Goal-oriented Models for Teaching and Understanding Data Structures

Xavier Franch[1][0000-0001-9733-8830] and Marcela Ruiz[2][0000-0002-0592-1779] ✉

[1]Universitat Politècnica de Catalunya, Barcelona, Spain
[2]Zürich University of Applied Sciences, Winterthur, Switzerland
franch@essi.upc.edu
marcela.ruiz@zhaw.ch

Abstract. Most computer science curricula include a compulsory course on data structures. Students are prone to memorise facts about data structures instead of understanding the essence of underlying concepts. This can be explained by the fact that learning the basics of each data structure, the difference with each other, and the adequacy of each of them to the most appropriate context of use, is far from trivial. This paper explores the idea of providing adequate levels of abstractions to describe data structures from an intentional point of view. Our hypothesis is that adopting a goal-oriented perspective could emphasise the main goals of each data structure, its qualities, and its relationships with the potential context of use. Following this hypothesis, in this paper we present the use of iStar2.0 to teach and understand data structures. We conducted a comparative quasi-experiment with undergraduate students to evaluate the effectiveness of the approach. Significant results show the great potential of goal modeling for teaching technical courses like data structures. We conclude this paper by reflecting on further teaching and conceptual modeling research to be conducted in this field.

Keywords: goal-oriented models; iStar2.0; *i**; data structures; software selection; comparative quasi-experiment.

1 Introduction

Data structures (DS for short) are a programming concept that emerged in the early 70s as standardized solutions to the need of storing and manipulating data elements according to some particular requirements [19]. In spite of the profound changes that the computing discipline has experienced since then, DS still play a crucial role in programming. Fundamental DS such as lists and hash tables are in use in a myriad of contexts, from classical problems such as compiler construction or networking [21], to emerging domains such as blockchain [17]. In addition, new DS emerge to respond to specific challenges, e.g., compact DS [31] for storing large quantities of data, or concurrent DS [25] used in programs running on server machines with hundreds of cores.

As a result, most (if not all) computer science curricula include a compulsory course on DS to ensure that students acquire the necessary knowledge and skills on the topic. However, teaching DS and conversely, from the students' point of view, learning the

This is the accepted version of a conference paper published in *Conceptual Modeling*.
The version of record is available at https://doi.org/10.1007/978-3-030-89022-3_19

basics of each DS, the difference with each other, and the adequacy of a DS to a context of use, is far from trivial. On the one hand, student difficulties with DS span over comprehension of recursive programs, analysis, identification, and implementation of basic DS [29, 38]. On the other hand, students are prone to memorize facts about DS instead of understanding the essence of the underlying concepts [38].

One possible solution to provide an adequate level of abstraction to the description of DS is to adopt an intentional viewpoint [37] for their teaching and study. This way, the emphasis is shifted towards understanding the main goals of every DS, what are their qualities, and relationships with their context of use.

Aligning with this vision, we explore in this paper the use of goal-oriented models [7] for describing DS. To this end, we propose an extension of the iStar2.0 language [6] as the basis for building these models and we evaluate the effectiveness of the approach through a quasi-experiment with students. We show that a group of students who studied DS specified with the iStar2.0 language is significantly more effective when analysing DS in contrast to a control group that studied DS without iStar2.0.

The rest of the paper is organized as follows. Section 2 describes the background of this research and related work. Section 3 states the problem context and the main research question of our research. Section 4 introduces iStarDust, an extension to iStar2.0 for DS. Section 5 describes the design and results from a controlled quasi-experiment conducted with a group of 2nd year computer science students of an algorithms and DS course. Finally, in Section 6 we summarise the main conclusions and future work.

2 Background and Related Work

2.1 DS as implementations of abstract data types

As commented above, DS emerged at the early 70s, at the same time as the related concept of abstract data type (ADT) [13]. ADTs provide a high-level specification of a DS, declaring its operations and the properties that they fulfil. ADTs are implemented using DS which remain hidden to the client of the ADT. In the context of programming with ADTs [24], two questions arise:

1) Which is the most appropriate ADT according to some functional characteristics? Typical ADTs are lists, mappings, graphs and trees.
2) What is the DS that best implements the chosen ADT according to some quality requirements? DS differ mainly in efficiency both in time (of their operations) and space (to store its elements).

Along time, there were a number of proposals either defining new languages with specific constructs for manipulating ADTs and DS, like CLU [23] and SETL [33], or annotation systems like NoFun [3]. Both types of proposals supported the declaration of main characteristics of DS (like their efficiency) and requirements from the program looking for the most appropriate implementation. While they provided some support to the problem of DS selection, their constructs were at the programming level and made their use cumbersome when a more lightweight approach suffices, e.g. to summarize the main characteristics of the DS in a teaching context. This motivates the use of a higher-level notation as *i**.

