J. Comput. Educ. (2014) 1(4):251–270
DOI 10.1007/s40692-014-0018-6

# Competence analytics

**Vivekanandan Kumar · David Boulanger · Jeremie Seanosky · Kinshuk · Karthikeyan Panneerselvam · Thamarai Selvi Somasundaram**

Received: 30 August 2014 / Revised: 20 October 2014 / Accepted: 21 October 2014 /
Published online: 10 December 2014
© Beijing Normal University 2014

**Abstract** A smart learning environment is characterized by the key provision of personalized learning experiences. To approach different degrees of personalization in online learning, this article introduces a framework called Smart Competence Analytics in LEarning (SCALE) that tracks finer-level learning experiences and translates them into opportunities for customized feedback, reflection, and regulation. The SCALE framework is implemented in four layers: the sensing layer, the analysis layer, the competence layer, and the visualization layer. The sensing layer provides the datasets to support context-awareness through state-of-the-art sensing technologies. The analysis layer, by the means of powerful code analysis tools, derives performance metrics (e.g., learner coding metrics) which serve as input to the competence layer to identify proficiency levels of learners. Finally, a learning analytics dashboard called MI-DASH (visualization layer) allows interaction with performance and competence metrics. The current SCALE system has been used in a study to track the habits, performances, and competences of novice programmers.

V. Kumar · D. Boulanger (✉) · J. Seanosky · Kinshuk
Athabasca University, 1200, 10011 – 109 Street, Edmonton, AB T5J 3S8, Canada
e-mail: david.boulanger@dbu.onmicrosoft.com

V. Kumar
e-mail: vive@athabascau.ca

J. Seanosky
e-mail: jeremie@rsdv.ca

Kinshuk
e-mail: kinshuk@athabascau.ca

K. Panneerselvam · T. S. Somasundaram
CARE Lab, MIT Campus, Anna University, Chennai, India
e-mail: karthikn2789@gmail.com

T. S. Somasundaram
e-mail: stselvi@annauniv.edu

Growth of coding competences of first-year engineering students has been captured in a continuous manner. Students have been provided with customized feedback to optimize their learning paths. The article describes the analytics-based approach pursued in the study and highlights key findings.

**Keywords** SCALE framework · Smart learning environment · Novice programming · E-learning technologies · Big data learning analytics · Competence

## Introduction

Smart learning could mean customized learning that optimizes learning pathways, engages learners in positive interactions, and guides instruction in a goal-oriented fashion. While the why (optimal learning through customization), where (ubiquitous learning interactions), and how (technologies for goal-oriented learning) of smart learning environments (SLEs) are rather obvious at a coarser level, the degree of customization, the scalability of ubiquity, and the integration of learning-related data are still key challenges facing educational technologists. SLEs encompass traditional classrooms as well as online and distance education. Taking learning anywhere and everywhere in a consistent fashion requires technologies that move such as the smart phones supported by 3G and 4G networks (Lee et al. 2012) as well as learning environments that move such as the flipped classrooms at homes. To provide context-aware learning (Yu et al. 2009; Hwang et al. 2008), hardware and software sensors are necessary to recognize the context and the learning needs of the user to tailor learning content and activities. SLEs are expected to be highly distributed and cloud based to accommodate federated and goal-oriented study activities. To make a smart learning environment context-aware, technologies need to collaborate seamlessly, ubiquitously (Ogata and Yano 2004; Hwang et al. 2012), and purposefully in order to recognize the context, translate the knowledge of the context in a proper learning recommendation, and provide learning materials based on that recommendation.

Competency is the ability to adequately perform a task. Historically, the number of years of experience of a person in a given job/role/position used to be the sole currency of competence. Competences of individuals in an organization are typically assessed by people corresponding to the hierarchical reporting structure established within the organization. Contemporary technologies enable the observation of accomplishments of people at the summative level at the completion of tasks, or at the formative workflow level during the completion of a task, or at time intervals set by the organization. These observations could be used as evidences toward competence analysis. Competence analytics, on the other hand, allow for the translation of observational evidences from a learning environment into expressions of competences, where evidences arrive continually and at random times and the types of evidences need not be structured.

Learning analytics is the science of analysis, discovery, and utilization of learning traces in emergent and related levels of granularity. Analysis could include techniques ranging from data mining to machine learning and to big data analysis.

The discovery of new relations and the discovery of even new data include unconventional data, for example, the family income of a politically competing region, inherent economic drivers influenced by a curriculum, and rate of changes in motivation levels of students with respect to weather. Relations of interest include sentiments among learners across collaborating groups, inter-institutional credit transfer policies among institutions, and mutual respect among instructors.

Trace data refer to observable raw data of study activities such as reading, writing, conceptualizing, critically thinking, solving problems, storytelling, and visualizing, where a network of study activities leads to a measurable chunk of learning. Learning measurements can be made using the trace data, and these measurements can be standardized using IMS Metric Profiles defined under the IMS Caliper[1] specifications.

This article introduces the Smart Competence Analytics in LEarning (SCALE) system and outlines how it relates to smart learning environments ("SCALE framework and smart learning" in section), following a literature review on SLEs ("Literature review" in section). "SCALE layers" in section delineates the details of the SCALE framework and describes the application of SCALE in the Java programming domain. Results of a study are described in "A study with SCALE" section. "Future work" in section outlines our vision of SCALE and describes the challenges we expect to face along the way. "Conclusion" in section offers conclusions.

## Literature review

Smart learning environments involve context-awareness. However, context may involve almost anything. Different research projects on SLEs may analyze and focus on different aspects of a context. The precision with which a context is defined and recognized by an SLE will significantly influence its overall performance.

One of the early applications of Smart Learning is from a smart cloud computing framework that uses a model called E4S (Kim et al. 2011). The E4S model consists of four basic services: pull, prospect, content, and push. The researchers rely on built-in sensors in mobile devices to define the user's behavior or environment. The pull service will extract the type of content to be delivered to the user. The prospect service is responsible for the preparation of the learning content to comply with the user's context. The content service generates the content and establishes the connection between the server and the target device. Finally, the push service performs the synchronized delivery of the generated content to the target device.

