A new approach for teaching programming: Model-based Agile Programming (MBAD)

Rainer telesko
Fhnw University of Applied Sciences and Arts Northwestern Switzerland

Knut Hinkelmann
FHNW University of Applied Sciences and Arts Northwestern Switzerland

Charuta Pande
FHNW University of Applied Sciences and Arts Northwestern Switzerland

Maja Spahic-Bogdanovic
FHNW University of Applied Sciences and Arts Northwestern Switzerland

ABSTRACT

Designing courses for introductory programming courses with a heterogeneous audience (business and IT background as well) is a challenging task. In an internal project of the School of Business at the FHNW University of Applied Sciences and Arts Northwestern Switzerland (FHNW) a group of lecturers developed a concept entitled "Model-based agile development" (MBAD) which supports the learning of elementary programming concepts in an agile environment and builds the basis for advanced courses. MBAD will be used as a basic learning module for various Bachelor programs at the FHNW.

CCS CONCEPTS

• **software development**; • **model-based approach**; • **agile paradigm**;

ACM Reference Format:
Rainer telesko, Fhnw University of Applied Sciences and Arts Northwestern Switzerland, Maja Spahic-Bogdanovic, Knut Hinkelmann, and Charuta Pande. 2023. A new approach for teaching programming: Model-based Agile Programming (MBAD). In *2023 The 8th International Conference on Information and Education Innovations (ICIEI 2023), April 13–15, 2023, Manchester, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3594441.3594445

1 INTRODUCTION

In this section we shortly describe the Bachelor programs at our university FHNW which form the basis of our research. We show how programming is currently organized and the difficulties students are facing.

IT plays more and more the role of an enabler in business. Since more than 20 years, the School of Business at the FHNW has been offering the Bachelor programs entitled WI (in German "Wirtschaftsinformatik") and BIT (Business Information Technology) where students learn how to support companies with state-of-the-Art IT solutions.

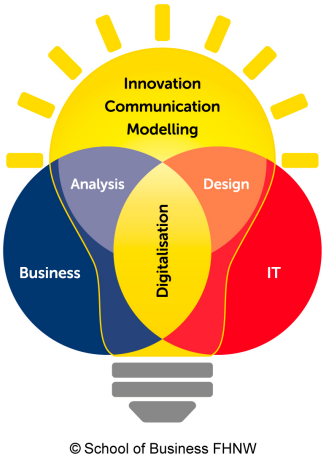

Figure 1: Design of WI / BIT programmes

In Figure 1 the design of the WI / BIT programmes is shown. The two circles "Business" and "IT" represent the pillars management science and computer science. Management science comprises "business" subjects like finance, supply chain management etc. which are of particular interest for WI / BIT students. Computer science is an analogy on the IT side with subjects like programming, databases, IT-security etc. The intersection "Digitalization" makes up the added value of WI and BIT. In courses like Business Process Management, Digital Business Models etc. students learn how IT can be used to bring the business forward. On top of the diagram innovation, communication and modelling highlight topics where WI / BIT seek to put a particular focus also with regards to competitors in this field.

Even though WI/BIT are not dedicated "Software Engineering"- or "Programming"-programs, understanding basic programming concepts is essential for successfully passing the assessment stage of the programs. Over the years, it turned out that the drop-out rate in programming courses is significantly higher than in other WI / BIT courses which led to a redesign of the software engineering part. For all students, passing basic programming courses is mandatory, for the students aiming at deepening this topic, a specialization entitled "Software Engineering Leadership" is offered. Nevertheless, fundamental difficulties when teaching programming remain which are summarized in section 3.

ICIEI 2023, April 13–15, 2023, Manchester, United Kingdom

ACM ISBN 979-8-4007-0061-3/23/04.
https://doi.org/10.1145/3594441.3594445

2 RESEARCH DESIGN

This paper deals with the challenges of teaching object-oriented (OO) programming for undergraduate students. The research question can be formulated as follows:

RQ: How can a concept be defined for teaching (OO) programming in undergraduate courses by adequately considering the challenges reported in practice and scientific literature

The approach starts from the application instead of the technology. It first considers a business challenge and a need to be addressed. This is modelled computer-independently in terms of activities and decisions, similar to modelling a business process. This helps to understand the problem better and identify the classes and objects, which are then implemented in a future step using a programming language.