2.2 *i** as a conceptual tool for software analysis

The *i** language has been primarily used for strategic reasoning in the context of socio-technical systems that include human-related actors (people, organizations, etc.) [37]. However, a line of research has focused on using *i** as a conceptual tool for software analysis. The common characteristic of these works is the predominant role that software elements, represented as actors, play in the model, in contrast to human-related actors, which are less significant (or even are not represented in the models).

There are several proposals using *i** for software analysis. Some researchers propose *i** models as a representation of software architectures [14, 35]: software components are modelled as actors and their connections represent expectations from one component to another (e.g., a goal to be fulfilled, a file to be delivered, an API to be provided). In a similar vein, other works use *i** to model or reason about product lines [1], service-oriented systems [8] and systems in domains like IoT [1] or business intelligence [20]. These proposals made use of *i** constructs in a particular way (e.g., importance of *i** positions to represent components that cover different roles), and relied on softgoals to represent software quality (inside actors) or quality requirements (as dependencies).

In our paper, we are primarily interested in the use of *i** to support software package selection [11], considering DS as the software to be selected. We explored this objective in a short paper presented at the iStar'20 workshop [12]. The current paper extends this preliminary contribution in several directions. On the one hand, the proposal is formulated following a well-defined methodology for extending *i** that includes a systematic analysis of requirements and needs for the extension, and the inclusion of the new proposed constructs in the *i** metamodel. On the other hand, it includes a preliminary evaluation of the proposal by means of a controlled quasi-experiment.

2.3 Teaching DS

Students suffer from misconceptions during their learning process [4], whose existence needs to be identified and mitigated to continuously improve a subject matter. In the computing science discipline, several works have focused on misconceptions related to introductory programming [27], while for more specialized topics like DS, contributions are scarce, but indeed still exist, as enumerated below.

Some researchers have focused on particular DS, e.g., misconceptions on heaps [34] or in binary search trees and hash tables [18]. In our paper, we take a more general viewpoint and target DS in general, looking for a framework that can be customized or extended to any set of DS.

In 2018, Zingaro et al. [38] reported a multi-method empirical study about the difficulties that students have on understanding DS. After a round of think-aloud interviews to gather a series of questions to elicit barriers to understanding, the authors ran a final exam study session containing 7 questions for a Java-based CS2 course on DS. While the results of this paper are highly interesting, the study was explorative in nature and focuses on highlight students difficulties; in contrast, our paper seeks for a constructive solution proposing a concrete framework to improve students' DS understanding and effectiveness, and provides empirical data to elucidate potential benefits in helping students to achieve intended learning goals.

3 Problem Scope

The goal of this paper can be stated as: to **analyse** *i** **for the purpose of** *teaching and learning DS* **with respect to** *functional and non-functional requirements* **from the point of view of** *educators and students* **in the context of** *software development.* In order to make this goal more concrete, we have taken the following decisions:

- We select iStar2.0 [6] as the language used to formulate *i** models. The reason is that iStar2.0 has been proposed as a standard di facto in the *i** community with the purpose of having an agreed definition of the language core constructs.
- We follow the PRISE method [15] to extend iStar2.0 with constructs that fit the goal. For space reasons, in this paper the reporting of the method application is kept at the level of main steps (sub-processes) and focusing on the essential tasks and artifacts.
- We select as DS a typical subset that is taught in a DS introductory course in a computer science syllabus, namely sequences and tables, well-documented in a number of classical textbooks [5, 10].

From this goal and these decisions, we derive the following research questions:

RQ1: What extensions can be applied over iStar 2.0 to make it suitable to describe DS? Following PRISE [15], we will identify the constructs needed to make iStar2.0 suitable to the DS domain. We will identify the needs, conceptualize the solution, and extend the iStar2.0 metamodel [6].

RQ2: When the subjects study DS specified with iStar2.0, is their performance in describing DS affected? To answer this question, we compare completeness and validity of described DS by subjects that have studied DS specified with iStar; to a control group that has studied DS specified in the traditional way (without iStar2.0).

4 iStarDust: iStar2.0 for DS

In this section, we show the application of PRISE to the domain of DS in order to answer RQ1. We call iStarDust (arguably, quasi-acronym of *iStar2.0 for DS*) the resulting language extension. We focus on the first three PRISE steps, because: Step 4 is evaluation of the extension, which deserves a full section in this paper (Section 5); Step 5 provides a sense of iteration which is in fact embedded in our way of working; Step 6 consists of reporting and publicizing the results, which turns out to be this very paper.

4.1 Analyse the need for an extension

PRISE first step checks whether an extension of iStar2.0 is really needed. It may certainly happen that the iStar2.0 constructs suffice to represent the concepts that we need. We list below the requirements that make iStar2.0 fall short to support DS description:

- **Req1**. *The language should allow relating specification and implementation of DS.* As explained in Section 2, we distinguish among the specification of a DS (i.e., the ADT it represents) and the implementation of a DS (i.e., the code used to implement the operations). We need to keep this relation explicit in order to accurately describe

the DS. The current actors, and association among actors, that iStar2.0 includes, do not allow representing this concept.