Another early development toward a smart learning environment is a self-adapting algorithm for context-aware systems, where "*the algorithm is characterized by a closed feedback loop with four phases: monitoring, analyzing, planning, and execution*" (Cioara et al. 2010). As part of the monitoring phase, the algorithm uses the Resources, Actions, Policies (RAP) context model to programmatically describe the sensed environment. The analyzing phase involves evaluating the

[1] Caliper (http://www.imsglobal.org/caliper/).

changes in the context using the context entropy concept to determine the extent to which the context follows a predefined set of policies. The planning phase selects appropriate adaptation action with respect to changes in the context. The execution phase implements the adaptation action.

Yu et al. (2009) discuss a semantic infrastructure, the Semantic Learning Space, for context-aware e-learning. The Semantic Learning Space *"supports semantic knowledge representation, systematic context management, interoperable content integration, expressive knowledge query, and adaptive content recommendation."* They recognize the need to adapt the learning content to the user's context, which is a challenge quite distinct from flexible content delivery. They define the e-learning context as *"a user's prior knowledge, learning style, speed of learning, current activities, goals, available learning time, location and interests."* For example, in a smart learning environment, the system will track the knowledge gap between the current user's knowledge and the targeted learning outcomes and provide the user with the proper learning content to fill that gap taking into account the user's context. In another article, Yu et al. (2007) proposed an ontology-based approach for semantic content recommendation. The recommender goes through the following sequence of steps: semantic relevance calculation, recommendation refining, learning path generation, and recommendation augmentation. While going through these steps, the recommender takes into account knowledge about the learner, knowledge about content, and knowledge about the learning domain in order to offer highly contextualized content.

With a view to provide contextual information to instructors, the Teacher ADVisor (TADV) framework (Kosba et al. 2005) uses Learning Content Management System (LCMS) tracking data to elicit student, group, and class models, and uses these models to help instructors gain better understanding of their distance students. It uses a set of predefined conditions to recognize situations that require educators' intervention. When such a condition is met, TADV generates an advice for the educator, as well as a recommendation for what is to be sent to students. While TADV is focused on the educators' day-to-day activities, SCALE aims at helping them rethink the quality of the employed learning content and learning design. SCALE also encourages students to share their learning experience, reuse additional learning resources collected by their peers, receive fine-grained feedback about their progress, and regulate individually or in groups.

Sherlock (Lesgold et al. 1991), an intelligent computer-based training system, offered smart training environments for technicians to troubleshoot USAF F-15 aircraft's electronic modules. When F-15 modules were suspected of malfunction, technicians attached those modules to test stations to diagnose the modules. About 95 % of the technicians' tasks were considered routine while the real work environment offered very little experience to deal with more complex problems. Sherlock enabled trainees to learn a wide array of problems in a simulation environment and to receive help from a computer-based coach when necessary. Sherlock reviewed trainees' performance, criticized them, compared their actions against an expert's actions, and provided explanations of the expert actions. Sherlock's team claims that *"independent field tests by the Air Force showed that*

*20–25 h of training on the system was roughly equivalent to four years of on- the-job experience."*

A collaborative intelligent tutoring system for medical problem-based learning called COMET (Suebnukarn and Haddawy 2007) was developed to track individual and group knowledge and learning activities in order to sharpen the clinical-reasoning skills of students in problem-based learning. COMET leveraged Bayesian networks to track the knowledge and activity of individuals or groups and used generic tutoring algorithms to provide guidance to students. Suebnukarn and Haddawy (2004) had previously performed a laboratory study which had revealed a high degree of agreement between COMET and human tutors. A second experiment was later conducted with COMET involving 36 second-year medical students. Students were divided into six groups according to their background knowledge. Three groups were tutored by COMET, while the other three groups were tutored by experienced human tutors. All groups participated in a two-hour problem-solving session. The session was preceded by a pretest and followed by a posttest to measure the improvement in clinical-reasoning skills. The tests were evaluated by 10 volunteer general practitioners to ensure a uniform level of difficulty which was critical to measure accurately knowledge gains. Table 1 shows that the mean scores and standard deviations of both types of groups in the pretest were very close to each other, but the mean score of the COMET-tutored groups (65.12) was significantly higher than the mean score of the human-tutored groups (59.46) in the posttest.

As part of the European Air Traffic Management Programme's (EATMP) initiative to increase traffic capacity and improve safety and efficiency simultaneously, a computer-based training (CBT) system for novice air traffic controllers called MemPAC was developed. Its goal was *"to increase the ability of controllers to recognize the basic mechanisms and limitations of the cognitive processes involved in Air Traffic Control (ATC) tasks"* (Bellorini et al. 1999). MemPAC training consisted of presenting everyday life scenarios through interactive exercises, videos, games, and so on. During that training, students participated in classroom lectures, CBT, visits to the real working environment, simulation, and on-the-job training. An experiment has been conducted in the first MemPAC release with eight novice controllers and two participants completely unrelated to ATC. Participant's subject knowledge was tested using questionnaires before and after the MemPAC training course. The results of the experiment showed that the number of right answers increased from 135 to 184, that wrong answers decreased from 64 to 45, and that "I don't know" answers decreased from 31 to 1.

The literature reveals that competence-based analysis and competence-based analytics are now viable. While 'analysis' caters to summative and formative

**Table 1** Mean scores for all clinical-reasoning problems (Suebnukarn and Haddawy 2007)

| Cohort | Mean score (SD) | |
|---|---|---|
| | Pretest | Posttest |
| Comet (all) | 36.36 (0.42) | 65.12 (0.38) |
| Human tutor (all) | 36.54 (0.44) | 59.46 (0.31) |

datasets, analytics caters to continuously arriving, semi-structured datasets. This research specifically targets an analytics-oriented competency management system that tracks both cognitive and task-specific learning processes, at real-time, to identify and address gaps in competences with the goal to optimize the effectiveness and efficiency of learning.

