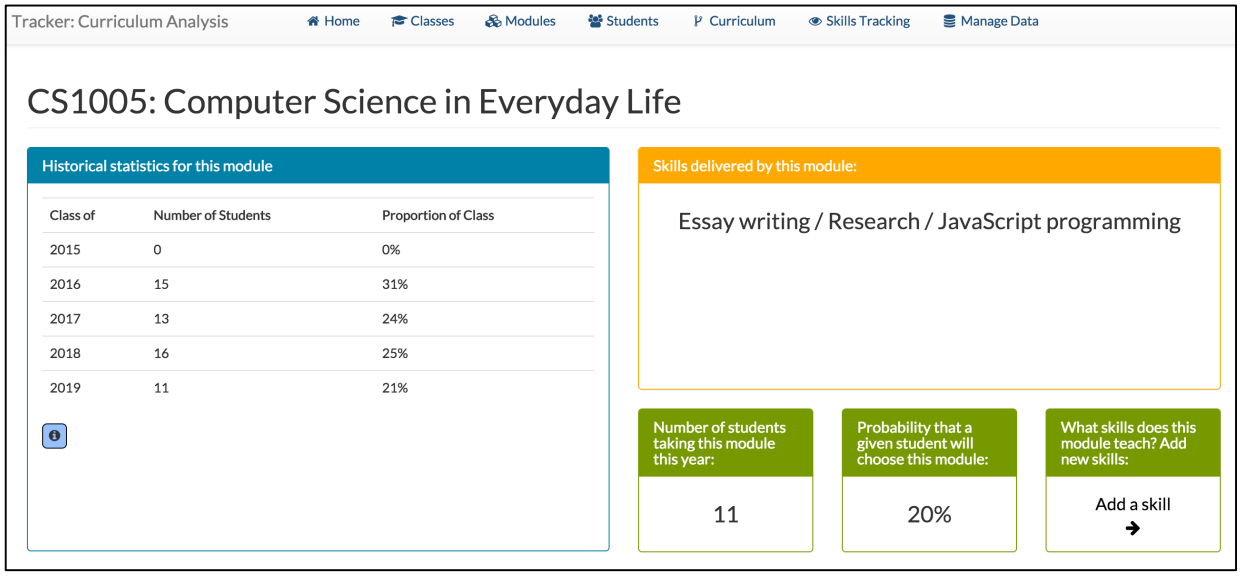
This is the class overview page. It lists the four currently active classes in the school. For clarification, Tracker defines classes by their graduation date. So the 'Class of 2018' would be 2nd years at the beginning of the 2015/16 academic year. Tracker also automatically updates its records at the end of each academic year. So the current 1st year class is moved up to 2nd year, the 2nd years are moved up to 3rd year, 3rd to 4th year, and the 4th years are no longer marked as an 'active class'. Their spot in the class overview is automatically filled by the incoming 1st year class. In the event of 2nd year students not getting into Honours, the user can manually edit their year in the data management section.



After clicking one of the panels for a class, the user is directed to the profile page for that class. An example for this can be seen above with the Class of 2017. On this page is three main pieces of information. First is simply the number of active students currently in this class, as shown by the panel in the top right. The table on left shows a list of the modules currently being taken by the students in this class. As long as one student in this class takes a module, it will appear on this table. The far right column shows what proportion of the class is taking that specific module.

The visualisation on the bottom right presents the skill distribution for the students in this class. This is a breakdown of what percentage of the class has gained each skill that the user wants tracked so far in their academic career.

6.3. Modules

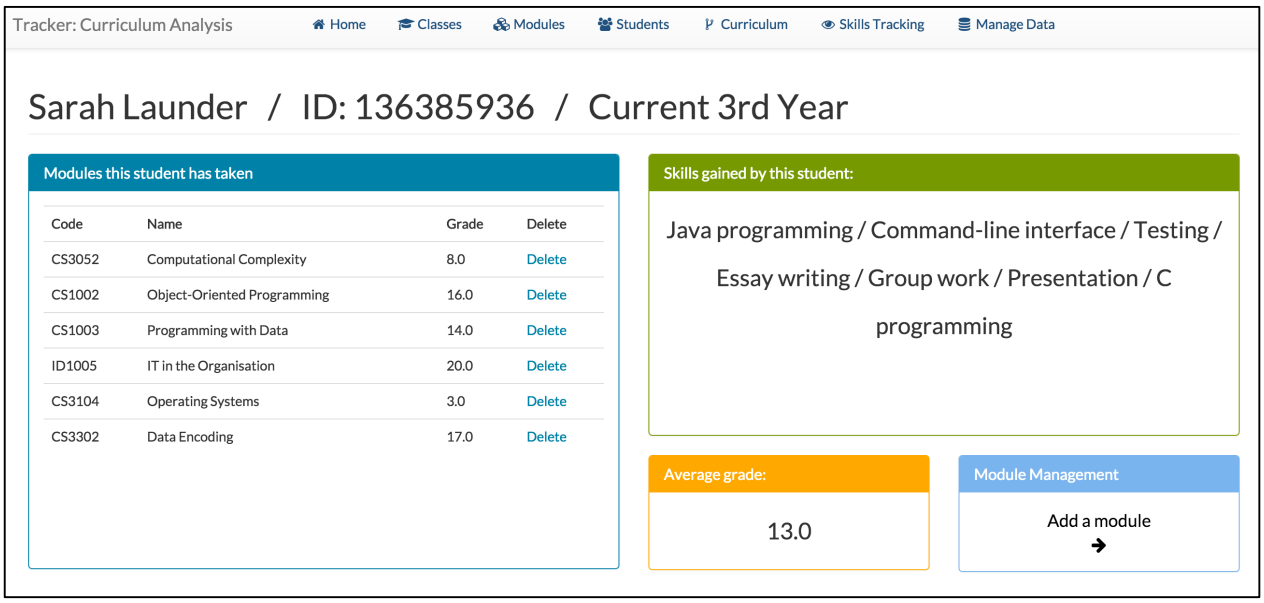


Similarly to classes, each module created by the user has a profile page, except with a different focus. The table on the left outlines the historical data for the popularity of the module. Each row refers to a different class. For example, we can see that in the academic year where the class of 2017 was in 1st year, thirteen students chose to pick CS1005, which equated to 24% of that class at the time. At the bottom right corner, the user has the option to add to the list of skills that this module teaches. We can see from the panel at the top right that it is currently associated with three skills. Also shown on this page is the probability that a given student will select this module if given the opportunity, based on previous historical popularity.