- **Req2**. *The language should allow to encapsulate intentional descriptions*. DS are modular in nature. They become integrated into software programs as building blocks. In fact, they are normally reused from some existing software library. The original iStar2.0 does not provide such encapsulation mechanisms, beyond the notion of actor, which is not enough for our purposes.
- **Req3**. *The language should allow representing similarities among DS*. Typical DS are organized following a hierarchical structure to keep similarities among them. For instance, sequences are a category of DS, including stacks, queues and lists. At their turn, lists can be sub-categorized (at least) in one-direction lists and bi-directional lists (depending on the type of traversals they support). Likewise, there are several variations of hash tables that share a number of commonalities. iStar2.0 does not include a concept for representing such hierarchies.

These requirements yield to the concepts to be included in iStarDust extending iStar2.0:

- **Conc1**. Specification and implementation of a DS
- **Conc2**. Relationship among specification and implementation of a DS
- **Conc3**. Encapsulation of specifications and implementations of DS
- **Conc4**. Hierarchical organization of specifications & implementations of similar DS

4.2 Describe concepts of the iStar2.0 extension

Following PRISE recommendations, we first searched in the available *i** literature for constructs already proposed, and found **Const1** and **Const3** (see below). In addition, we proposed an additional construct (**Const2**) to cover a missing concept. **Fig. 1** shows the mapping between concepts and constructs.

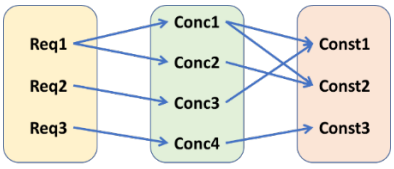

Fig. 1. From requirements to constructs

- **Const1**. The concept of *module* for representing encapsulation. Modules have been proposed as a means to encapsulate a set of actors and dependencies [28]. With respect to this proposal [28], we restrict modules to contain only one actor that will represent either the specification or the implementation of a DS. We prefer not to distinguish explicitly the two types of modules to keep the number of new constructs as low as possible. Every module will have open incoming or outcoming dependencies to/from its enclosed actors, representing the intentions that the DS specification or implementation offers/requires. These open dependencies become complete when the DS is inserted in a particular context. **Fig. 2** shows three examples: two modules encapsulating a specification (*Stack*, *Mapping*) and one module encapsulating an implementation (*Hash Table*). While *Stack* does not require anything from its context of use, *Mapping* requires that the stored elements have the concept of *Key* (to provide individual element look-up). The *Hash Table* implementation offers fast look-up and requires to know the approximate number of elements to store (to size the table) plus a hashing function.

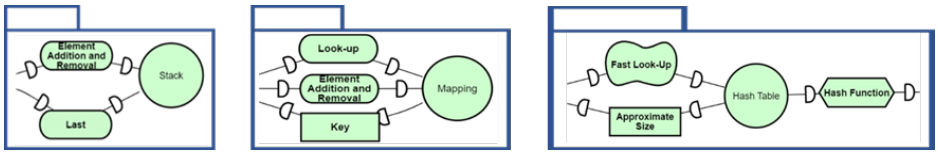

Fig. 2. Modules for the *Stack* and *Mapping* specifications, and *Hash Table* implementation[1,2]

- **Const2**. A type of actor link, *implements*, to connect an implementation to the corresponding specification. A specification may (and usually, will) have several implementations, while the opposite is false. As a side effect, the link allows to clearly identify when a module defines an implementation or a specification, considering whether it is the source or the target of an *implements* link. **Fig. 3** shows the link from the *Hash Table* implementation to the *Mapping* specification presented above.

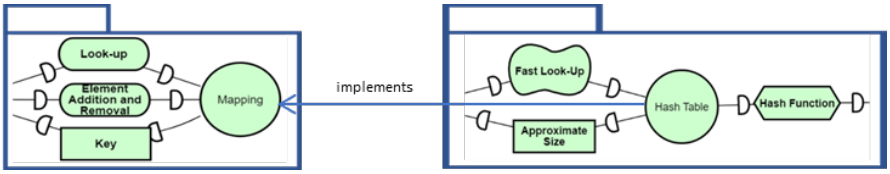

Fig. 3. The *Hash Table* implementation *implements* the *Mapping* specification

- **Const3**. The concept of specialization for representing the requested hierarchies. Specialization makes a good fit to represent commonalities and differences. We adopt López et al.'s framework to make precise the effects of the iStar2.0 *is-a* actor link at the level of intentional elements [26], distinguishing three types of specializations: extension, reinforcement and cancellation. **Fig. 4** shows an example that puts the *Mapping* specification presented above into context. As root of the hierarchy, the *Function* specification represents the family of DS that support individual operations (addition, removal and access). *Mapping* and *Set* are two particular DS belonging to this family. They only specialize through reinforcement the access operation to better reflect the differences: while sets provide membership only, mapping allows looking up elements. These specifications can be further specialized, as we show with *Mathematical Set*, which extends *Set* with *Union* and *Intersection*.