## SCALE framework and smart learning

Smart Competence Analytics in LEarning is a mixed-initiative learning analytics framework aimed at collecting learning traces from a variety of learning domains and analyzing learning traces to extract underlying competence levels exhibited by learners. The SCALE framework has been designed for seamless integration with different types of learning-related artifacts such as the Moodle[2] learning management system, the Web-CAT[3] suite of automated grading and testing tools, or the NetBeans[4] integrated development environment (IDE). The current SCALE system does not focus as much on the physical context of a student (e.g., classroom, instructor availability) as it does with the student's learning context (e.g., learners' background knowledge, learner motivation).

SCALE's layered architecture consists of a sensing layer, an analysis layer, a competence layer, and an interactive visualization layer. The sensing layer is implemented through the Hackystat[5] framework which provides a collection of customizable sensors embeddable in learning-related tools. The analysis layer consists of parsers and analysis tools pertaining to the learning domain. For instance, SCALE's analysis layer applied in the programming domain consists of compilers and static/dynamic code analysis tools. The output of the analysis is then automatically converted and stored in a competence ontology. The competence layer associates competences with learning outcomes and shows evidences on students' learning progressions. Ontologies, implemented using Semantic Web technologies, along with inference engines pave the way toward discovering new patterns and trends in the learning styles and learning paths of students. Competence ontologies hold and define the knowledge background of students, a prerequisite to offer customized learning materials. Finally, the interactive visualization layer provides a graphical interface consisting of a set of visualization and communication tools to play back the student's performance, display the student's competences in relation to recent learning activities, provide an environment where all learning stakeholders (e.g., instructors, students, peers, parents, recruiters, etc.) can meet and create shared goals and explore strategies to achieve them, and give the student the opportunity to comment on his/her learning. The framework offers a plug-and-play architecture to allow any data-centric learning-oriented application to be able to easily plug into SCALE.

[2] Moodle (http://moodle.org).

[3] Web-CAT (http://web-cat.org/group/web-cat).

[4] NetBeans (http://netbeans.org).

[5] Hackystat (https://code.google.com/p/hackystat/).

We aim to make SCALE context-aware in terms of user's prior knowledge, regulated learning (self-regulated and co-regulated), learning style, learning efficiency, current activities, goals, available learning time, location, and interests, as partly defined by Yu et al. (2009). Programming is by far one of those easily traceable (not necessarily analyzable) domains due to the availability of explicit data that identify stepwise progressions made by programmers as they complete their coding tasks. In other words, the number of sensing hours spent by the system to track and update the learning context of the student may be much greater than in a chemistry course (depending on how e-learning is applied to the chemistry field). Due to the great number of environments in which Hackystat sensors (one of the SCALE components) may be embedded, the system is expected to have a better representation of learners' context across a number of study activities in various subject domains.

One important question concerns the degree of ubiquity of SCALE. This may be a tricky question to answer. Consider the Java programming domain as an example. We have integrated multiple programming tools within the SCALE system to capture coding-related activities of learners—for example, Eclipse IDE sensor, SCRL regulation tools, NetBeans IDE sensor, GEOIA conversation analyzer, Virtual Programming Lab[6] (VPL) IDE sensor, and Mixed-initiative learning analytics tutoring and training environment (MI-LATTE) reflection and regulation sensor. The dilemma is about the number of disparate learning environments offered to students and the learning effectiveness of the combination of these environments. To contribute positively to the development of 'good' coding habits and competences, should SCALE prescribe a subset of available tools? This remains an open-ended question for the time being. The ubiquitous nature of the SCALE framework allows it to be pluggable with any tool or system that could supply raw learning traces data.

The SCALE system guarantees reliable data collection through the Hackystat sensors despite low-speed Internet connections and inevitable connectivity issues.

In the future, we plan to provide students with a gamut of recommendations about learning paths, career orientations, job interview preparations, cognitive development, and soft skills. Currently, SCALE provides students with minimalist recommendations about programming concepts that require a deeper understanding to maximize their successful application in the upcoming assignments or exams. The SCALE framework is a work in progress, and the current implementation will incrementally add new functionalities.

## SCALE layers

### Data collection and storage

Smart Competence Analytics in LEarning is a framework collecting and analyzing data from a set of learning tools with the key goal of translating student's

[6] Virtual Programming Lab (http://vpl.dis.ulpgc.es/) and C. The reader should note that the current implementation of SCALE could be straightforwardly adapted to many other programming languages supported by the IDE.

performance into a competence portfolio. In this section, we exemplify the universal nature of the system by describing SCALE's implementation in introductory programming courses in Java.

Any open-source code editor or IDE can fit into the SCALE framework to give SCALE the capability to monitor several types of software engineering activities. So far, SCALE has been tested in real study environments with Eclipse IDE, VPL, and MI-LATTE (specialized tutoring software guiding students' one line of code at a time in a set of programming exercises). VPL and MI-LATTE provide highly scaffolded learning environments for the novice programmer to reduce the first programming experiences to simplest forms. Afterward, students can move forward with the professional and full-fledged Eclipse IDE or any other IDE supported by SCALE. SCALE, through its plug-and-play architecture, empowers instructors to build a unified dataset from all the course tools.

### Sensing layer: Hackystat

Hackystat is an open-source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. Hackystat offers sensing technology with pre-built sensors (e.g., Eclipse and Visual Studio sensors) for the software engineering domain and a generic sensorshell to create custom sensors for a number of other domains.

Hackystat sensors collect data of different types, which are sent to a sensorbase repository. The Hackystat protocol, which regulates the communication between the sensors and the sensorbase, requires that for every sensed data instance, the following pieces of information should be provided: timestamp, runtime, sensor data type (SDT), tool, owner, resource, and properties. The properties are a set of key-value pairs storing learning-trace data as strings, while the other fields serve as metadata for the learning-related data.

Hackystat provides a generic user interface to visualize data as well as a Java library to programmatically handle the retrieval of project data.

The sensorshell enables systems to sense customized data types and transmit them reliably to the sensorbase, in an online environment. It means that if it recognizes any failure in the Internet connection, it will store the data in their original XML format on the client computer awaiting re-transmission once the Internet connection is re-established. Sensors can be hosted both on the client side and the server side according to the security, reliability, and interoperability needs of the system at hand.