6.4. Students

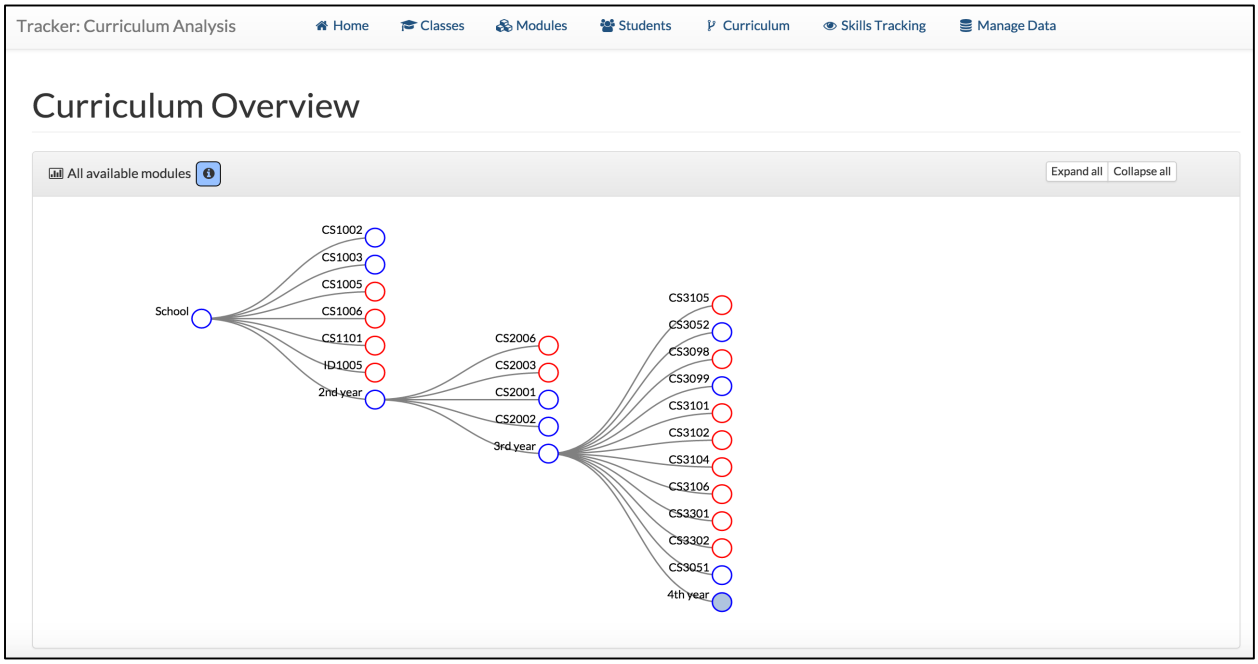
An example profile page for students can be seen below. The table on the left outlines all the modules that the student has taken along with the grade they achieved. The average of these scores is tracked by the panel in the bottom right. The panel at the top right is a dynamic list

of all the skills that the student possesses, as a result of their module choices. As soon as a module is added to this student's record (done using the button at the bottom right) or removed, this skills list will be automatically updated.



6.5. Curriculum Overview

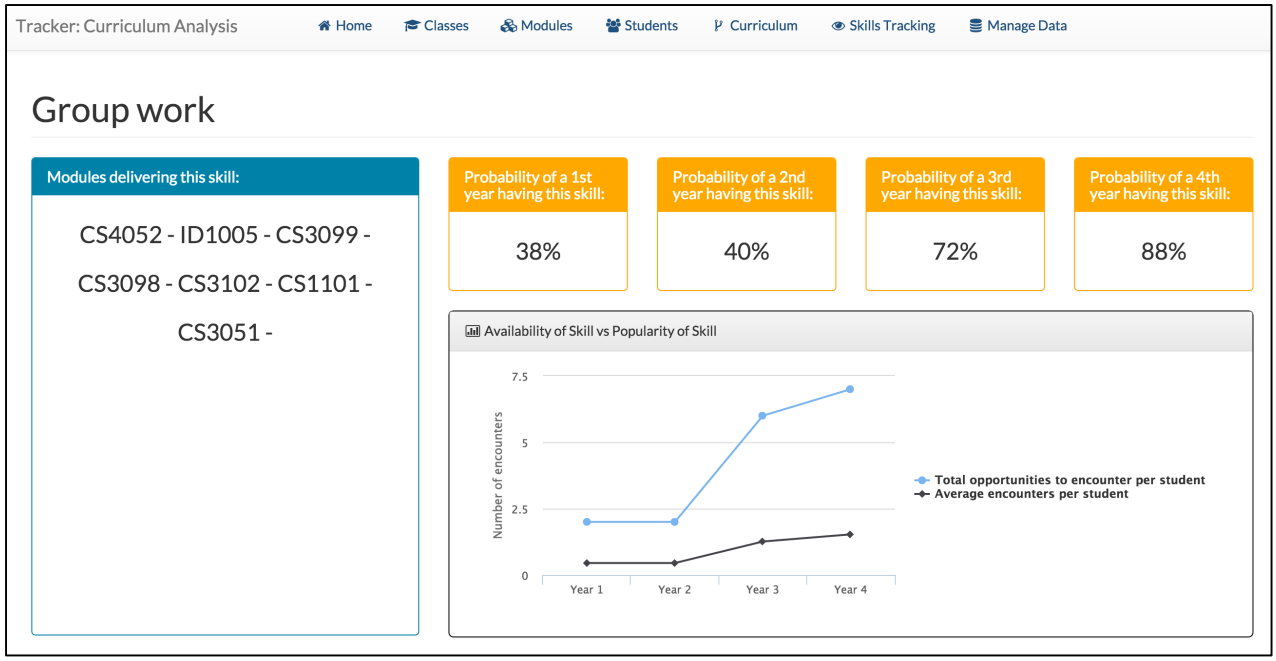
This section is a visual overview of the school's curriculum, as defined by the user. Once a new module is created, it is automatically added to the curriculum. The blue nodes shown below represent compulsory modules, and the red nodes represent optional ones. Filled nodes, such as the one labelled '4th year' can be expanded to reveal more. Potential future evolution of the curriculum overview is discussed later in the report.



6.6. Skills

Below is an example of the profile page created for each skill. The list on the left shows the modules which teach the skill, automatically updated as that relationship is added. The panels along the top right show the result of analysing the historical module choices for 1st, 2nd, 3rd, and 4th years to determine the probability that each would hold this skill.

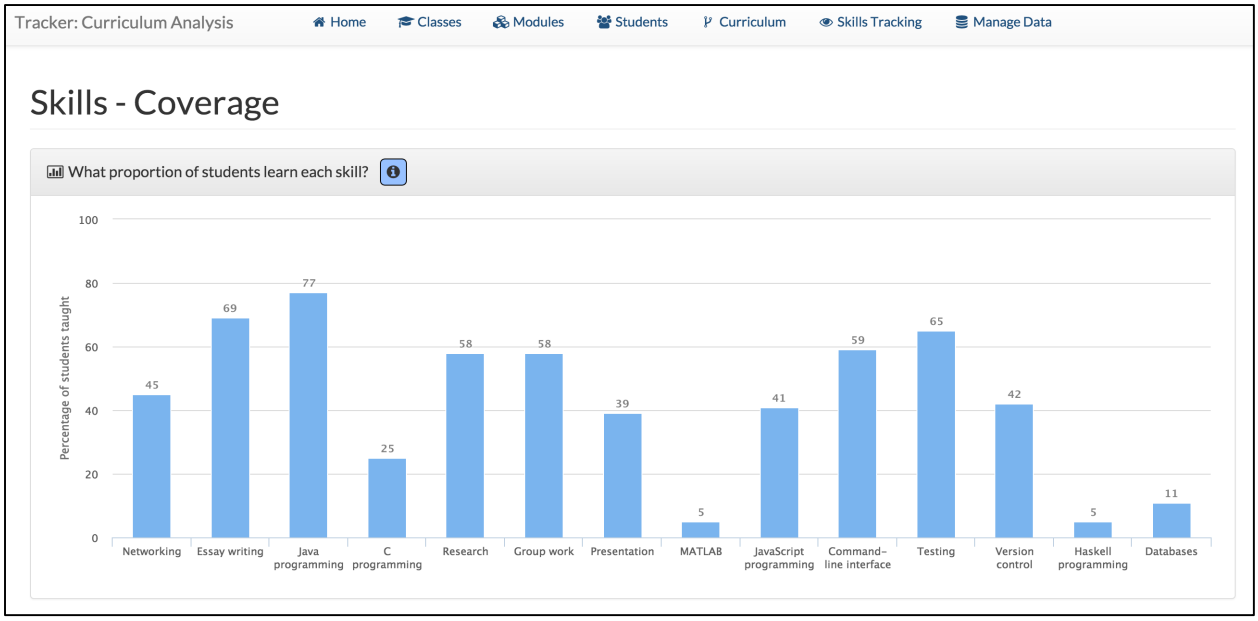
The line graph at the bottom right charts the total opportunities for students to encounter this skill at each year against the actual popularity of the skill at these times. This is something that my supervisor and I were very interested in capturing. It may be the case that although it appears that the school is providing many opportunities for students to learn a particular skill, these modules may not actually be as popular as others in the curriculum, resulting in an inflated expectation for the coverage of this skill.