4.3 Develop iStar2.0 extension

In this step, the original iStar2.0 metamodel [6] is enriched with the new constructs, also including integrity constraints needed to ensure the correctness of the models.

Fig. 5 shows the excerpt of the iStarDust metamodel which contains changes with respect to the iStar2.0 metamodel. **Table 1** lists the corresponding integrity constraints and one derivation rule. We observe:

---

[1] All diagrams have been drawn with the piStar tool, https://www.cin.ufpe.br/~jhcp/pistar/, and manually modified to add constructs not included in iStar 2.0.

[2] For clarity of the paper, the examples already present the concrete syntax proposed for the constructs, which PRISE considers as part of the next step (developing an iStar2.0 extension).

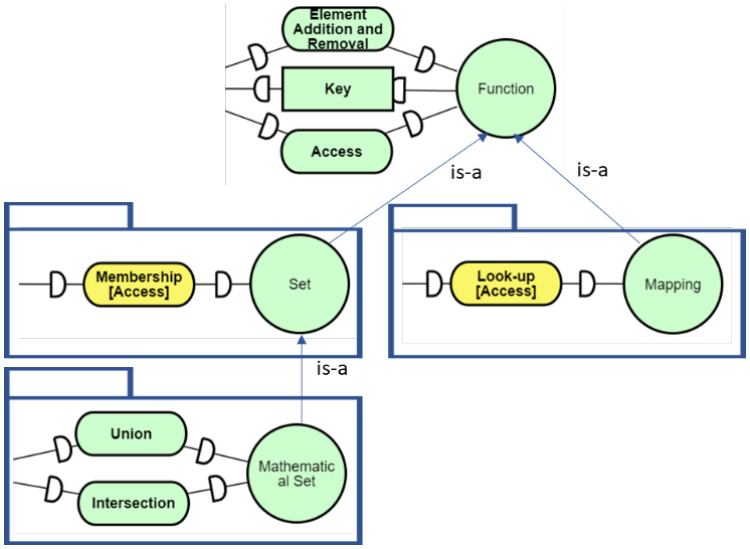

Fig. 4. Specialization for functions[3]

- The new *Module* class contains one actor and a number of open dependencies in and out. For clarity, we add a derived attribute stating whether the module corresponds to a specification or an implementation (**DR**). An intentional element cannot participate in more than one association (**IC1**), and specifications cannot have qualities as dependencies (**IC2**) (qualities should appear only at the level of implementations).
- The new *implements* association among actors is required to bind one implementation to one specification (**IC3**, **IC4**).
- Last, we add the notion of *reinforcement* at the intentional element level. **IC5** ensures that, given a reinforcement at the level of intentional elements, there is a corresponding *is-a* relationship at the level of actors, while **IC6** states that the types of intentional elements match according to [26] (e.g., we cannot reinforce a task into a goal).

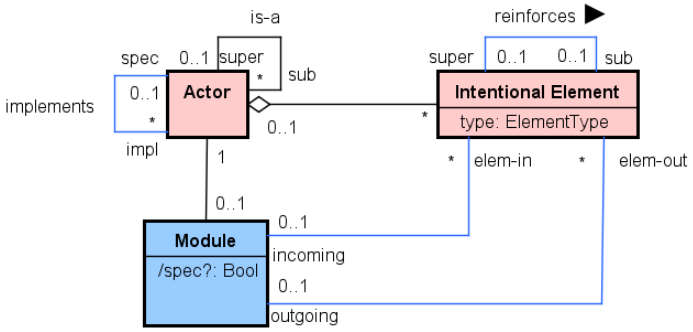

Fig. 5. iStarDust metamodel (elements added to the iStar2.0 metamodel appear in blue colour)

[3] We fill with yellow colour the elements affected by the specialization.

Table 1. iStarDust integrity constraints required by the added constructs

ID	Derivation Rules and Integrity Constraints
DR	**context** Module **def**: spec? = not actor.implements <> null
IC1	**context** IntentionalElement **inv**: actor->size() + incoming->size() + outgoing->size() = 1
IC2	**context** Module **inv**: spec? **implies** (not elem-in->exists(type=quality) and not elem-out->exists(type=quality))
IC3	**context** Actor **inv**: spec <> null **implies** actor.module.spec? = false
IC4	**context** Actor **inv**: impl->notEmpty() **implies** actor.module.spec? = true
IC5	**context** IntentionalElement **inv**: super <> null **implies** (incoming.actor.super = super.incoming.actor or outgoing.actor.super = super.outgoing.actor)
IC6	**context** IntentionalElement **inv**: super <> null and type <> super.type **implies** ((type = task **or** type = resource) **implies** (super.type = goal **or** super.type = quality) **and** (type = goal **implies** super.type = quality))

5 Validation by means of a comparative quasi-experiment

We have performed a comparative quasi-experiment to measure undergraduate students' performance in describing DS, after being exposed to DS described with the iStar2.0 language. This quasi-experiment has been designed according to Wohlin et al. [36], and it is reported according to Jedlitschka and Pfahl [16].