But what is the value of such an unstructured dataset? First, this makes up a good starting point to convert data and store them into different data storage technologies such as SQL databases, RDF/OWL ontologies, object-oriented, and document-oriented databases. Relations across such converted data could provide valuable insights about competences of learners and the growth patterns of such competences.

How can those data be queried in order to convert them in the proper format? Querying happens through project creation. Projects are merely datasets. In a

learning context, projects represent universities, departments, courses, students, assignments, or any learning object of interest.

Data instances within a project are identified with URIs. As indicated before, a data instance, in order to comply with the Hackystat protocol, must provide, among other pieces of information, a Resource string. By specifying locator(s), name(s), or both, one may build his/her own dataset for further analysis. Data collection may occur explicitly and implicitly. In the former case, it means that the generation and collection of data are triggered by an explicit (or intentional) action of the student in his/her learning environment. In the latter case, the data are passively collected according to a time frame (e.g., 50 data instances per minute). We may want to differentiate those data in URIs accordingly for more accurate interpretation of the collected data.

### Data types

During the development of SCALE, we enhanced the Hackystat Eclipse sensor to collect more data types about students' problem-solving processes in well-defined programming assignments. This sensor is referred to as Eclipse IDE Extension (EIDEE). The data types collected by EIDEE can be classified into SourceCode, Edit, Build, and Debug. Basically, an Edit data type stores information about changes made to an Eclipse project or a file as well as the API of a class. It can also track code progression at the following granularity: characters, statements, and methods. The Build data type records all compile-time errors generated when the student saves or compiles his/her code. The Debug data type captures the steps taken by a student when debugging his/her code. Stepping into code blocks, stepping over code blocks, setting/unsetting breakpoints as well as starting and ending debugging sessions are among the debugging activities recorded by EIDEE. Finally, the most important data type collected by EIDEE is the SourceCode. Eclipse captures the student's source code as he/she progresses through the assignment.

As previously mentioned, EIDEE can be configured to change the frequency of code capture sensed per minute either to improve the accuracy of analyses or to alleviate the processing load. SCALE is purposefully designed to compile and parse the student's source code at short and regular time intervals as the student progresses in an assignment problem. The source code is therefore compiled programmatically and separately by the Eclipse JDT compiler. All errors are stored in a central repository. Thus, students can view the progression of their errors in all their assignment problems.

The Eclipse JDT allows SCALE to identify approximately 561 error types, with each detected error being assigned an error type and a customized error message. MI-DASH provides a set of visualization tools to visualize these metrics and report to the student the error types which he/she seems to be struggling most with. Moreover, the Eclipse JDT enables to build the abstract syntax tree (AST) of every source code capture to enable SCALE to analyze the growth of the tree as the student works through a specific programming exercise.

Source code is arranged into an AST by breaking it down into 84 programming constructs. These 84 programming constructs (or concepts) can then be mapped to a Java competence ontology to translate the student's performance into proficiency levels.

## Source code processing

In order to provide useful information to students about various aspects of their performance, SCALE retrieves all datasets pertaining to the programming problem at hand. Besides, due to the huge amount of data and the expensive processing to transform those big data into meaningful information, the system provides different levels of abstraction. For example, by default, SCALE retrieves approximately 50 source code captures distributed as equally as possible over the development time of that particular assignment problem. The student also has the capability to modify this abstraction level to fine-tune his/her view of his/her performance. Note that EIDEE could sense student's code at almost every keystroke. Hence, when the student completes a programming exercise in one of the assignments, the system could have collected several thousand captures of the source code. Compiling every capture consumes a significant amount of computing power and produces little benefit to the student. The 50-instance approach indicates to be a reasonable balance between processing speed and analysis accuracy.

This approach also fits well with the processing of the StateChange data type. This is a subtype of the Edit data type, and it tracks the number of characters, programming statements, and methods (defining the API of a Java class) at a given point in the problem-solving process of an assignment problem. Perhaps thousands of StateChange data instances will be generated during the development of a solution to a coding problem. As before, SCALE will pick up an equally distributed data subset over time to show the progression of a student's code.

For each code capture (approximately 50 for each problem), SCALE has the ability to construct an AST and retrieve its enclosed errors. The system, by default, shows the tree of the latest version of the student's source code for a given problem. As an extension, MI-DASH enables the student to view the growth of his/her AST for the entire duration of the coding episode.

As for the errors generated for each compilation, every error is stored in a database along with the related source code build. Other compile-time data being collected include error type, error message, start and end character positions, and source line number. Error types can then be aggregated as a list of errors ordered by pedagogical priorities of the instructor or the student.

While low-level bugs are directly and automatically detected, the detection of higher-level misconceptions requires other reasoning mechanisms. SCALE, currently, employs a rule-based approach to record specific misconceptions or concepts. For example, SCALE can recognize code corresponding to Recursion or a Software Design pattern.

Analysis layer

Mixed-initiative learning analytics tutoring and training environment tutors provide a guided environment for solving a programming exercise one line of code at a time. The embedded software agent offers instructions for each line of code. Tutoring consists of five different types of feedback: instructions, error detection feedback, compile-time errors, corrective actions, and solution code snippets. There are different instances of MI-LATTE tutors for different problems. MI-LATTE tutors support students to write a valid piece of code one line at a time. So, it is quite natural to capture code every time the student presses the return key. Moreover, MI-LATTE builds the code every time the student requests one of the five feedback types. For example, a programming exercise consisting of 18 lines of code could have generated a total of 36 builds, of which 18 are for each line of code and 18 are for the instruction requests for each line of code. Figures 1 and 2 show the number of errors observed in the code during the problem-solving process and the number of lines of code written so far for each build. If the number of errors at the last build is zero, then MI-LATTE assumes that the student has written a valid code. The students' requests for feedback, the contexts in which they are made, and the types of feedback (i.e., the students' interaction with the software agent) are all recorded by the system. If there exists a plateau as noted in Fig. 2, MI-LATTE infers that the student requires help with that specific line of code. These types of analysis are also provided for each assignment problem solved from within the Eclipse IDE.

