International Journal of Engineering Education Vol. 32, No. 2(B), pp. 1036–1052, 2016
Printed in Great Britain
0949-149X/91 $3.00+0.00
© 2016 TEMPUS Publications.

Design Patterns Combination for Agile Development of Teaching/Learning Haptic Simulators*

CAMINO FERNÁNDEZ, GONZALO ESTEBAN, FRANCISCO J. RODRIGUEZ-LERA and FRANCISCO RODRÍGUEZ-SEDANO
Robotics Group, Department of Mechanical, Computer Science and Aerospace Engineering, University of León, Campus de Vegazana, S/N, 24071-León, Spain. E-mail: {camino.fernandez, gestc, fjrodl, francisco.sedano}@unileon.es

DAVID DÍEZ
Computer Science and Engineering Department, University Carlos III of Madrid, Avda, Universidad, 20, 28911-Leganés, Madrid, Spain. E-mail: ddiez@inf.uc3m.es

Software engineering courses usually face a situation where a reasonable size application has to be developed within a semester period. But, on the one hand, if students begin from scratch and start with the knowledge acquisition and requirements phases, they usually do not get to the deployment one. And, on the other hand, developing quality software should go on being the main focus of a software engineering course. Our proposal is the development of a haptic simulator as a teaching/learning tool for this purpose using the SHULE framework and following a Scrum development methodology. The core of this framework includes a combination of design patterns that also guide the development of the whole simulator. This approach has been used in the development of a cataract surgery simulator as a teaching/learning tool, and the experience is shown as an example of the general Scrum development that is presented.

Keywords: design patterns; agile development; teaching/learning environment; haptic simulator

1. Introduction

Design patterns are not an easy subject for students to learn. When they are presented in class, students can get the general idea of their design but it is not until they code them in a real application when they really know how patterns work. These elements of reusable object-oriented software, as they were defined by the GoF [1], have become an essential ingredient for any quality software development. But in order to truly understand their importance, they have to be used in more than a small code example. They do not just affect the software design itself improving flexibility and reuse, but also the way that software can be developed in terms of the division of tasks. This is a key point when facing a software development using any methodology, and specially when using Agile methodologies.

Although Scrum cannot be considered as a methodology for analysis or design, such as The Unified Software Development Process [2], it is a methodology for work planning and management. This Agile development framework defines the process, rules, practices, roles and artifacts necessary in order to increase productivity of development teams, based on an iterative and incremental cycle of software creation and development [3].

The first systematic review on empirical studies of Agile software development [4] found 36 empirical studies published before 2005, 9 of them performed at university courses. The project duration for these ones varied from 3 weeks to 1 year, and the team size was between 4 and 6, only one of the studies showed a team of 16. They usually developed a research prototype and none of them used a framework as the starting point. They concluded that more studies were needed, especially about Scrum, because the studies investigated XP almost exclusively. In 2012, another systematic review was conducted to capture the status of combining agility with global software engineering in studies published between 1999 and 2009 [5]. The majority of the existing research was industrial experience reports and the paper was focused on knowing the reasons for successful empirical cases reported.

Regarding teaching in higher education, no agile review has been found, but a review on empirical studies of pair programming was conducted by Salleh et al. [6]. They concluded that students' skill level was the factor that affected effectiveness the most.

Software development does not always show successful results. Sometimes projects are delayed or even abandoned. A study performed by Chow and Dac-Buu [7] of the use of agile methodologies in the literature tried to identify the critical success factors that help software development projects using agile methods to succeed. A web-based survey was conducted from 109 agile software projects from 25 countries and multiple regression analysis was performed. They found that out of 48 possible factors only 10 were supported by evidence.

* Accepted 1 September 2015.

And only three of them could be considered as critical ones: a high-caliber team, a correct delivery strategy and a proper practice of Agile software engineering techniques. Besides focusing on the last one, our course is also devoted to teamwork. Transitioning from individual work to self-managing teams requires a reorientation by developers. In a nine-month field study of professional developers in a Scrum team performed by Moe et al. [8], highly specialized skills and a corresponding division of work was found to be the most important barrier for achieving effective teamwork.

The application of Scrum implies a continuous process of testing and learning, where rules and practices can be dynamically adapted in order to ensure an optimal use of available resources in order to increase efficiency. This process fits perfectly a software engineering course where requirements may not be clearly stated at the beginning, especially because students are new to the problem environment, and small improvements can be defined through the process in order to add value to the stakeholders.