5.1 Experimental design

The experimental goal, according to the Goal/Question/Metric template [2] is to **analyse** DS descriptions **for the purpose of** *understanding whether iStar2.0 could help to describe DS in a more structured form,* **with respect to** *its effectiveness* **from the point of view of** *computer science students and teachers* **in the context of** *a bachelor course on algorithms and DS at the Zürich University of Applied Sciences in Switzerland*. The main research question of this experiment is formulated as RQ2 presented in Section 3.

Experimental subjects. The experiment was conducted in the academic year 2020-2021 (from September 2020 until January 2021) within the bachelor-level course of Algorithms and DS (ADS) offered at the Zürich University of Applied Sciences[4]. The subjects were 61 second year students of the computer science curriculum enrolled in two different groups with distinct schedules. Both groups have students with experience in industry. In general, some students are currently working in industry (79%), and none of them had been in contact with the *i** framework or used iStar2.0 for describing DS. The course planning was not updated to incorporate the experimental set-up, still maintaining the original course objectives. However, the content presented to one of the two groups was updated adding an iStar2.0 description to some of the DS taught. The group who received materials of DS with iStar was considered as experimental group. The subjects executed the experimental task as part of the course's end-of-semester exam. Properly speaking, we have performed a quasi-experiment because the subjects were not sampled randomly across the population; however, this is typical in software engineering experiments [32].

[4] Course description available at https://tinyurl.com/3pdnb3xa

Variables. We consider one independent variable:

- **DS specification.** The way DS are described by the subjects. This variable has two values:
  - DS specified with iStar2.0, as defined in Section 4.
  - DS specified without iStar2.0, serving the purpose of a control group.

We consider the following dependent variables, which are expected to be influenced to some extent by the independent variable. We have adapted the Method Evaluation Model (MEM) to structure the dependent variables of this experiment [30]. In this way, the effectiveness of iStar2.0 to describe DS is measured by evaluating subjects' performance regarding completeness and validity of subjects' described DS [22] (see Fig. 6).

- **DS completeness**. The degree to which all the intrinsic characteristics that should be described regarding a certain DS (because they explain the meaning of a DS or its differences with another DS) are actually mentioned by the subjects. To facilitate this calculation, the researchers consider a reference solution containing the minimum indispensable description.
- **DS validity**. The degree to which the characteristics of a certain DS are described by the subjects in the right way. Acting as reviewers, the researchers identify properly or wrongly described characteristics based on a reference solution, and then discuss them until they agree on the verdict.

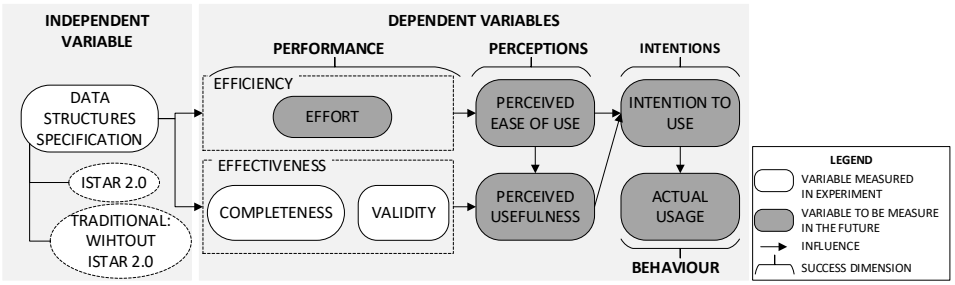

Fig. 6. Variables structure according to [30], and envisioned variables to be measured in future empirical evaluations.

For this experiment we consider two types of possible analysis of DS: **type A)**, description of differences between two given DS; **type B)**, description of the most important requirements or operations of a given DS. We consider these two possibilities to formulate below the hypotheses of our study and experimental objects.

Hypotheses. We define null hypotheses (represented by a 0 in the subscript) that correspond to the absence of an impact of the independent variables on the dependent variables. Alternative hypotheses (represented by a 1 in the subscript, e.g., $H1_1$ is the alternative hypothesis to $H1_0$) suppose the existence of such an impact. Alternative hypotheses correspond to our expectations: DS specified with iStar2.0 will have a positive impact on the dependent variables (For the sake of brevity, alternative hypotheses are omitted).