MI-DASH can be used to compare the completion percentage of a student's assignment problem in relation to a proposed model solution and to compare his/her code-writing speed against the average speed. Such analytics are derived from the analysis of the StateChange data type.

All measures can be shown in terms of characters, statements, and methods. For example, Fig. 3 indicates that the student has written 460 characters out of the expected 4,066 characters in the model solution. At the far left side, the red line indicating the student's progress should theoretically start at zero and end at the far right somewhere near the green line.


**Fig. 1** MI-LATTE tutor: number of errors per build

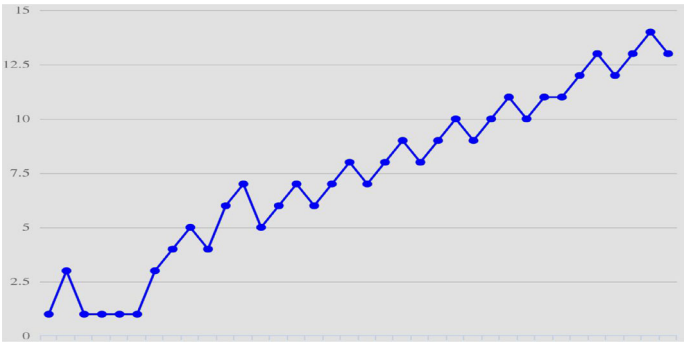


**Fig. 2** MI-LATTE tutor: number of lines of code at each build

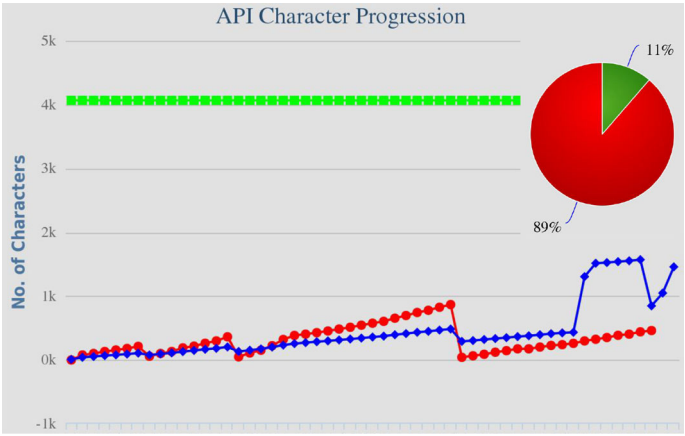


**Fig. 3** Eclipse: character progression (candidate.java)

The reader should note that there could be several model solutions to a problem. Thus, the end of the red line should only be in the vicinity of the green line when the problem is almost done.

The blue line indicates the average progress of all students. Moreover, if a group of students starts working simultaneously on a course assignment, then the current student could compare his/her progress in terms of the aforesaid programming metrics against the average of the group. Hence, the student might find that he/she is getting behind or ahead of the group.

Smart Competence Analytics in LEarning also enables students to compare their performance in specific rubrics, such as code quality, writing speed, code functionality, quality of test case performances, extent of documentation, and capacity to regulate, against the average and/or top performers of the class.

Smart Competence Analytics in LEarning also unfolds the AST of the latest version of student's source code for every assignment problem. Students can visualize the AST tree in MI-DASH and view his/her code associated with each node in the tree.

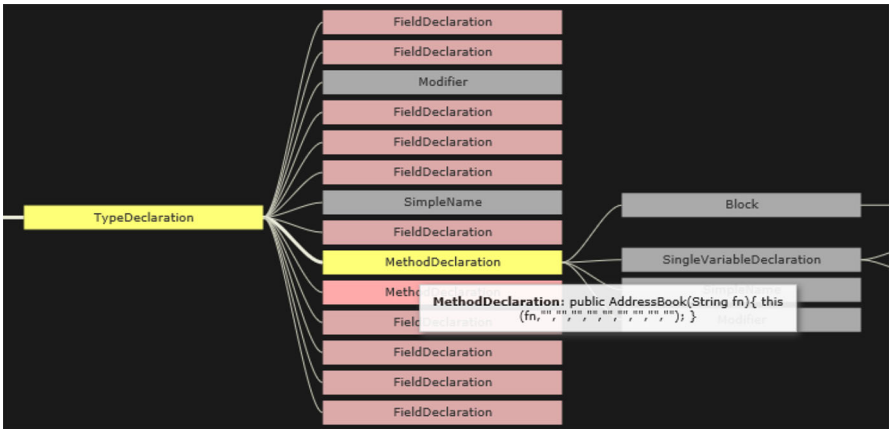


**Fig. 4** AST (Abstract Syntax Tree) viewer

Note that each node corresponds to one of the 84 Java concepts as specified by the Eclipse JDT. Allowing students to view the AST of their code (see Fig. 4) is very instructive in that it helps them to associate their code with concepts taught.

Smart Competence Analytics in LEarning also allows identified concepts to be mapped on to cognitive development metrics such as the Bloom's taxonomy.

Another fundamental metrics SCALE offers students are the identification of potentially problematic areas in their understanding. MI-DASH guides students to identify errors or concepts that should be corrected or reviewed. These errors or concepts are assigned a priority in order to convey their importance in relation to other errors. These errors are laid out in a quadtree[7] with each level in the tree showing a different degree of criticality. The weight of each error type takes into account the frequency of the errors made across all assignment problems as well as the time the errors remained uncorrected in every specific programming problem. The student should be aware that the error priorities have been computed from the full list of errors captured from a suite of programming tools from among a set of programming exercises.

Finally, the student can view the 84 programming concepts specified by the Eclipse JDT compiler and their frequency of use. This will provide some evidence to instructors that the student has been exposed to or practiced or encountered errors with certain concepts introduced in the course. This data may pinpoint some weaknesses and/or strengths. For instance, a programming construct which has never been used during the entire course may suggest a deficiency in student's learning (see Fig. 5).