Our proposal for using design patterns in a Scrum development takes our framework, SHULE, as its starting point [9]. SHULE (*Simulators for Haptic Use in Learning Environments*) is a framework for building haptic simulators as teaching/learning tools. The core of SHULE combines several design patterns for offering a flexible and reusable architecture for developing haptic simulators. The haptic layer is provided by HBOgre (*Haptic-Bullet-Ogre*), a library that abstracts the low level details of haptic devices and haptic interaction and offers a simplified way to create virtual scenes inside the Ogre 3D engine.

The experience we are showing in this work uses SHULE and HBOgre to develop a cataract surgery haptic simulator. For a Scrum development, the lead user role is vital. In our case, a veterinary surgery professor played that role. It is very important to get a lead user from outside the software world because that is the way students get a more real experience. Users usually do not speak the same technical language as software engineers do. Students have to practice to try to communicate their ideas and work to a non-technical audience.

Developing haptic simulators must be linked to tasks where the sense of touch is part of the experience being transmitted, especially if these simulators are to be used as teaching/learning tools. In this sense, performing a surgical procedure lays on the surgeon's sensory information, that is, the information gathered by sight, hearing and mainly touch. Acquiring the necessary manual and visual skills for such task is just a question of experience, which is usually obtained by practicing.

Partly thanks to the use of haptic interfaces (i.e. hardware that simulates touch); training surgery using virtual reality simulators is becoming more important every day [10]. This is due to the fact that those systems allow practicing in a secure and controlled environment, without having any kind of risk for both the patient and the medical trainee, the same way virtual words are being used for other types of training processes in health [11]. Working with haptic interfaces in a virtual environment allows having almost infinite practical cases [12]: from simply measuring the pulse of a patient or the hardness of a tissue to learning how to perform a special cut during a complex surgical procedure. This context presents the issue of the lack of a standard framework [13] that enables to develop virtual reality simulators to be used as tools for teaching surgery.

The goal of this paper is to detail the use of the proposed software design described in [9] as a framework that can be used to develop haptic simulators in the context of a software engineering course using an Agile methodology approach. The main advantages of this approach are, on the one hand, that students do not start a project from scratch, nor in its design, nor in its implementation, so that valuable results can be achieved in a semester. On the other hand, the architecture combines several design patterns for students to adjust and complete, which is the way to really learn them. Besides, the development process including a lead user provides an environment closer to the real world, and the final product obtained, a cataract surgery haptic simulator, can be used as a teaching/learning tool in a context so different from software engineering as veterinary surgery. Our proposal does also include several tools that the teacher can use to evaluate the student's performance during the process and also the final product quality.

This work is structured as follows: section 2 summarizes SHULE's design. Section 3 details the Scrum process proposed for a software engineering course including the first phase and a generic sprint, together with an example application. Finally, section 4 presents the conclusions of this work.

2. Framework

The work presented in [9] includes the formal definition and the object-oriented software design of a framework that allows to model haptic teaching/learning systems by using virtual reality simulators. Although in that work, the application field presented is surgery, the framework has been proposed as a more general purpose one. Its main features are detailed below.

2.1 System's description

The main goal of the framework is to build haptic simulators that can be used as teaching/learning tools in educational environments. A haptic simulator includes the sense of touch, so that any kind of procedure involving some degree of expertise related to that sense is suitable for being considered. The framework incorporates the domain expertise knowledge in order to produce rich feedback to the user. The only requirement is that the process to be represented has to have a sequential order in the main tasks performed.

The framework allows to dynamically model a sequential procedure. The procedure is considered as a series of ordered *steps*. Each step includes every possible action (named *state*) that can be performed in such context, both the "expected" and the unusual ones. Furthermore, each action is triggered by some events or situations that may occur during the performance of the procedure. It should be noted that, unlike steps, actions do not follow a sequential order because a step can be completed in different ways.

Therefore, the framework must offer a mechanism to control, at any time, the action that is currently being performed and thus determine which one is next, based on the events. Thanks to this design, an expert may model a procedure according to its needs and preferences and, if needed, insert any modifications later in a simple way.

However, the key of the framework lies on learning from its use, which implies the need to capture all the information related to the performance of the procedure during the simulation. In other words, capture the information related to which elements of the simulation have been touched, moved or modified, in which exact place, in what way, etc.

With the purpose of evaluating the information, the framework must also offer a mechanism to capture such information in a precise and automatic way.