In the MBAD project, we combined Design Science Research with agile development to create a modelling-based agile development teaching approach (MBAD). Following the Design Thinking process, the first step was empathizing with students to understand the problem. The current situation of teaching programming was analyzed by gathering input primarily from students. The weaknesses were made evident, and the optimization potential addressed by the new teaching approach was identified.

In the second step, the findings were discussed in several workshops with the heads of the WI/BIT programs and BIT lecturers teaching programming. The output of the workshops resulted in innovative ideas on how the approach can be developed. The prototyping followed an agile approach with several sprints in which different deliverables were created and tested. Prototyping needed several iterations before the final version of the MBAD approach was released.

A pilot implementation of MBAD is envisaged for the new Bachelor "Business Artificial Intelligence" to be launched in the autumn term 2023.

3 RESULTS FROM THE ANALYSIS

In this section results from an FHNW-internal analysis about challenges in programming courses together with some promising concepts to overcome these challenges are summarized.

3.1 Challenges for students when starting to learn programming

In the WI / BIT programmes, there are two basic programming courses introducing Java, entitled "Programming 1" and "Programming 2", each of them having a size of 5 ECTS. "Programming 1" gives an overview of basic concepts like variables, control structures, etc. and an introduction into OO. The goal of "Programming 2" is to practice enhanced concepts like Threads, Collections, Exceptions, basic design patterns like Model-View-Controller, database integration etc. and to enable students to write larger programs. As frontend technology JavaFX is used, as Integrated Development Environment the students can choose between Eclipse and IntelliJ. Based on interviews with students and teachers in 2021 the following main challenges have been identified:

- Programming starts with the formulations of an algorithm which requires analytical capabilities. Researchers generally agree that abstraction ability is a necessary skill for programming ([1], [3], [8], [10]). This capability (like understanding a sorting algorithm, ordering concepts in a hierarchy etc.) is often not adequately present at the beginning and also differs from what students already know from subjects like mathematics (e.g. solving (in)equalities, assignment operator etc.).
- In introductory courses, usually teachers have to decide if they go for "OO first" as leading teaching paradigm or not. Starting with OO has the main advantage that students learn to apply this way of thinking right from the start with extensive examples, however face completely new and sometimes disturbing concepts like inheritance, late binding, overloading and overriding in the first weeks of teaching. Starting with old-fashioned procedural programming like control structures, datatypes etc. makes life easier for students at the beginning. When learning OO later, students may ask what paradigm to choose (procedural or OO) and very often bypass or avoid OO because they are unsure how to use it. During the interviews with students, it turned out that despite a more "OO first" approach promised at the beginning, the level of thoroughly understood OO concepts at the end of the first term was still very poor.
- Programming means doing a lot of things doing at the same time, like developing a concept, learning the syntax, constructing or adapting code and using an Integrated Development Environment. This often causes an overload and stress for students, because they quickly see that they may run out of time. Furthermore, especially when starting with programming, the results are poor in terms of visible output (e.g. console applications), therefore additionally decreasing motivation.
- Developing an application is a multi-disciplinary process requiring knowledge from disciplines like project and infrastructure management, requirements engineering, database management, programming etc. In most cases however, the curriculum is fragmented in many modules and thus neglecting these interrelationships. This leads very often to the fact that programming is dealing with mathematical or "toy" problems which is a clear contradiction to the main idea of a BIT program. This fact is also the main trigger for developing the new MBAD approach.
- Syntax is often causing confusion and provoking "why"-questions. When comparing simple print statements in Java and Python, the student usually does not understand the necessary boilerplate code in Java.
- Using concepts and methods from industry usually does not work, because in industry the focus is not on syntax or sophisticated algorithms, however on an adequate division of labor (e.g. frontend vs. backend) and productivity issues (e.g. DevOps etc.). Additionally, some concepts taught often at a university like recursion are frowned upon. Generally, "programming in the large" as relevant for industry and covering topics like multi-layer applications, object-relational-mapping and integrating legacy systems totally differs from

"programming in the small" which is mostly taught in introductory courses and typically covers simple desktop applications with a web frontend.