Competence layer

Data stored in the Hackystat sensorbase along with SCALE analysis results are converted into RDF/OWL ontologies. By building a huge web of ontologies and

[7] Wikipedia, the Free Encyclopedia. "Quadtree." (http://en.wikipedia.org/wiki/Quadtree).

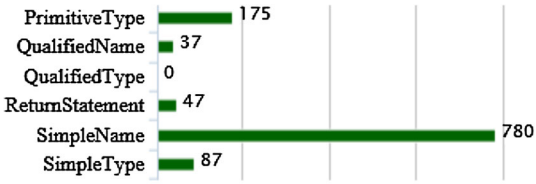


**Fig. 5** Concept graph (Frequency of Use)

importing them into an ontological and rule-based reasoning engine (e.g., BaseVISor),[8] the system may discover new patterns and trends in study habits yet unknown to instructors. For instance, at the successful completion of a Java course, the student's proficiency in Java will be stored in a Java competence ontology. In turn, the Java competence ontology could be mapped to an object-oriented ontology which at the end could also be mapped to a C++ ontology. Therefore, we could evaluate the readiness of a student having completed a Java course to take another course requiring C++ background. Such metrics offer a measure of confidence to instructors and students about the student's probability of success.

Smart Competence Analytics in LEarning ontologizes the entire learning space and all learning episodes of students so that instructors and students can query the distributed ontology to infer information about learning.

## A study with SCALE

A study was conducted at the Madras Institute of Technology (MIT), Anna University, India, to analyze the introduction of trace-oriented learning technologies among first-year university courses. The traces target the evolution of the student's learning experience in terms of well-defined skills and associated competencies. The study was designed in the context of a C programming course with 767 participating students and 10 professors (one professor per classroom). Students belonging to nine different classrooms received traditional lectures while a randomly chosen 10th classroom received a traceable online learning environment in Moodle in addition to classroom lectures. The e-learning technologies introduced in the course include the Moodle learning management system, the VPL, the Eclipse IDE sensor, and CTAT[9] tutors. The CTAT tutors guide students to solve programming exercises at a finer level of one line of code at a time.

The study content was presented to students using a quadrant-based framework (Kumar et al. 2009; Lee et al. 2011a, b, 2010). The new design offers four tightly related learning experiences based on the theory of experiential learning. The four modes of learning are watching, discussing, conceptualizing, and trying out. It also provided guidelines to the student as to how to study in addition to what to study. The collected data were integrated in a singular framework to associate individual datasets originating from different sensors.

[8] BaseVISor (www.vistology.com/basevisor/basevisor.html).

[9] Cognitive Tutor Authoring Tools (http://ctat.pact.cs.cmu.edu).

The objectives of the study include the discovery of new trends in learning experiences and examination of the influences of these trends on students' performance. The reader should note that the study was not a controlled experiment.

Participants of the experiment originated from 10 different classrooms. Approximately 75 students attended each classroom, and each classroom had a distinct professor. Classrooms were numbered from classroom0 to classroom9. Classroom3 is the classroom of interest in this experiment. The performance of classroom3 is compared with the average performance of all the other classrooms.

All classrooms teach the same course using the same structure. The course consists of three assessments/assignments, one theory exam, and one practical exam. Assessment1 consisted of theoretical questions while Assessment2 and Assessment3 consisted of programming exercises. Hence, classroom3's students were not yet exposed to the VPL tool in Assessment1.

Figure 6 displays the average marks for each assessment for both classroom3 and the rest of the classrooms. Figure 7 shows the percentages of students who passed the assessments. Both figures show that classroom3 performed below average in the first assessment likely because (1) classroom3 students took additional time to adapt to the new instructional design, (2) the new instructional design is not optimal for

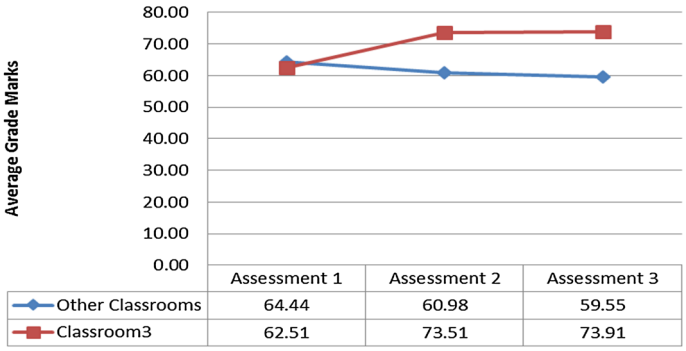


**Fig. 6** Assessment average grade marks

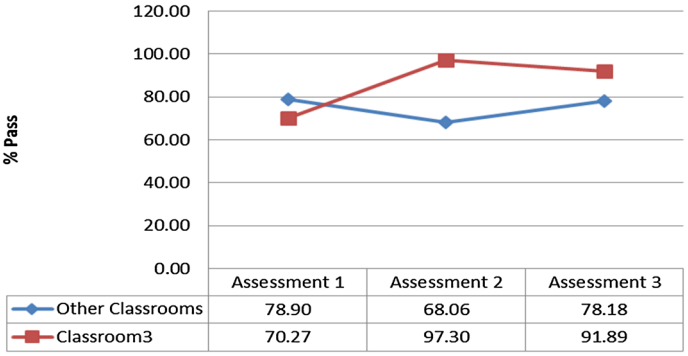


**Fig. 7** Assessment % pass

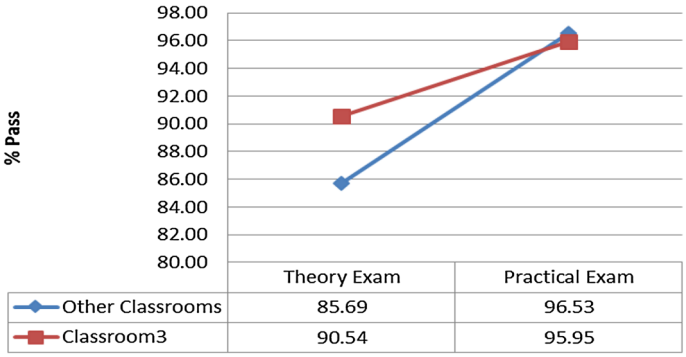


**Fig. 8** Exam % pass

theoretical aspects of the programming domain, and (3) the new instructional design did not offer adequate support to students.