2.2 Design patterns used

The previous section described that the framework has some "gaps". From the design's point of view, the proposed solution described in [9] to fill those gaps includes the combination of three design patterns:

2.2.1 State machine

The first obstacle is to find a representation of the mechanism that allows the simulation to control the action at any given moment and determine which is the next action, based on the events that occur. The State Machine design pattern [14] meets these requirements, because it can alter the behavior of an object when its internal state changes. Besides it perfectly fits the proposed abstraction model for modeling such procedures, as it is comprised of states and transitions.

2.2.2 Visitor

The next obstacle is to find a mechanism that is able to capture the information related to the performance of the procedure. During the simulation, the user interacts with the elements of the simulation, which constantly produces information. However, before collecting any kind of information, the framework needs to know from where it can be done. The Visitor design pattern [1] comes in very handy for this task, because it allows defining new operations for a class without changing it. For the particular case of the framework, such operations will consist of setting the mechanisms that gather the information on each state.

2.2.3 Observer

The last obstacle is a direct result of the latter since it concerns the automatic capture of information. The best candidate for this task is the Observer design pattern [1]. It defines a one-to-many dependency between objects so that when one object changes its internal state, it notifies to all its dependent objects to update automatically. Inside the framework, the Observer's role consists of gathering the information resulted from the user interacting with the elements of the simulation involved in the procedure.

2.3 Information

Once the basic components of the framework architecture have been settled, there is still an important part missing. Thanks to the combination of the three designed patterns mentioned above, the mechanism for obtaining information about the user performance with the simulator is already determined. That information has to be also included into the framework design in such a generic way that it can be suitable for many types of specific information.

Inside the framework, the information has been designed associated to each element in each state. Three types of information are provided: recordable data, reasonable data and computable data. The first one corresponds to the information that has to be stored and later showed to the teacher about the student's performance. The second one is the information for providing the student with feedback about her performance. The last one is the information used to elaborate a grade for each step and a final grade of the whole procedure.

In order to make the information treatment as flexible as possible, the Strategy design pattern has been used [1]. Three families of algorithms have been defined, one for each type of information, so that the specific strategies can vary independently from the clients that use it, in this case, the class representing that information.

3. Scrum process

Software engineering courses usually face a situation where a reasonable size application has to be developed within a semester period. When students enter these courses, they have already had at least four semesters of programming related courses, so that they have acquired the basic background needed to accomplish a whole project development.

Our proposal is to use our framework, SHULE, for developing a haptic simulator as a teaching/learning tool following a Scrum development methodology [15]. The main advantages of this approach are:

- Development does not start from scratch, so valuable results can be achieved in a semester period of time.
- The framework is flexible enough to allow dealing with a broad range of applications as final products.
- The architecture combines several design patterns and using design patterns is the way to learn design patterns.
- Implementing a Scrum methodology on a real product brings real business world into the classroom.
- If the development can incorporate a lead user, software engineering students' experience will result enormously enriching.
- The final product can be used as a teaching/learning tool in another environment.
- The teacher can use several tools to evaluate the student's performance during the process and also the final product quality.

The main issues to consider for each new project are the final product and the development team. The development team is composed by the students of the software engineering course. Choosing the product is the key for the rest of the process. Depending on the product, the lead user has to be set. The ideal case involves a real user from a different field. In the work presented, the lead user was a veterinary professor teaching surgery. The software engineering teacher assumes the Scrum Master role and one of the students is assigned the Build Master role. This role is extremely important due to the tools that will be used to support the continuous integration and continuous delivery process linked to the product development. The rest of the students will be part of the development team and they will have different tasks assigned for every Sprint.

In the rest of the section the product development will be shown in detail. The first phase of the process, the product backlog, and one Spring will be described. For each phase, there is a subsection explaining what SHULE provides, another one detailing what the development team has to code, and the last one describing a real example of use. The section ends detailing the tools used to support the development process using continuous integration and delivery.

3.1 Product backlog

The goal of the starting meeting is somehow to establish the equivalent to the initial user requirements, the product backlog, in terms of user stories. The only condition that SHULE demands is that there should be a sequence of steps to be performed in a specific order inside the simulator, although inside each step there is not a preset order. For instance, in a cataract surgery procedure, the example that will be used all through this section, an incision has to be made in the anterior capsule before the cataract can be removed, but the actions to be performed in order to make that incision can vary widely. Each one of these steps will be assigned to one Sprint and the development order does not need to be a particular one. The Scrum team can decide which step to implement first, taking into account the opinion of the lead user.