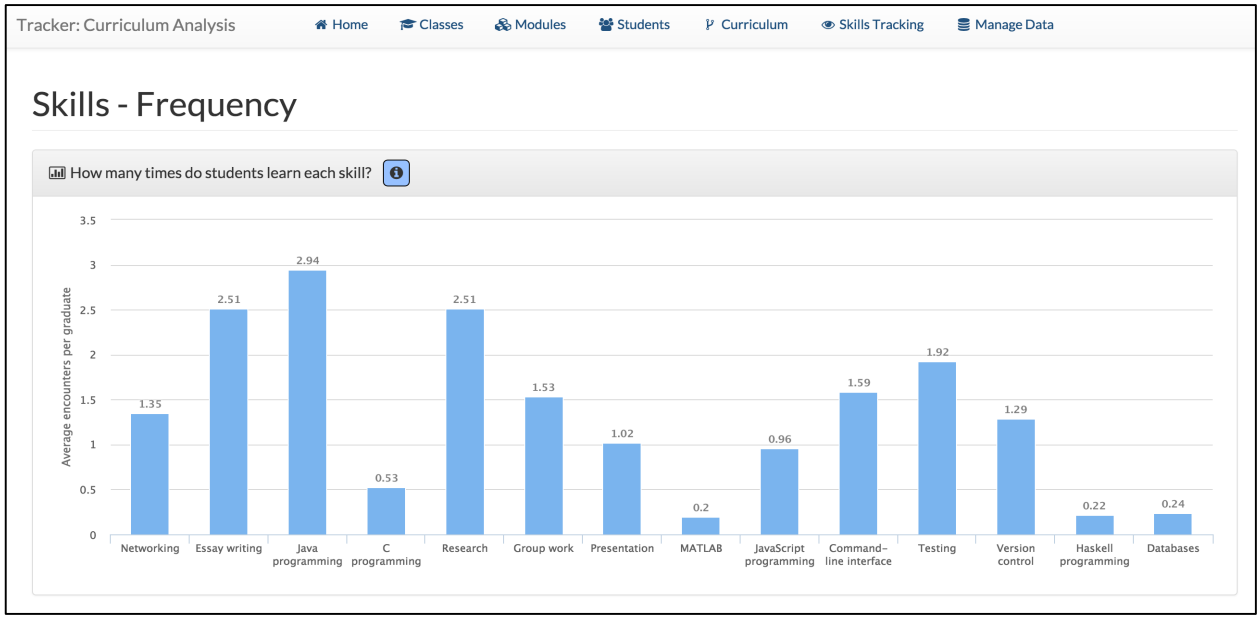
6.7. Skill Tracking

The skill tracking facility focuses on analyzing data for the entire school, and is split up between four perspectives on this data: timing, coverage, frequency, and projection.

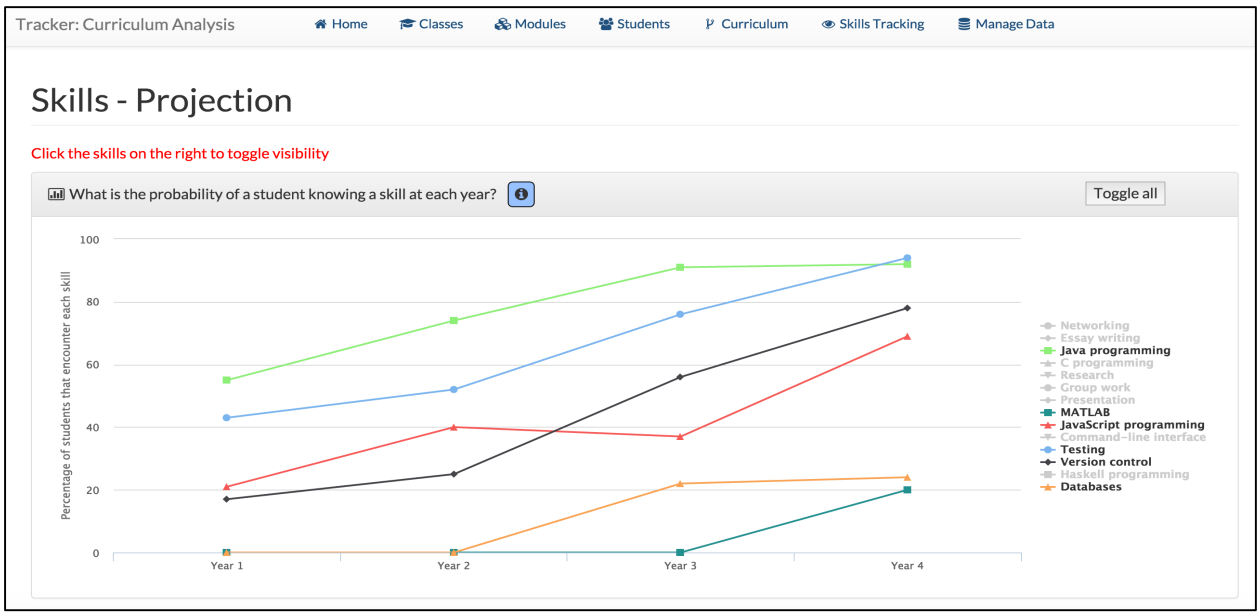
The timing visualisation (below) shows the number of times students in the school have encountered each of the skills listed along the x-axis. The colour of the bars indicate the year they were in when the encounter took place. The power of this kind of analysis is that it reveals the points in the curriculum where specific skills are taught most, allowing the school to plan new modules or changes with this information in mind. The x-axis of these graphs are automatically updated when the user enters a new skill to be tracked in the database.