The graphs, however, show that average marks and pass percentages of classroom3 are significantly higher than those of the other classrooms in Assessments 2 and 3. We speculate that classroom3′s students started benefitting from the new instructional design and new e-learning technologies after Assessment1.

As for the theory and practical exams, Fig. 8 denotes that the percentage of students in classroom3 who have passed the theory exam is greater than the percentage of students in other classrooms. This observation may support the speculation that students took more time to adapt to the new instructional design but that this design optimized the students' learning experience at a later time.

As for the practical exam, the percentage of students in other classrooms who passed the practical exam is marginally greater than classroom3′s students pass percentage. However, both classroom3 and the other classrooms students performed well in the practical exam. We may, nevertheless, observe that the pass percentage gap between the theory and practical exams is greater in the other classrooms.

Finally, in Figs. 9 and 10, we observe that fewer students in classroom3 have grades C, D, or E, and more students in classroom3 receive grades S, A, and B. Note that the order of GPAs from best to worst is S, A, B, C, D, and E.

All these are mere observations and suggest some trends. More experiments will be conducted in the near future in several universities across the world to understand the impact of SLEs on student performance.

## Future work

Our vision for SCALE transcends geographical boundaries and allows students from across the planet to be able to connect, share, and learn as a collective.

One of the key challenges in competence analytics is data integrity that ensures the authenticity of the arriving data and the mapping of the data onto competencies. Such mapping requires external as well as internal authentication.

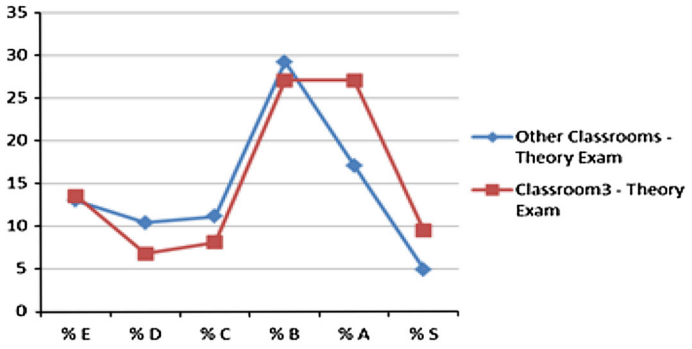


**Fig. 9** Theory exam % GPA

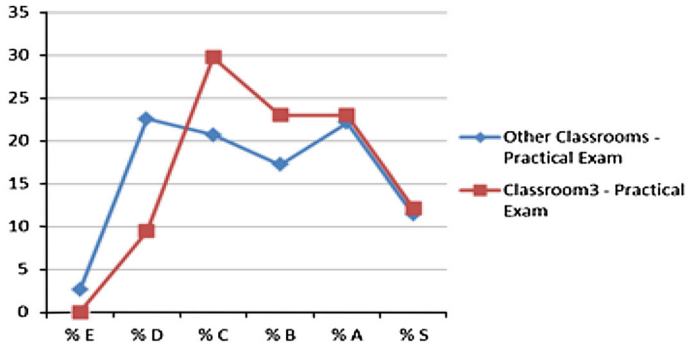


**Fig. 10** Practical exam % GPA

Data can be processed by any technique, whether it is a data mining technique or a machine learning technique, or a big data clustering technique, or a sentiment analysis technique. We advocate that students and instructors should be equally in control of the choice of analyses and the corresponding techniques.

In addition to the students initiating interactions with the analytics tools, the tools themselves could also be initiating interactions with students.

Performance indicators, by themselves, offer reflection opportunities. Competence analytics enables students and instructors to proceed further, toward regulation opportunities.

Analytics being offered through SCALE are available without "any" intrusion on the student's course interactions. There is no controlled experiment, as such. Instead, data are being collected in the background, with students' explicit permission. We plan to conduct more controlled experiments in the near future.

Competency growth in programming can be observed within a specific language (say, Java) across multiple courses. Equally, competency growth in programming can be observed across multiple languages (say Java and C++). Competency growth in programming can be observed at the individual student level, or the course level, or the departmental level, or even at the institutional level.

Key inferences about intervention opportunities could be made available to instructors. These opportunities arise out of competence analytics with substantiating evidence. Intervention opportunities could be tuned toward learning efficiency of individual students and groups of students.

The development of a comprehensive coding traces platform, SCALE, reporting on the efficiency of coding and effectiveness of coding instruction is a key contribution of this research. It not only reports on learner habits but also relates instructor habits. The traces platform is quite conducive to the development of software agent-oriented guidance that can monitor the data sources on a student-by-student basis, to identify successful patterns of learning, based on a student's interaction with the digital learning environment, and suggest corrective actions, materials, and behaviors that might improve performance, based on the past results of similar student behaviors.

The underlying datasets approach the quantities expected in the context of big data analytics. The platform that we have developed supports the inclusion of Hadoop[10]-based analytics that can handle large volumes of data.

Coding efficiency encompasses aspects of coding style, metacognitive scaffolds, peer interactions, and interactions with instructors, among others. Instruction on effective coding encompasses aspects of intervention, guidance, content material, sensitivity to learner capacity, and timeliness of feedback, among others. Datasets corresponding to each of these aspects can be observed, and relations among these datasets can be established using the platform.

We envision creating a customized 3D environment displaying the scope of the learning space of an individual, classroom, or the institution. Every learning space could report learning metrics corresponding to that space.

Furthermore, we plan to build a distributed NoSQL sensorbase using Hadoop, an object-oriented database such as HBase, and a document-oriented database such as MongoDB. Hadoop will enable us to distribute data processing and data querying across all datasets collected by the SCALE network and to discover new patterns by creating a highly distributed dataset instead of analyzing each data subset individually.

Smart Competence Analytics in LEarning will also incorporate a causal learning analytics (CLEAN) extension to determine the causes of various learning-related occurrences.