3.2 Existing approaches for teaching agile software development

In [2] the authors present an experience report about teaching agile software development where two teams of students developed web applications for a real customer. The course ran over 13 weeks and consisted of lectures, group assignments, and lab work thus containing typical entry qualifications for junior software developers in industry. Students learned typical agile practices by practicing continuous deployment, acceptance testing, refactoring, unit testing, incremental design, retrospective analysis, iterative planning, pair programming, progress tracking and lean engineering management. Overall, 14 students enrolled in the course, with one half undergraduates and the second half graduates. It was expected that students would spend at least 10 hours per week on this course

The authors conclude in [2] that the teams successfully finished the projects with running software for both projects. With regards to the course setting, the authors focus on three issues which need particular consideration. Firstly, there were too many lectures which in programming courses often may result in distracted and tired students. Secondly, the customer should have been involved more, and finally a "scope creep" - a disability to assess the true size of work and work changes - led to an unbalanced workload in teams.

In [7] the authors describe the evolution of an agile student project, which over three years has included students from three different faculties. In the first year, only students from the Master of Software Development (SwDev) participated in the group work, in the second year students from the Master of User Experience (UX) joined the project and in the third year the full stage of expansion was reached by integrating students from the Master of Professional Business Analysis (BA). Lessons learned from this project are important when setting up similar endeavors with complementary expertise like SwDev, UX and BA. First, the authors stress that the different cohorts have to collaborate in order to fully exploit their expertise. Each team member should be actively involved in the project - but not necessarily in every phase - and thus contributing to the success of the project. Furthermore, the authors underline the importance of actively engaged industrial partners during the project. Here, the role of the Product Owner - as "voice of the customer" - is central in the case of using Scrum as agile methodology. An "outsourced" or "weak" Product Owner will negatively impact the motivation of students and thus endanger the success of the whole project. The agile projects involved over 70 students from the three Master programmes SwDev, UX and BA demanding close programme-spanning coordination. One crucial point was the definition of adequate learning objectives comprising the creation of artefacts in all three disciplines. The authors make an interesting recommendation with regards to coping with the steep learning curve in real development projects. They propose that each team should preferably solve problems using its own code base, thus avoiding frustration when facing communication and technical integration issues in a "super team" handling the entire code base. Modern software engineering is supporting this idea by offering respective architectural patterns like Model-View-Controller, Client-Server, Microservices etc.

In [6] the authors present a curriculum for teaching programming in the Bachelor of Computer Science at the FHNW University of Applied Sciences and Arts Northwestern Switzerland. Teaching introductory programming consists of three pillars: Programming in Java (first year), Software Development (second year) and a Project Module (first and second year). In "Programming in Java" the students get acquainted with the basics of programming based on a true "Objects first" approach and taught with the flipped classroom paradigm. Flipped classroom means "turning the tables" so that students learn the content at home (via books, slides, videos etc.) and practice it at school. In the subsequent module "Software Engineering" the focus is on software development with iterative (Rational Unified Process) and agile methodologies (Scrum, Extreme Programming). Teachers use the proven "Lego Scrum city game" here to make the agile artefacts, roles and ceremonies clear. At the end of each term in the first and second study year students execute an agile software development project over one whole semester in "real" teams of five to nine members. The authors emphasized the fact that the IT infrastructure available for software development projects (i.e. for managing requirements, adding and sharing code etc.) as well as collaboration tools (i.e. learning management systems, Wiki etc.) are very helpful to boost productivity and increase transparency.

An approach focusing more on the way of teaching and related artefacts than on using a customized agile method to be used in classes is discussed in [9]. In order to find out promising practices for introductory programming courses, the authors designed and distributed a questionnaire among teachers of major Danish educational institutions. They quickly found substantial evidence that teachers prefer to work with an integrated tool supporting more explicitly the sequence of needed activities during a hybrid programming lecture and enabling an automatic monitoring of students. In particular, according to the authors such a hybrid classroom requires at least a tool for video-lecturing, a slide presentation tool, a way to deliver and retrieve exercises and solutions from the students, and if possible, tools for collecting feedback and assessment. The authors believe that an integrated, specialized solution for software development - covering all the functionality listed above - should be preferred in order not to deteriorate learning performance with topics like getting acquainted with multiple tools, media breaks and data integration issues.