3.1.1 What does SHULE provide?

- The overall framework architecture for the development of the haptic simulator.

The first meeting of the development team is a very important one. During this meeting everyone has to understand how the framework that they are going to use works. That means that both profiles, lead user and software engineering students, have to see it clear. The teacher is an important part of that meeting because, as Scrum Master, has to assure that the process is being developed properly.

For the lead user, the artifacts used to explain it will be user stories. These same user stories will be used to establish the product backlog.

For the students, the artifacts used will be the class diagrams of the framework architecture showing the design patterns involved. Among these patterns, the state machine, implemented with the state machine design pattern, is the simulator's key component. Its mission consists of controlling the simulation flow by having a set of states and transitions. Its implementation has been slightly modified with respect to the original pattern:

- Executing a simulation implies having a large number of transitions between states. To avoid creating and destroying states whenever a change of state occurs, a `StateFactory` class has been created to act as a "pool of states". This class stores all the states contained in the state machine of a particular step, so they can be provided on request.
- To define the list of transitions between states, a new `init` method for the `StateMachine` class has been implemented. This method initializes the list of transitions from a properties file.
- To uniquely identify each step and state, a new attribute has been added to the `StepContext` and `IState` classes.

More detailed information about the state machine design can be found in [9].

3.1.2 What is left?

- Defining the specific steps, choosing the first one to start with, and detailing it.

Once the whole team understands how SHULE works, the steps for the new product have to be determined. In that moment, the lead user can propose which step could be the first one to develop, although it will be up to the team. For that step, a detailed user story will be produced.

3.1.3 Example: cataract surgery

For our example, our lead user chose a cataract surgery simulator. The steps to perform this surgery were established and they are summarized in Table 1.

Once the product is chosen and its steps settled, the framework can be explained to the whole team using that specific product. For the lead user, a diagram like the one showed in Fig. 1 was used. This diagram serves two purposes. The first one is to be the artifact for sharing a common language among the different types of stakeholders, in this case, software engineers and surgeons. The second one is that the whole team understood the framework architecture for the simulator development.

What Fig. 1 tries to transmit is that the surgery is divided into six steps that are sequential (first row). Inside each step, a non-sequential set of actions can take place, depending on the tool chosen by the user

Table 1. Steps defined by the lead user

Step	Name-ID	Description
1	*preparation*	Small prick in the cornea and viscoelastic filling of the anterior chamber
2	*capsulorhexis*	Tear apart a circular section of the anterior lens capsule
3	*phacoemulsification*	Remove the nucleus of the clouded lens
4	*insert lens*	Inject the lens implant and position it correctly
5	*final rinse*	Remove viscoelastic
6	*close*	Remove aux route and close aperture