The coverage graph (above) shows the percentage of students which have encountered each skill at least once. In practice, the bar for a skill such as 'Java' would be at 100% since it is a core part of many compulsory modules, but the dummy data used to populate the tables for these graphs was distributed randomly among all the modules. Compare this to the frequency graph (below) which reveals the number of encounters that the average student has with each skill over the course of their four years. This is used for understanding both the extent to which a certain skill is taught across the curriculum and the popularity of the modules teaching it.



Finally, the projection tool (shown below with some of the skills' visibilities toggled for clarity) calculates the likelihood that a student will posses a certain skill by the end of each academic year. Hovering over each data point shows the name of the skill and its percentage probability. For example, we can see version control rising steadily as the years go on, in contrast to MATLAB, which isn't offered until fourth year.



7. Implementation

The structure of the final implementation of the database underneath Tracker can be observed in Figure 7.1. Each of the blue boxes represents an entity and the attributes that define it. Underlined attributes indicate a primary key which is the identifier for that entity. For example, 'studID' is listed as the primary key of 'Student' because each student has a unique ID number. Attributes marked with 'FK' are foreign keys, which are the primary keys of entities that share a relationship. For example, 'gradDate' is listed as a foreign key for students because of the relationship between 'Student' and 'Class'.

Relationships are represented by the lines that connect entities. Cardinality constraints are also expressed by these connections. An arrow indicates 'one' where an undirected line indicates 'many'. For example, if we examine the student-class relationship, we can observe that it is a 'one-to-many' association since a class is comprised of many students but a student can only belong to a single class.

The reason this particular structure was chosen is based around the types of queries that the database must support. For example, there are many cases where Tracker needs to retrieve the skills that a particular student possesses. Therefore, there must be a particular path of connections linking students to skills – via modules in this case. It is important that the path linking two such entities remains as short as possible to keep queries simple and maintainable.

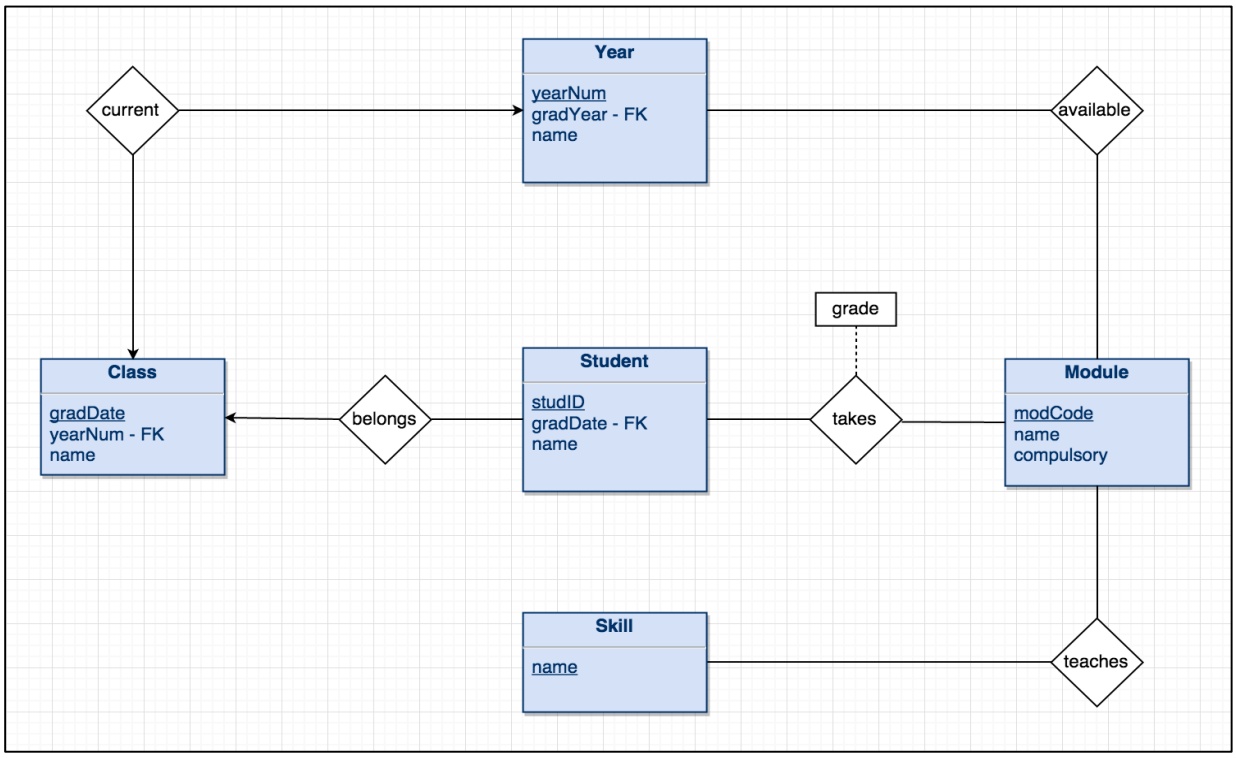


Figure 7.1
Schema for Tracker's final database structure, showing the various relationships between its entities and their attributes.

Figure 7.1 shows the final version of the back-end structure of Tracker. However, this was the product of a gradual process, with many stages of refinement and changes. The following subsections outline the development process and the decisions that led to the final system.

7.1. Stage 1

The project began with a series of meetings between myself and my supervisor establishing the objectives and subsequently the requirements that the system should meet. These are discussed in detail in earlier sections. Before development began, I started drawing up an early draft of the planned database schema, which can be seen in Figure 7.2. There are notable differences between this diagram and the final version shown in Figure 7.1.

Firstly, in the early draft, there are many more attributes contained within entities 'Student' and 'Module' than the later version. This is due to the gradual refinement of requirements that took place during the first few sprints as a result of discussions with my supervisor. For example, attributes such as 'attendance_issues' and 'coursework_issues' are representative of features that were decided to be less relevant to the aims of the project than others.