Table 2. Hypothesis description

Null Hypothesis id	Statement: DS specified with iStar2.0 does not influence the…
$H1_0$	…completeness of identified differences between two given DS
$H2_0$	…completeness of described requirements of a given DS
$H3_0$	… validity of stated characteristics of a given DS when subjects intent to draw differences
$H4_0$	… validity of described characteristics of a given DS

5.2 Procedure and data analysis[5]

As part of the experimental task, we provided to the subjects with three different types of input questions to increase the external validity of the experiment based on questions' objective (see Table 3).

Table 3. Description of the experimental objects

Question type	Objective of the question	Example
A	Description of the main differences and similarities between two DS.	Which are the differences and similarities between stacks and lists in ADS?
B	Description of the main requirements (i.e., operations) of a given DS.	What is a stack, and which are the basic operations?
C	Control question about DS without associated iStar 2.0 language description.	What is the main difference between a singly and doubly linked list?

We created four equivalent versions for each question type, changing the type of DS (e.g., queues, lists, etc.) but maintaining the overall objective of the question so that resulting answers are isomorphic. The design, according to [36] is a "one factor with two treatments", where the factor is the DS specification and treatments are the DS specified with iStar2.0 or in the traditional way (without iStar 2.0). The experimental procedure is presented in Fig. 7. The control group received the ADS content as it was designed for the course. In contrast, the experimental group received slides with the prescribed ADS content including DS exemplified by means of iStar2.0. Before the exam, the students answered some learning questions, and the experimental group provided qualitative feedback regarding the use of iStar2.0. The experimental task was executed in the context of the oral end-of-semester exam. During the exam, each subject randomly selected a set of questions and received the experimental objects as presented in Table 3. Two teachers assessed the subjects' performance, gave a grade, and took notes of the answers.

Data Analysis. For this type of experimental design, the most common analysis is to compare the means of the dependent variable for each treatment [36].

[5] To facilitate further replication of this controlled experiment, the material describing the experimental objects, sample slides of DS in iStar provided to experimental subjects, set of questions used during the execution of the experimental task, experimental design, and output from statistical analysis can be found at https://drive.switch.ch/index.php/s/6IuFjGtONSWV2bB

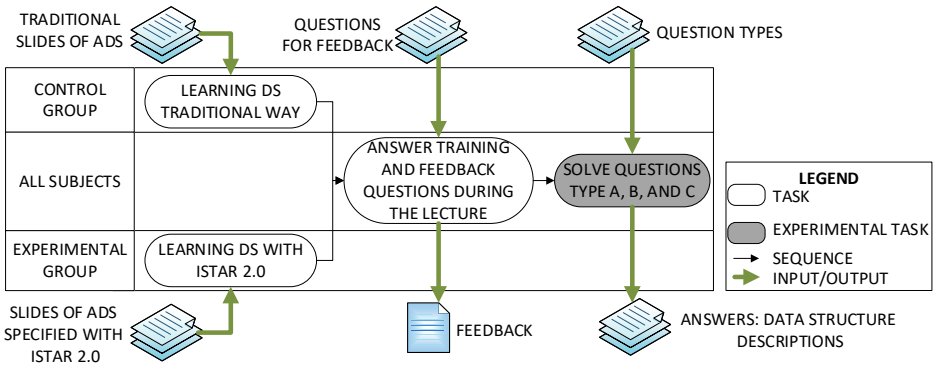

Fig. 7. Experimental procedure

DS completeness. The results of descriptive statistics for the control group show completeness average of 12% for question type A, 96% for question type B, and 89% for question type C; vs 72% for question type A, 91% for type B, and 84% for type C of the experimental group. The ANOVA Test was applied, and we verified a **significant difference** ($p<0,05$) between the two groups for $H1_0$. Thus, the alternative hypothesis $H1_1$ is corroborated, demonstrating that iStar2.0 does influence the completeness of described DS when subjects are comparing them. On the other hand, no significant difference was observed for question type B. As a result, our hypothesis $H2_1$ is not corroborated and we conclude that iStar2.0 does not influence the completeness of described DS.

DS validity. The results of descriptive statistics show an average of 48% of valid answers for question type A, 84% for questions type B, and 90% for our control question type C for the control group, versus 91% for questions type A, 88% for type B, and 81% for type C for the experimental group. By applying the ANOVA test, we observe there is **significant difference** for question type A ($p < 0,05$) between the two groups. Therefore, the null hypothesis $H3_0$ is rejected and the alternative hypothesis $H3_1$ is corroborated demonstrating that iStar2.0 does influence the validity of described DS when the subjects are analysing differences. Nevertheless, no significant difference is observed for questions type B. As a result, our hypothesis $H4_1$ is not corroborated and we conclude that iStar2.0 does not influence the validity of described DS.