An interesting idea described in [9] is to support content with linear and non-linear parts. The teacher presents content (mostly a well-defined chunk of knowledge like a chapter or section), spanning relatively short intervals which are followed by student-paced non-linear parts. The non-linear part is triggered by questions which lead in case of a wrong answer to a more detailed explanation and in case of the correct answer to a more challenging question. Since quizzes and exercises' delivery will be integrated in this tool, students' progress can be automatically recorded. This allows the teacher to quickly react to unforeseen (aka non-linear) events and dynamically adapt learning and grouping strategies to performance measured in real-time. We believe that this strategy

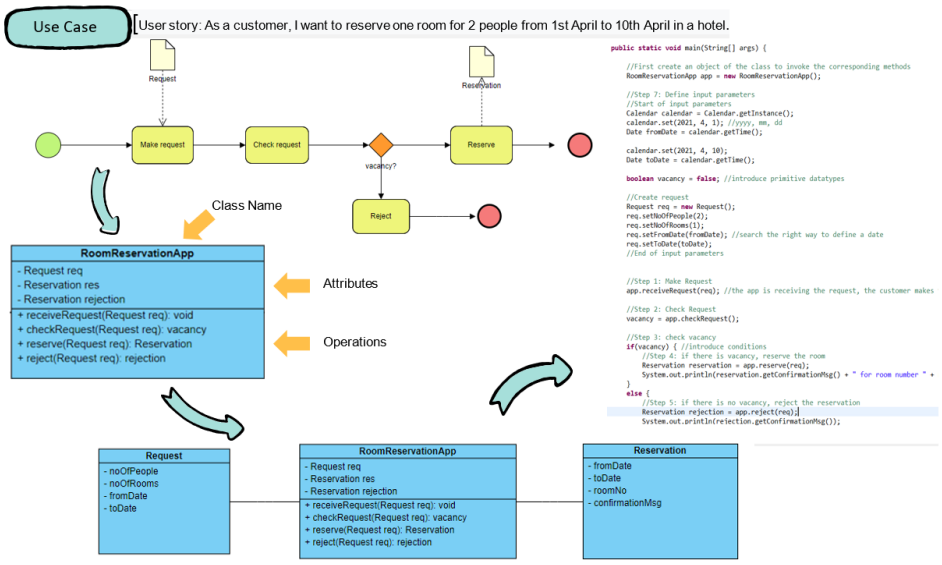

Figure 2: From Business Use Case to Models and Code

would be a good remedy against one of the core problems in teaching programming, i.e. "loosing" students quickly even if the teacher believes that the pace is easily manageable.

eduScrum [4] is a tailored version of Scrum which is used for educational purposes. Our MBAD approach is based on eduScrum roles, ceremonies and artefacts. The main idea of eduScrum is that teams complete assignments for a well-defined topic in timeboxes. The students plan and manage their own learning activities in a way a software engineering team is keeping track of its shared codebase. The role definition in eduScrum is slightly different than in Scrum, by foreseeing the role of a Team Captain instead of a Scrum Master. The Team Captain is a team member and supports the team as a Servant Leader. The main task of the Captain is to set up an interface with the teacher, the responsibilities may vary according to the experience.

4 MBAD APPROACH

In the WI/BIT programs at the FHNW School of Business, the students are introduced to programming in a topic-by-topic fashion. This means that the students would be actually able to program only after they learn at least a few topics, which could last several weeks. By this time, the students have already developed a fear of programming and are getting lost in information overload.

In their study, [11] highlighted, among other things, that access to a computer science degree programme is increased when the program's content, such as programming, is positioned as learnable and interesting. The project MBAD created a concept that promotes learning a programming language from a creative perspective. The concept developed in this project puts creativity and business application in the foreground. It infers the fundamental principles of the programming language, following the approach "programming is modelling" and "programming is design". Figure 2 shows an overview of all the steps in the MBAD approach.