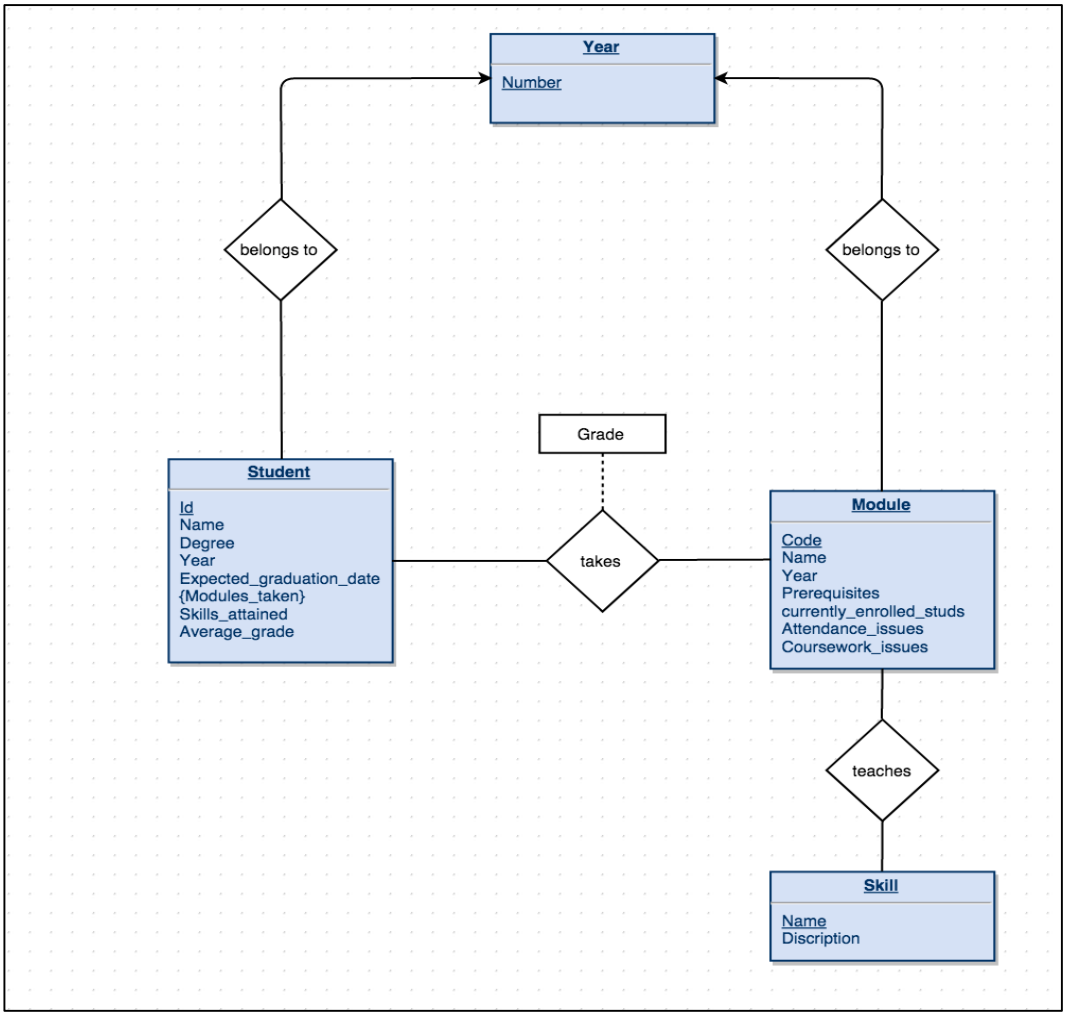


Figure 7.2
First draft of Tracker's database structure, prior to refinement.

Much more notable a change however can be spotted in the exclusion of the 'Class' entity seen in the final version. This is the result of a larger structural shift that came about following a better understanding of the core objectives of the project. For example, there are many features present in Tracker, such as the the automatic class-year updates at the end of every academic year and historical data for modules that require an entity relationship between classes and years built into the database structure.

After gaining a better understanding of the project and refining the requirements for Tracker, development officially began with building the skeleton of the user interface. As previously discussed in the design section, the user interface was very much planned around the idea of 'levelled perspectives' which was evident even in the early iterations of the menus and navigation bar. The core of the user interface was built in the first few sprints with assistance from tools such as Bootstrap and jQuery.

7.2. Stage 2

The next major step to take was exploring how the actual visualisations would fit into the existing user interface. I first wanted to create hollow visualisations that were hardcoded with dummy data before working on database integration. I felt that this approach would help compartmentalize the tasks so I could focus solely on how the visualisations would be perceived and interacted with by the user before writing the algorithms/queries for presenting data.

As previously discussed, much of this stage was spent refining the visualisations and deciding how best to convey the information we wanted. However, another important aspect of this stage was exploring the many visualisation tools available to use. For the line and bar charts present in many of the skill focused pages I looked into Flot, a JavaScript plotting tool. Flot appealed to me since it was built to integrate with jQuery, which was already widely present in Tracker from the first development stage. However, I opted not to use Flot due to its lack of compatibility with the Rails framework – which was a major strength of Highcharts. I ended up deciding to use Highcharts for the line and bar charts due to the presence of a dedicated Rails 'gem' specifically created for integrating the tool in Rails applications.

One of the most challenging and time-consuming parts of this development stage was designing and implementing the page structure and navigation. Due to the large number of perspectives that had to be shown by the tool (students, classes, modules, skills) and the multiple requirements that had to be fulfilled for each, it had to be clear to the user – regardless of technical knowledge – how to quickly navigate the application to find the information they are looking for. This resulted in the single-page style present in the final implementation, where each section's visualisations (students, classes, modules, skills) are clearly separated from the others and can always be viewed on a single page without scrolling or navigating further. Furthermore, each section is clearly accessible from the home page without the need for further exploration.

7.3. Stage 3

This was the longest stage of development; after the front-end work was complete the goal was to build a database that the user could interact with. The schema shown in Figure 7.1 was the basis for this work. During these sprints, much of the development time was actually spent researching and learning more about the Rails framework. I had experience with it in the past, but I had never built anything of this scale in terms of back-end complexity.

With Rails, back-end data passing is based around controllers and models. As a general rule, each entity in your schema should have a corresponding controller. This is where you define all the methods/functions related to that entity. For example, the 'Student' controller has a method named 'index' which defines how the list of all students is generated in the data management section. Models are where the relationships between entities are established. This makes it easier to query data since the application knows how the dependencies are defined. For example, the 'Module' model includes the lines shown in Figure 7.3.

```
has_many :studmods, :dependent => :destroy
has_many :students, through: :studmods

validates :modcode, presence: true, uniqueness: true
```

Figure 7.3
Code snippet from 'app/models/mod.rb'

The first line tells the application that each module has many 'studmods' which is an instance of the student-module relationship. The first line also indicates that, in the event that this module is deleted, its associated studmods should also be deleted. This is essential as instances of student-module associations cannot exist without the relevant module, so various queries would result in errors. The second line tells the application that each module can have multiple associated students, and that they are associated through the aforementioned studmods entity. The third line is in reference to the module code attribute that each module carries. This line ensures that no module exists without an identifying code and that it must be unique – no two modules should have the same code.

After building the structure of the back-end so that the user could save their own data, the next step was to link this data to the existing visualisations, which were running on hard-coded values at this point. Many of these queries, such as the ones used for the school-wide skill tracking visualisations are very long, so I will demonstrate using a segment from the projection algorithm, shown in Figure 7.4. This snippet shows embedded ruby code (the 'erb' at the end of the file name) which allows developers to add back-end data manipulation code in front-end html files.

```
<% Skill.all.each do |sk| %>
{
name: '<%= sk.name %>',
data: [
     <% studCount = 0 %>
     <% Year.where(num: 1).take.cla.students.each do |st| %>
          <% skillArray = Array.new %>
          <% st.skills.each do |msk| %>
               <% skillArray.push(msk.name) %>
          <% end %>
          <% skillArray = skillArray.uniq %>
          <% if skillArray.include?(sk.name) %>
               <% studCount = studCount + 1 %>
          <% end %>
     <% end %>
     <% skillProp = studCount / Year.where(num:1).take. cla.students
     .count.to_f %>
     <%= skillProp.round(2)*100 %>,
```

Figure 7.4
Code snippet from 'app/views/welcome/skillprojection.html.erb'

The purpose of the entire projection algorithm is to calculate the odds of a student having a particular skill at the end of each year. The segment in Figure 7.4 shows the code for calculating the odds of a student knowing each skill by the end of first year. The first line begins iterating through each of the skills as defined by the 'Skill' model. The next two lines are for the Highcharts tool, it defines the name of each line and the data that visualises it. We begin by finding the class that is currently assigned to the year with 'num: 1' which is the first years and iterating through all those students. For each student, we create an array of unique skills and check to see if the current skill we are looking for is present. If it is, we add it to the count of students who possess that skill in the first year class. Finally, we calculate the probability of a first year having this skill by dividing the historical number of first years with that skill by the number of first years examined and add that figure as a point on the line graph.