Results to the control question type C did not show any significant difference between the two groups when answering questions without associated iStar2.0 DS. This minimises the internal validity threat regarding the maturity and experience of the subjects. Further research needs to be conducted for question type B.

5.3 Analysis of the threats to the validity of the results

Internal validity. The subjects *matured* their competence in DS through the semester, which can have had a positive impact in their performance. To minimise this threat, we

conceived an experimental design involving both control and experimental groups. The *selection-maturation* threat could indicate that the experimental group could have learnt DS skills faster and better than the control group. To minimise this threat, we have included a control question in our experiment that is not related to a DS we had introduced with iStar2.0. *Social threats* are unlikely to happen because control and experimental groups are totally independent from each other through their course of studies. Our experimental design assures that both control and experimental groups do not suffer disadvantages by keeping the original course's plan and objectives for both groups.

External validity. Since this experiment is conceived to be conducted in an educational context, the subjects have been properly selected. However, we acknowledge limits in the *generalisation of the experiment results*. We have performed a quasi-experiment since our subjects were not randomly selected from the target population. Yet the experiment was conducted in a real educational setting. We acknowledge that we need to conduct further experiments involving groups of students from different countries and take into consideration variations in the computer science curricula. The results from this study just apply to the set of DS selected for this quasi-experiment. As explained below in the construct validity analysis, we plan to perform deep analysis on the impact of iStarDust for each data type. Moreover, we plan to investigate the extent iStarDust affects overall subjects' performance.

Construct validity. We minimised the *threat of inadequate preoperational explication of constructs* by means of using a widely accepted model for method evaluation. On other matters, the fact that we compare a new method for describing DS with traditional approaches may seem an inadequate comparison of treatments. Nevertheless, our objective is to advance current DS teaching practices by involving conceptual modeling methods and techniques. A potential threat regarding the equivalence of questions' sets given to the subjects' groups could pose a *mono-method bias threat*. The questions' sets evaluate the extent a subject can identify similarities, differences, and requirements for given DS, which are wide-known difficulties when understanding DS. Since our subjects did not answer the questions at the same time, we opted for not giving the same questions to both groups. We plan to perform further experiments where each data type is analysed in a controlled experiment. This experiment presents valuable results towards a new generation of teaching content for traditional computer science courses.

6 Conclusions and Future Work

This paper presents our research efforts in effectively implementing goal-oriented perspectives for the teaching and understanding of DS concepts. Our research is motivated by the need to augment how DS are taught to mitigate students' difficulties when using DS, and their need to simply memorise DS characteristics to excel in their studies [38]. Our hypothesis is that by teaching DS with a goal-oriented perspective, students will be able to experience a better understanding of the main goals of every DS, what their qualities are, and relationships with their context of use. To test this hypothesis, we follow the PRISE method [15] to extend iStar2.0 with constructs that facilitate teaching

and learning DS with respect to functional and non-functional requirements. In this paper we present iStarDust: iStar 2.0 for DS, and report on an initial validation regarding students' effectiveness by means of a controlled experiment. The results from the controlled experiment have shown that iStar 2.0 can improve students' effectiveness in terms of completeness and validity of described DS. However, further experiments must be conducted to be able to generalise the results to a wider population.

As part of our future endeavours, we plan to conduct further controlled experiments to observe students' effectiveness, efficiency, perceptions, and intentions when using iStarDust for learning DS (see Fig. 6). Despite having students as a target population to validate iStarDust is considered a valid approach in software engineering [9], we also envision to investigate to what extent iStarDust can also be adopted by professionals. We also envision to perform a focused analysis on the impact of using iStarDust for each data type concerning wide-known learning difficulties. This would allow us to observe the benefits of teaching each data type and the impact in students' effectiveness when learning DS. Last, to facilitate the adoption of iStarDust by both instructors and students, we plan to develop a plug-in for the piStar tool to support the specification of iStar models with the extensions provided by iStarDust.

Acknowledgements

This work has been partially supported by the by the DOGO4ML Spanish research project (ref. PID2020-117191RB-I00), the Digitalization Initiative of the Canton of Zürich (DIZH), and ZHAW Digital.