Smart Competence Analytics in LEarning will track the type and sequence of programming activities (debugging, building, testing, documenting, code writing, etc.) typical for every student category (at risk, average, and top students).

Smart Competence Analytics in LEarning will also look for the learning approaches and behaviors which are the most effective as well as the conceptual causes of students' errors.

The CLEAN extension will be implemented as a rule-based subsystem using pattern-matching techniques (e.g., production rules). In summary, SCALE aims at tracking a student's competences in as many learning activities as possible and at explaining the factors contributing to the strengthening of those competences.

[10] Apache Hadoop (http://hadoop.apache.org).

## Conclusion

This article describes SCALE, a framework for competence-based learning analytics. The framework can be embedded in a smart learning environment to offer guidance, feedback, and regulation opportunities to students based on observed study activities. The framework has been implemented in the domain of Java programming, and a study was conducted at a partner university in India. The results are quite encouraging, and student satisfaction with competence analytics is found to be high. The potential of having a global implementation of the framework that seamless connects students studying similar courses or concepts has been highlighted.

## References

Bellorini, A., Larsen, M. N., Pistre, M., Considine, B., Woldring, M., & Van Damme, D. (1999). Computer based training on human factors for Ab-Initio air traffic controllers. In *Tenth International Symposium on Aviation Psychology*.

Cioara, T., Anghel, I., Salomie, I., Dinsoreanu, M., Copil, G., & Moldovan, D. (2010). A self-adapting algorithm for context aware systems. In *Ninth roedunet international conference* (pp. 374–379).

Hwang, G. J., Tsai, C. C., Chu, H. C., Kinshuk, & Chen, C. Y. (2012). A context-aware ubiquitous learning approach to conducting scientific inquiry activities in a science park. *Australasian Journal of Educational Technology, 28*(5), 931–947.

Hwang, G. J., Tsai, C. C., Yang, & Stephen, J. H. (2008). Criteria, strategies and research issues of context-aware ubiquitous learning. *Educational Technology & Society, 11*(2), 81–91.

Kim, S., Song, S.-M., & Yoon, Y.-I. (2011). Smart learning services based on smart cloud computing. *Sensors, 11*(8), 7835–7850.

Kosba, E., Dimitrova, V., & Boyle, R. (2005). Using student and group models to support teachers in web-based distance education. In *Proceedings of the 10th international conference on user modeling* (pp. 124–133).

Kumar, V., Manimalar, P., Somasundaram, T. S., Sidhan, M., Lee, S., & El-Kadi, M. (2009). Open instructional design. International workshop on technology for education (pp. 42–48).

Lee, S., Barker, T., & Kumar, V. (2010). Approaches to student modeling in the context of e-learning 2.0. In *Proceedings of the 9th European conference on e-learning* (pp. 59–66).

Lee, S., Barker, T., & Kumar, V. (2011a). Learning preferences and self-regulation—design of a learner-directed e-learning model. *Software Engineering, Business Continuity, and Education, 257*, 579–589.

Lee, S., Barker, T., & Kumar, V. (2011). Models of eLearning: The development of a learner-directed adaptive eLearning system. In *Proceedings of the European conference on e-learning* (pp. 390–398).

Lee, J., Jung, Y. J., Park, S. R., Yu, J., Jin, D.-s., Cho, K. (2012). A ubiquitous smart learning platform for the 21st smart learners in an advanced science and engineering education. In *15th international conference on network-based information systems* (pp. 733–738).

Lesgold, A., Katz, S., Greenberg, L., Hughes, E., & Eggan, G. (1991). Intelligent coached apprenticeship systems: Experience from the Sherlock project. In *IEEE international conference on systems, man, and cybernetics* (pp. 1725–1737).

Ogata, H., & Yano, Y. (2004). Context-aware support for computer-supported ubiquitous learning. In *2nd IEEE international workshop on wireless and mobile technologies in education* (pp. 27–34).

Suebnukarn, S., & Haddawy, P. (2004). A collaborative intelligent tutoring system for medical problem-based learning. In *International conference on intelligent user interfaces* (pp. 14–21).

Suebnukarn, S. & Haddawy, P. (2007). COMET: A collaborative tutoring system for medical problem-based learning. *Intelligent Systems, IEEE*, 70–77.

Yu, Z., Nakamura, Y., Jang, S., Kajita, S., & Mase, K. (2007). Ontology-based semantic recommendation for context-aware e-learning. In *4th international conference on ubiquitous intelligence and computing* (pp. 898–907).

Yu, Z., Zhou, X., & Shu, L. (2009). Towards a semantic infrastructure for context-aware e- learning. *Multimedia Tools and Applications, 47*(1), 71–86.

**Vivekanandan Kumar** is an Associate Professor in the School of Computing and Information Systems. His research interests are in applications of causal modelling in big data learning analytics, artificial intelligence in education, collaborative learning, self-regulated learning, co-regulated learning and mixed-initiative interactions using anthropomorphic pedagogical agents.

**David Boulanger** is a research assistant and undergraduate student at Athabasca University. His areas of research interests include bigdata collection, ontologies, analytics, and visualization of learning-oriented data.

**Jeremie Seanosky** is an undergraduate student and research assistant at Athabasca University. His areas of research interests include bigdata analytics, learning analytics sensors, failsafe data collection, retrieval, analysis, visualization, and automated assessment.

**Kinshuk** holds the NSERC/iCORE/Xerox/Markin Research Chair for Adaptivity and Personalization in Informatics. He is also Full Professor in the School of Computing and Information Systems and Associate Dean of Faculty of Science and Technology, at Athabasca University, Canada. Areas of his research interests include learning analytics; learning technologies; mobile, ubiquitous, and location aware learning systems; cognitive profiling; and interactive technologies.

**Karthikeyan Panneerselvam** is a PhD scholar at MIT Campus of Anna University, Chennai. His areas of research interests include educational technology, self-regulated learning, recommender systems and learning analytics.

**Thamarai Selvi Somasundaram** is Full Professor and Dean of the MIT Campus at Anna University, India. Areas of her research interests include high performance computing, educational technology, cognitive modelling, and instructional design.