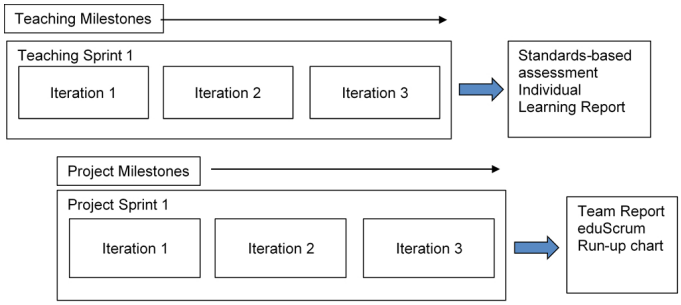

Figure 3: Teaching in Springs

4.1 Teaching in sprints

The agile approach to teaching is a very effective way to introduce students to programming. By breaking down the concepts into small, manageable iterations, students can understand the material more easily and retain more information. Additionally, by focusing on the business relevance of each concept and applying a project-based learning approach, students are motivated to learn and can see the practical applications of what they are learning. This approach is also beneficial for those with high programming skills, as they can progress at their own pace and still get enough challenges.

It is intended that the lecturers provide use cases for projects at the beginning of the semester (e.g., build a web shop or reservation platform for event tickets) and that the students work in groups on the implementation, which is also achieved using project sprints (see Figure 3). The students start by creating a storyboard for a given use case and identify several user stories. User stories begin with implementing very basic concepts (i.e. login, screen for entering basic data, . . .) and gradually become more difficult from sprint to sprint as additional aspects are added. For each sprint,

Table 1: Sprint learning goals mapped to Bloom's Taxonomy

	Knowledge	Understand	Apply
Requirements Engineering	What is a use case, actors, user story	how they are related to each other	Students can identify main requirements of the use case at the most abstract level Students are able to describe a user story at the most abstract level Students are able to visualize the use case as a process at an abstract level
Conceptual Modelling	Identify domain concepts from the user story such as data structure and behavior		
Object Oriented Concepts	Students know what is a class, object, attributes, methods	Students are able to relate the requirements to the classes and objects	Students are able to interprete the classes and objects as a class diagram
Programming	Students know the concepts of variables, primitive datatypes, classes, objects, input and output in a programming language e.g. Java/Python Students know the syntax and semantics of the chosen programming language	Students are able to relate programming concepts to the requirements of the use case Students understand the basic constructs of programming like conditions (and loops mybe interation 2) Students understand the concept of compiling and executing a program	Students are able to program the classes and interaction using a programming language e.g. Java/Python Students can create a project in a programming environment like Eclipse/IntelliJ/VSC Students can fix the errors in the program based on the compilation messages

learning goals are defined following Bloom's taxonomy. This approach "programming is modeling" allows for four different pillars to be considered when determining sprint goals: (1) Requirements Engineering; (2) Conceptual Modelling; (3) Object Oriented Concepts; and finally, (4) Programming itself. Initially, the problem and the current situation are to be understood, and then the most important concepts are to be identified before designing a solution that will be transformed into software code to achieve these goals. To accomplish that, different methods and techniques from these four pillars are required. Table 1 shows the defined learning goals for one of the sprints.

4.2 MBAD in action

Programming is regarded as a modelling task where a computer program is an executable model. As the first step in the iteration, the requirements are captured in the agile approach as a user story. As an example for a use case "hotel reservation platform" a user story in the first iteration could be "As a customer, I want to reserve one room for two people from 1st of April to 10th of April in a hotel". The steps are then modelled as a process, the conditions and involved data objects in the process flow are identified (see Figure 2).

Subsequently, the data and classes are mapped to data objects identified in the process flow as a class diagram (see Figure 2). At this stage, the object-oriented programming construct and variables, conditions, and loops are introduced. Based on the user story and process model, the attributes, as well as the operations, can be identified. It is evident that the number of people, the number of rooms, and the arrival and departure dates are attributes of a reservation. These can be found as class attributes in our class diagram. The prerequisite to checking a reservation is an application that manages bookings. For booking requests, different operations need to be executed. The operations are services ("methods") the class provides.