7.4. Stage 4

Following the previous stage, Tracker had met almost all of the requirements. At this point my supervisor and I decided to begin the user testing phase. The findings from these evaluations are discussed at length in a later section, but here I will outline the final implementation stage which was making improvements to the system in light of these findings.

One of the most common pieces of feedback was that it was not immediately clear what some of the more complicated visualisations were conveying, and how they reached their conclusions. To combat this issue, I wanted to implement explanations in each page without intruding on the content already there. Figure 7.5 shows the solution I came up with. Many of

the pages in Tracker include a blue information button that when pressed will create an easily dismissed pop-up box with a quick explanation of the contents.

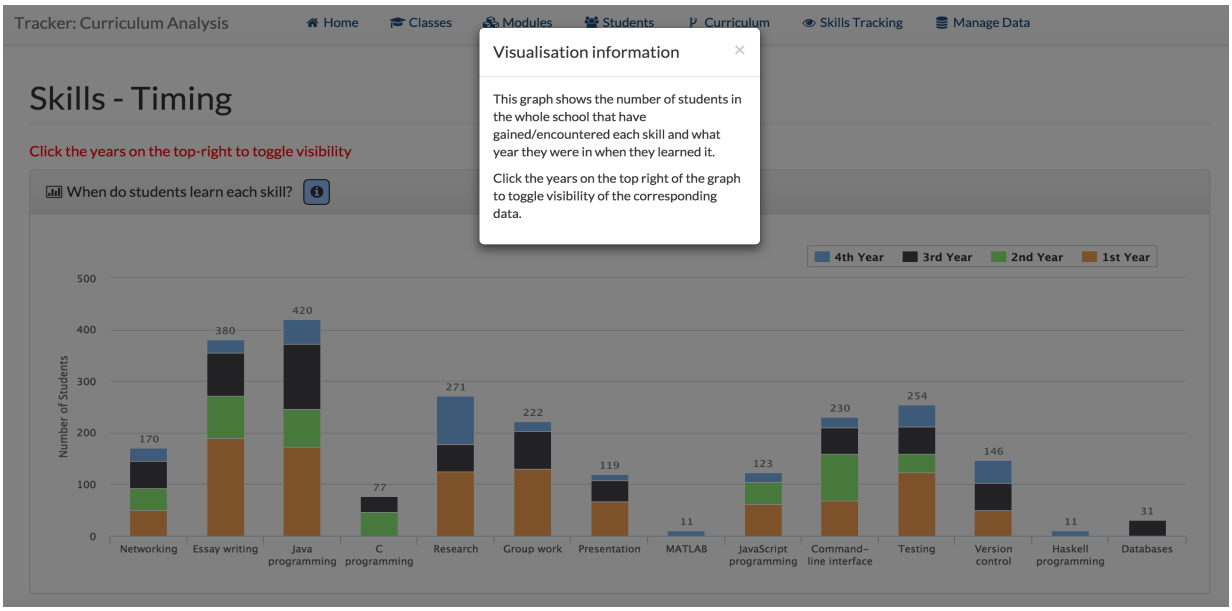


Figure 7.5
Skill timing page after the blue information button has been pressed.

One of the most important changes made in the final stage of development was the implementation of a large-scale CSV upload facility for adding students to the database. Following several discussions with my supervisor and another Director of Teaching during my user tests, I learned that schools can generate excel-style CSV files with student data. Given how time-consuming it would be to add each student individually to the database, I implemented a CSV upload facility, shown in Figure 7.6. It takes in as many student names, IDs, and graduation years as the user wants and parses out the data to create student instances in the database that match this information.

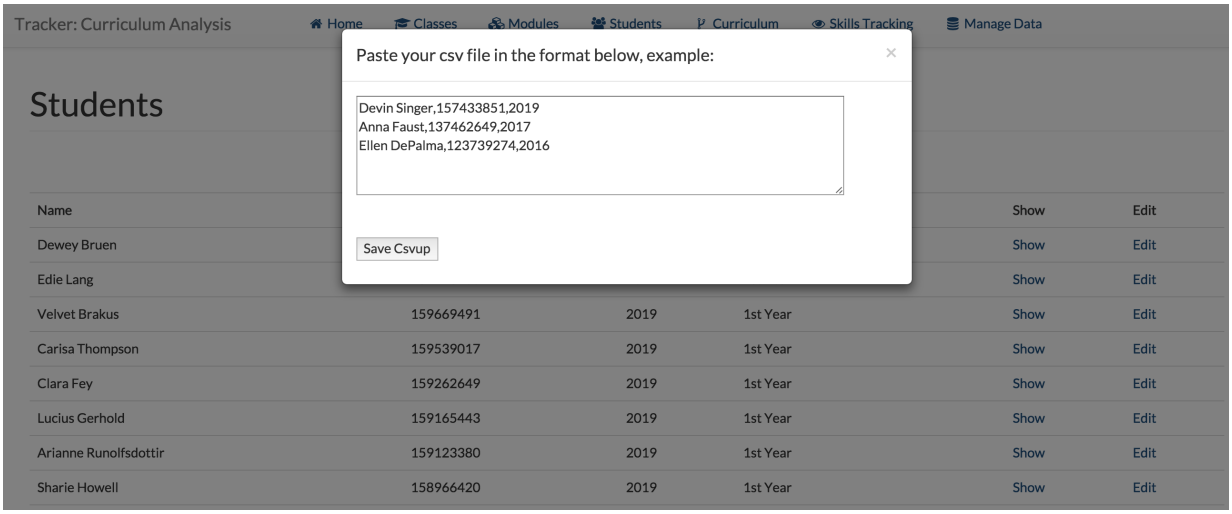


Figure 7.6
CSV upload facility for students in the data management section.

8. Evaluation

8.1. Primary Objectives

- Design and implement a system which manages data relevant to student skills and degree paths as a result of the modules available from the University of St Andrews Computer Science school.
- Beyond skill constraints, also allow the user to access visualisations relevant to class-wide statistics such as attendance, coursework submissions, and grades.

Both primary objectives have been achieved. The back-end is fully capable of managing student, skill, and module data relevant to the School of Computer Science. The structure of the database was successfully built to accommodate this requirement. Additionally, class-wide visualisations and analysis have been created to compliment the skill tracking functionality.

However, the examples above of attendance and coursework submissions were removed from the final implementation. This decision was the result of numerous discussions between myself and my supervisor, concluding that there is more meaningful insight to be gained from tracking alternative statistics such as historical data pertaining to students, modules, and classes.