References

1. Ayala, I., Amor, M., Horcas, J.M., Fuentes, L.: A Goal-driven Software Product Line Approach for Evolving Multi-agent Systems in the Internet of Things. *Knowledge-Based Systems* 184, 2019: 104883.
2. Basili, V., Caldiera, C., Rombach, H.D.: Goal Question Metric Paradigm. *Encyclopedia of Software Engineering* (Marciniak, J.J., ed.), John Wiley Sons. 1, 528–532 (1994).
3. Botella, P., Burgués, X., *et al.*: Modeling Non-Functional Requirements. JIRA 2001.
4. Confrey, J.: A Review of the Research on Student Conceptions in Mathematics, Science, and Programming. *Review of Research in Education*, 16(1), 1990: 3–56.
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: *Introduction to Algorithms*. The MIT Press, 1990.
6. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. arXiv preprint arXiv:1605.07767, 2016.
7. Dardenne, A., Lamsweerde, A. van, Fickas, S.: Goal-Directed Requirements Acquisition. *Science of Computer Programming* 20(1-2), 1993: 3-50.
8. Estrada, H., Rebollar, A.M., Pastor, O., Mylopoulos, J.: An Empirical Evaluation of the *i** Framework in a Model-Based Software Generation Environment. CAiSE 2006: 513-527.
9. Falessi, D., Juristo, N., *et al*. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23, 452–489 (2018).
10. Franch, X.: *Estructuras de Datos: Especificación, Diseño e Implementación*. Ed. UPC, 1993.
11. Carvallo, J.P., Franch, X., Quer, C.: Determining Criteria for Selecting Software Components: Lessons Learned. *IEEE Software* 24(3): 84-94 (2007).

12. Franch, X.: Using *i** to Describe Data Structures. iStar Workshop 2020: 59-64.
13. Goguen, J.A., Thatcher, J.W., Wagner, E.G.: An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. In *Current Trends in Programming Methodology*, *Vol. IV*, Prentice-Hall, 1978.
14. Grau, G., Franch, X.: On the Adequacy of *i** Models for Representing and Analyzing Software Architectures. ER Workshops 2007: 296-305.
15. Gonçalves, G., Araujo, J., Castro, J.: PRISE: A Process to support iStar Extensions. *Journal of Systems and Software* 168, 2020: 110649.
16. Jedlitschka, A., Pfahl, D.: Reporting guidelines for controlled experiments in software engineering. ESEM 2005.
17. Junhui, W., Tuolei, W., Yusheng, W., Jie, C., Kaiyan, L., Huiping, S.: Improved Blockchain Commodity Traceability System using Distributed Hash Table. CAC 2020: 1419-1424.
18. Karpierz, K., Wolfman, S.A.: Misconceptions and Concept Inventory Questions for Binary Search Trees and Hash Tables. SIGCSE 2014: 109–114.
19. Knuth, D.E.: *The Art of Computer Programming*, Vol. 3. Addison-Wesley, 1973.
20. Lavalle, A., Maté, A., Trujillo, J., Rizzi, S.: Visualization Requirements for Business Intelligence Analytics: A Goal-Based, Iterative Framework. RE 2019: 109-119.
21. Le Scouarnec, N.: Cuckoo++ Hash Tables: High-Performance Hash Tables for Networking Applications. ANCS 2018: 41-54.
22. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. *IEEE Software* 11(2): 42-49 (1994).
23. Liskov, B.H., Guttag, J.V. *Abstraction and Specification in Program Development*. MIT, 1986.
24. Liskov, B., Zilles, S.: Programming with Abstract Data Types. *ACM SIGPLAN Notices* 9(4): 50–59, 1974.
25. Liu, Z., Calciu, I., Herlihy, M., Mutlu, O.: Concurrent Data Structures for Near-Memory Computing. SPAA 2017: 235–245.
26. López, L., Franch, X., Marco, J.: Specialization in the iStar2.0 Language. *IEEE Access* 7, 2019.
27. Ma, L., Ferguson, J., Roper, M., Wood, M.: Investigating and Improving the Models of Programming Concepts held by Novice Programmers. *Comp. Science Educ.*, 21(1), 2011: 57–80.
28. Maté, A., Trujillo, J., Franch, X.: Adding Semantic Modules to improve Goal-Oriented Analysis of Data Warehouses using I-star. *Journal of Systems and Software* 88, 2014.
29. McAuley, R., Hanks, B., *et al*. Recursion vs. Iteration: An Empirical Study of Comprehension Revisited. SIGSE 2015: 350-355.
30. Moody, D.: The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods. ECIS 2003: 1327-1336.
31. Navarro, G.: *Compact Data Structures*. Cambridge University Press, 2016
32. Robson, C.: *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley-Blackwell (2002).
33. Schwartz, J.T., Dewar, R.B.K., Dubinsky, E., Schonberg, E.: *Programming with Sets: An Introduction to SETL*. Springer, 1986.
34. Seppälä, O., Malmi, L., Korhonen, A.: Observations on Student Misconceptions—A Case Study of the Build-Heap Algorithm. *Computer Science Education*, 16(3), 2006: 241–255.
35. Soares, M., Pimentel, J., *et al.*: Automatic Generation of Architectural Models from Goal Models. SEKE 2012: 444-447.
36. Wohlin, C., Runeson, P. *et al.*: Experimentation in Software Engineering. Springer (2012).
37. Yu, E.: Modeling Organisations for Information Systems Requirements Engineering. ISRE 1993: 34-41.
38. Zingaro, D., Taylor, C. *et al.*: Identifying Student Difficulties with Basic Data Structures. ICER 2018: 169–177.