In the next step, the classes, class attributes, and class operations are transferred into the code. The final step in Figure 2 shows how Java code was derived from the class diagram. Thus, a connection to the code in any chosen programming language can be established based on the user story, process model, and class diagram.

After one user story is implemented, the sprint review and the retrospective define the end of the sprint. The next iteration starts by extending the case with an additional programming aspect, which is again based on a new and extended user story. The iterative design combines immediate feedback from the teachers because each iteration leads to a functioning artifact.

4.3 Benefits of MBAD approach

Teaching programming using the MBAD approach has two main advantages:

Instead of regarding programming as a bottom-up technical task, it is taught as a top-down design task. This offers the opportunity to attract students with limited technical background and introduce algorithmic thinking independent of a particular programming language.

- 1 Programming is considered as a modelling task where a computer program is an executable model. The model-based approach is a realization of the concept of model-driven software engineering for teaching purposes. It enables students to identify the important concepts of a business application instead of dealing with details of a particular programming language. Students are trained to first understand and visualize this problem. This is an important skill as it narrows the distance between domain experts and software developers.
- 2 The ability to create models that computers can execute has many advantages. It allows for better communication between domain experts (who may need to become more familiar with programming) and developers, leading to an improved understanding of requirements. In addition, it makes it easier to visualize different possible solutions before one focuses on programming the solution, which can save time and effort in the long run. Finally, executable models can serve as documentation for future reference or maintenance purposes.

The innovative teaching and project methodology of MBAD has a limitation. For using this approach only lecturers are eligible who are familiar with all four pillars. This constraint can be addressed through focused training and coaching.

5 CONCLUSION

Teaching programming for undergraduate students is a challenging task. This research paper summarizes experiences from courses held at the FHNW University of Applied Sciences and Arts Northwestern Switzerland and describes the state of the art of research

in this domain. The authors outline a concept for a new didactical approach – entitled MBAD and based on eduScrum – where creativity, modelling and design are the main pillars. This new approach will be implemented in the upcoming Bachelor of Science "Business Artificial Intelligence" for teaching elementary software engineering skills in the first study year.

ACKNOWLEDGMENTS

Research reported in this publication was supported by the FHNW Hochschullehre 2025 under the project number TP3 MBAD.

REFERENCES

[1] Alphonce, C. and Ventura, P. 2002. Object Orientation in CS1-CS2 by Design. *SIGCSE Bull.* 34, 3 (2002), 70–74. DOI:https://doi.org/10.1145/637610.544437.

[2] Anslow, C. and Maurer, F. 2015. An experience report at teaching a group based agile software development project course. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Feb. 2015), 500–505.

[3] Caspersen, M.E. *et al.* 2008. Proceedings of the fourth International Workshop on Computing Education Research: 2008, Sydney, Australia, September 6-7, 2008. ACM Press.

[4] eduScrum: 2022. https://eduscrum.org/.

[5] IEEE Education Society and IEEE Computer Society *FIE Cincinnati 2019: 2019 conference proceedings.*

[6] Kropp, M. *et al.* 2016. Teaching agile collaboration skills in the classroom. *Proceedings - 2016 IEEE 29th Conference on Software Engineering Education and Training, CSEEandT 2016* (May 2016), 118–127.

[7] Lundqvist, K. *et al.* 2019. Interdisciplinary Agile Teaching. *2019 IEEE Frontiers in Education Conference (FIE)* (2019), 1–8.

[8] Nguyen, D. and Wong, S. Position paper for OOPSLA 2001 Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts OOP in Introductory CS: Better Students Through Abstraction.

[9] Nyborg, M. and Valente, A. 2021. An agile approach to teach introductory programming in the hybrid classroom. *Proceedings - IEEE 21st International Conference on Advanced Learning Technologies, ICALT 2021* (Jul. 2021), 40–41.

[10] Or-Bach, R. and Lavy, I. 2004. Cognitive activities of abstraction in object orientation. *ACM SIGCSE Bulletin.* (2004). DOI:https://doi.org/10.1145/1024338.1024378.

[11] Resch, D. *et al.* 2017. *Attraktivität von ICT-Berufen – Synthesebericht.*