8.2. Secondary Objectives

- Build the system to be dynamically adaptable for use by any university Computer Science school. This will be done by allowing the user to create their own degree curricula and module entries as a basis for their system.
- Allow the system to analyse the information it is managing to detect any potential oversights, such as a student not gaining an essential skill by a particular point or a module choice which cannot pass due to prerequisites having not been met, etc. Inform the user of such oversights as soon as they are detected.
- Make the system available through the web, and accessible to the user without special software (excluding a modern web browser).
- Provide an appropriate level of security by specifying different access levels for different types of users, so as to guarantee that they can only examine the data of specific students if they are cleared to do so by an administrator.

The adaptability of Tracker fully extends to other Computer Science schools, as no part of the design or underlying database structure is dependent on the St Andrews curriculum. The flexible nature of the tool allows other Computer Science schools to specify the skills and module associations they would like to observe.

The oversight detection facility has also been successfully implemented. The system scans each student's profile and is able to alert the user in the case of any inconsistency, such as a missing compulsory module.

The system has been tested and functions successfully on all major web browsers. However, the software is not currently being hosted on an external server, meaning that the application is accessible only by hosting it locally via a web browser and Rails.

It was decided that levels of security would be unnecessarily complex, although they could factor into future work on this system, as discussed later in this report. Since no one other than members of staff would be using the system, there is little risk of information being seen by an unauthorized party.

8.3. Tertiary Objectives

- Extend the system so that it may be used as an information management/visualisation tool for other subject areas, beyond Computer Science.
- Evaluate the effectiveness of the system, both in terms of functionality and aesthetics, using human subjects and surveys.
- Use the results from the survey to design and implement improvements for the system.

The final implementation of Tracker is adaptable for use in any subject area which offers branching module choices. This was achieved by granting the user comprehensive flexibility over the data that the tool runs on, while keeping the back-end structure consistent.

Evaluations were also successfully carried out in the form of structured interviews with various subjects of different backgrounds. The feedback gained from these evaluations proved to be incredibly valuable, leading to multiple improvements in Tracker in the final development sprints. The exact methodology of these evaluations and the changes they subsequently had upon the final system are discussed at length in the next section.

9. User Testing

Before the final stage of development began, I organized a series of separate user tests in order to learn from some other perspectives about how I might be able to improve Tracker.

9.1. Methodology

First the participant was given a briefing form, outlining the purpose of the evaluation, their role in the procedure, and information with regards to ethical concerns. After that, they were presented with a consent form, in which they would agree to take part in the study.

After receiving a short brief on the function and aims of Tracker, the participant was given the most up to date version of the system to use. They were instructed to test out all available features and to freely make changes to the database. During this phase, the participants were encouraged to take a 'think out loud' approach as all comments and questions were welcomed.

Participants were given a version of Tracker that was preloaded with around 200 students. This was done for two reasons. Firstly, it would be incredibly inconvenient and time-consuming for them to populate a realistically-sized school during a single evaluation. Secondly, it allowed them to immediately see the impact of changes they made to the database on the visualisations.

During this phase, I would observe as they used the tool and take notes. The conversational nature of the evaluation allowed for a consistent stream of feedback. Following the test, the participant and I engaged in a structured interview, in order to contextualize their thoughts and gather their feedback.

9.2. Data Collected

The following data was collected for each participant:

- A basic demographic questionnaire
- Observational notes from their use of the tool
- Notes about their comments and questions
- Answers to the structured interview questions

It should be noted that all data was kept secure. Furthermore, encrypted locations were used for the storage of all electronic data.

9.3. Findings & Analysis

Three evaluations were carried out. Participant A was a Neuroscience student. Participant B was a Computer Science student. Participant C was a Director of Teaching at the University of St Andrews. It was important to find participants both inside and outside the School of Computer Science, due to this resulting in a more diverse range of technical knowledge. However, the most valuable source of feedback was the Director of Teaching, since this is essentially the target audience for the final system.

Participant A made several suggestions regarding user interface elements. These were relatively minor like making a piece of text more obvious to the user, or tweaking the colour scheme slightly. This participant also proposed additional explanations for some of the more complicated visualisations, which led to the blue information buttons discussed previously. Another suggestion made by this participant that ended up influencing the final implementation was the addition of a 'Toggle all' button on the skill projection visualisation that allows users to change the visibility of all skill lines on the graph at once instead of individually. During the structured interview, participant A praised the usefulness of many of the visualisations but particularly the availability vs popularity graph for each skill. They also rated the tool as 4 out 5 for ease of use and suggested adapting the curriculum overview to reflect prerequisite dependencies.

Participant B was more focused on the functionality aspects of the tool. For example, one observation made that ended up impacting the final implementation was that it was initially impossible for the user to delete an instance of a student-module relationship, in the case that the grade was wrong or needed to be changed. This was overlooked during development and was promptly added following the evaluations. Interestingly, participant B made the same suggestion as participant A regarding a 'Toggle all' button for the skill projection visualisation. During the interview, participant B stated that they thought the tool would be very useful for staff due to its comprehensive coverage of the school, and also rated it 4 out of 5 for ease of use, but questioned the necessity of the skill frequency visualisation.

Participant C drew on experience as a Director of Teaching to provide insights into how the tool could be used in practice. For example, they stressed the need to speed up the process by which the system is populated with data. These comments are what led to the implementation of the CSV upload facility to quickly load the tool with as many students as the user wishes. Participant C also made several suggestions for future work to be done on a project like this, such as capturing the extent to which a module teaches a skill with 'beginner / intermediate / expert' levels of difficulty. With regards to the existing version of Tracker, during the interview participant C praised many of the skill tracking features; stating that they visualise many of the trickier concepts Directors of Teaching are concerned with. Like the others, participant C rated Tracker as 4 out of 5 for ease of use.

Overall, I found the user testing stage to be one of the most valuable additions to the project. Each participant offered thoughts that had a strong impact on the course of Tracker in its final development sprints, and helped improve my understanding of the user's perspective for the better. In particular, the insights offered by the Director of Teaching proved to be uniquely helpful in guiding my decisions and instilling a sense of confidence in my final implementation.

10. Discussion

10.1. Reflections

Designing and implementing this system with the guidance of my supervisor has been a phenomenal learning experience for me. I have never delved into a project so large or time consuming before and I sincerely believe that I have strengthened my abilities as a software developer as a result. I had minor experience with technologies such as Rails and JavaScript before, but I now feel confident in my ability to use these tools to tackle even more challenging projects in the future.

I found that the main advantage to using Ruby on Rails as the development framework for this project was the vast amount of support for Rails. A wealth of open-source code was available to me for many issues that I had throughout the implementation process. For example, the Rails gem 'Audited' (linked in the Acknowledgements section) was essential in tracking any user updates to the database tables, allowing the user to have access to the log of recent changes. The only disadvantage to using Rails was the fact that it is not compatible with many JavaScript libraries, as previously discussed with my attempts to use Flot. This often lead to additional time spent trying to make a library work or searching for alternatives.

I found the weekly sprint structure to be incredibly effective. The regular meetings with my supervisor ensured that I never stayed unproductive or started moving in a wrong direction for too long. Furthermore, regularly getting a perspective other than my own was invaluable to the continued productivity of my development process.

I am satisfied the final version of Tracker submitted alongside this report. It successfully meets nearly all of the objectives I set out to achieve almost five months ago. However, it wasn't until the user testing stage, where I got the opportunity to hear the thoughts of fellow students and even a Director of Teaching, that I became confident in the effectiveness of this tool.

In spite of these achievements, however, I feel that there is so much more I would like to do with this project, given the time.

10.2. Future Work

10.2.1. Online Access

The main hurdle to overcome is the accessibility of the application. In its current state, Tracker can be downloaded and hosted locally via the user's browser and Rails. This allows them to use every feature that the system offers, so the system is still fully available to them.

However, compared to the ideal situation, this is inconvenient. To begin with, I would like Tracker to be hosted on an external server, as this would circumvent the need to download the application to begin with. Additionally, I would like to grant users the ability to create an account on this host service, resulting in them receiving their own version of Tracker. Their version starts off with a blank database and keeps track of the changes they make. I believe this approach would be optimal for accessibility and long-term use.

To achieve this would not be overly complicated, as the base application is already built. The only change necessary would be to the back-end structure. Along with the tables discussed earlier, an additional table would be used to keep track of users. Each user is granted an instance of a 'School' entity which is associated with the others (students, modules, classes, skills).

10.2.2. Pre-requisite Dependencies

This is a feature idea that originally came up as a result of a discussion between myself and my supervisor. In essence, the flexible nature of module choices has a downside. Due to the fact that many modules have a pre-requisite requirement that all students taking it must have already passed a previous module, students regularly make decisions about their module choices in first and second year that unknowingly lock them out of taking modules in third or fourth year.

It would be incredibly useful if the school were able to see a clear visual representation of the pre-requisite dependencies among their modules. The way I would like to do this is by merging it with the existing curriculum overview feature in Tracker. Users could click on the nodes they have previously taken, then click the nodes of modules they would like to take, and the links between the nodes would change colour depending on whether or not the user's previous choices allow them to take each module.

10.2.3. Security

Levels of security was one of the original objectives for Tracker. As development continued, and the requirements became more refined, it became apparent that this was an unnecessarily complex addition. As the system stands currently, there is no risk of data leakage that is likely to be prevented by security levels.

However, if student records were to be directly integrated with Tracker, which would be the most convenient route for users, security precautions would be necessary. Furthermore, in the event that Tracker evolves in the direction discussed earlier regarding online accessibility, there may be a need for more advanced security provisions. In this situation, Tracker's back-end would be managing multiple accounts for users who are keeping track of information

about their schools. Individually, there is no risk, but consideration must be given to the possibility of one user gaining access to the data of another.

10.2.4. Visualisations

One of the most obvious ways to expand the usefulness of Tracker would be to explore more areas for analysis. For the purposes of this project, my supervisor and I wanted to focus on skill tracking but there is no reason why the system cannot extend towards other areas of academic interest.

For example, it may be worth analysing coursework submissions. There may be a strong relationship between submission time and final grade. Extensions such as this increase the depth of Tracker because they can be related to existing parts of the system; certain modules or classes may exhibit this relationship stronger than others, which could be reflected in their Tracker pages.

10.2.5. CAPOD

Tracker has received interest from an organization associated with the University of St Andrews called CAPOD (Centre for Academic, Professional and Organisational Development). The purpose of this organization is to work with students and members of staff to provide academic support. Naturally, in this endeavor, they dedicate a lot of resources to learning about the distribution of skills in the many schools they oversee, which is what led to their interest in my project.

I had a meeting with one of their Student Developers to discuss the role that CAPOD plays and the possibility of integrating Tracker into their service. After demonstrating the system in use, we discussed the potential for Tracker to be adapted for use across the entire spectrum of schools within the university. For this to happen, I envision that the main obstacle to overcome would be automating as much of the data entry process as possible. As it stands, users are able to upload students through CSV upload, but associating these entries with their module choices would be a time-consuming task if the system was scaled up to so many records.

The next step here is expanding the CSV upload functionality so that student module choices can be uploaded at the same time as the student entry itself. This would massively cut down on the time needed to get a school loaded into the system, and minimise data maintenance as all subsequent changes could be made through this same facility. I am optimistic about the prospect of integrating Tracker with CAPOD, and very pleased that they would take interest in my project.

11. Conclusion

Universities have an incredible amount of data at their disposal. Such data represents a mostly untapped reservoir of valuable information. Schools regularly have to make important decisions about their future direction; which modules to introduce, when to introduce them, how to adapt to changes in the field, and many others. The more informed they are about their active modules and students, the better equipped they are to make these decisions.

Tracker aims to fill the need for such a system. It provides analysis across all areas of schools, from the broad scope of classes to the individual nuances of students. With a focus on skill tracking, this tool offers the user a wide array of perspectives to take on a complex dataset. Furthermore, Tracker offers a high degree of flexibility; the system can be adapted for use in any university department that offers module choices and is interested in learning about the skills they deliver. The user is granted the power to completely govern their experience.

Tracker has achieved most of its original aims, but it has also provided a foundation for future work in this area. I sincerely hope that some of the possible extensions discussed earlier become a reality in the near future, as the scope of potential for applications like this remains to be seen.

References

Anscombe, F. 1973. Graphs in Statistical Analysis. *The American Statistician*, **27**(1), p.17.

Card, S., Mackinlay, J. and Shneiderman, B. 1999. *Readings in information visualization*. San Francisco, CA: Morgan Kaufmann Publishers.

Corbett, J. 2001. Charles Joseph Minard, Mapping Napoleon's March, 1861. *CSISS Classics*. UC Santa Barbara: Center for Spatially Integrated Social Science.

Flanagan, D. and Matsumoto, Y. 2008. *The Ruby Programming Language*. Sebastopol, CA: O'Reilly Media.

Hartl, M. 2015. *Ruby on Rails Tutorial: Learn Web Development with Rails*. 3rd ed. Boston, MA: Addison Wesley.

Hinrichs, U. (2016). Visual Representation to Amplify Cognition, *CS5044: Information Visualization*. University of St Andrews.

Munzner, T. 2014. *Visualization Analysis and Design*. Boca Raton, FL: A K Peters/CRC Press.

Ortiz, S. 2012. 45 Ways to Communicate Two Quantities. 27 June. [online] *Visually*. [Accessed 14 March 2016]. Available from: http://blog.visual.ly/45-ways-to-communicate-two-quantities/

Acknowledgements

Ruby - https://www.ruby-lang.org/en/

Rails - http://rubyonrails.org/

jQuery - https://jquery.com/

HighCharts - http://www.highcharts.com/

D3 - https://d3js.org/

SQLite - https://www.sqlite.org/

Bootstrap - http://getbootstrap.com/

Git - https://git-scm.com/

Audited (Rails Gem) - https://github.com/collectiveidea/audited

Randexp (Rails Gem) - https://github.com/benburkert/randexp

Flot - http://www.flotcharts.org/

Appendices

A – Set-up Guide
Attached

B - Description, Objectives, Ethics & Resources Document
Available from MMS

C – Initial (Preliminary) Ethics Form
Available from MMS

Set-up Guide

Step 1: Download Tracker (submitted alongside this report, available from MMS)

Step 2: If Rails isn't installed on your system, go to http://railsinstaller.org/en and download the package that suits your operating system

Step 3: Navigate to Tracker and run the command 'rails server'

Step 4: Open a browser window and go to http://localhost:3000