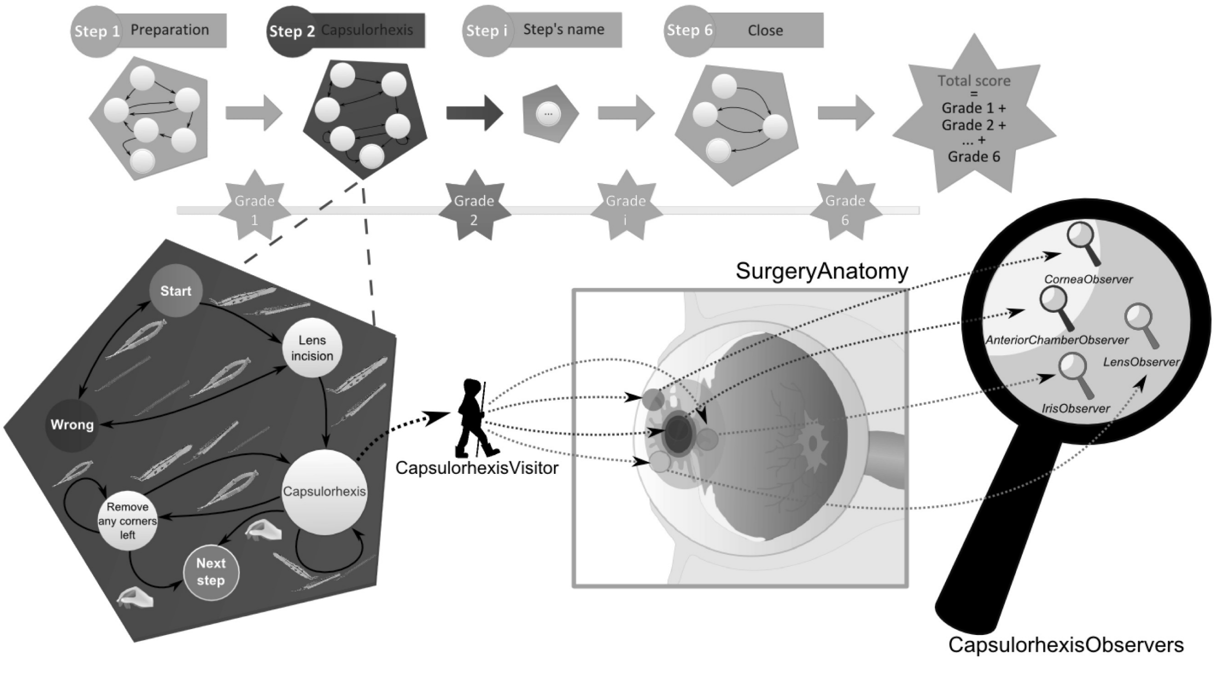


Fig. 1. Relationship between the state machine and the visitor pattern for the capsulorhexis step.

(second row). When one step is finished, the performance for that step is evaluated and a grade is obtained (third row). These grades will be used to obtain the final one (Total score). In the last row, a closer look to one particular step is taken. In the first part it can be seen how the changes in the states are driven by the change in the tool. It is also shown that, for every state, there is someone, a visitor, in charge of placing observers (magnifiers) on every element of the anatomy. These observers will collect all the information needed from every element of the anatomy depending on the active state. For instance, if active state is `capsulorhexis` or `remove corners` (states 3 or 4), the lens nucleus (element 8) can be touched, but the lens posterior capsule (element 9) cannot (see Table 6).

For the software engineering students, a class diagram of the state machine pattern, also applied to the final product, was used. The lead user proposed to start with the step number 2 and that decision was accepted by the team, so detailed information had to be obtained for that step. SHULE establishes which is the information needed to build the system following the design architecture of the framework. The information includes: the states of the step for the state machine (see Table 2), the tools available for the task (see Table 3), the state transitions depending on the tool chosen (see Table 4), the elements of the anatomy involved in the step (see Table 5) and the feedback to be provided for the student and the teacher (see Table 6).

This information is the basis for detailing the step number 2 in the first Sprint. Besides, this information was also used to develop the class diagram showed in Fig. 2. The diagram shows the classes and relations involved in the state machine pattern. In this figure, the states for the second step are shown.

Table 2. States for the capsulorhexis step

State	Name-ID	Description
1	*start state*	Initial state for this step
2	*lens incision*	Small prick in the anterior capsule
3	*capsulorhexis*	Tear a circular opening in the sac that holds the cataractous nucleus
4	*remove corners*	Check & remove any corners from the circular opening
5	*next step*	Continue to next step
6	*wrong state*	User did something wrong

Table 3. Tools for the capsulorhexis step

Tool	Name-ID	Description
1	*forceps*	capsulorhexis forceps
2	*cystotome*	small knife with a tiny curved or hooked blade
3	*small scissors*	scissors to remove corners left in the opening of the anterior capsule
4	*slit knife*	scalpel to enlarge precision incision

Table 4. State transitions depending on tools

State/Tool	1 (forceps)	2 (cystotome)	3 (scissors)	4 (slit knife)
1 (start_state)	2	2	X	X
2 (lens_incision)	3	3	X	X
3 (capsulorhexis)	3	3	4	3
4 (remove_corners)	3	3	4	3

Table 5. Elements of the anatomy involved in the surgery

Element	Name-ID	Description
1	*cornea*	Transparent front part of the eye
2	*aqueous humour*	Clear fluid filling the anterior part of the eye
3	*anterior chamber*	Space inside the eye between the iris and the cornea
4	*iris*	Diaphragm located in front of the lens
5	*posterior chamber*	Space between the iris and the lens
6	*lens*	The lens of the eye divided into anterior capsule, nucleus and posterior capsule
7	*lens anterior capsule*	Anterior layer of the lens
8	*lens nucleus*	Central part of the lens
9	*lens posterior capsule*	Posterior layer of the lens
10	*vitreous humour*	Clear gel that fills the space between the lens and the retina
11	*retina*	Posterior layer containing photo receptors

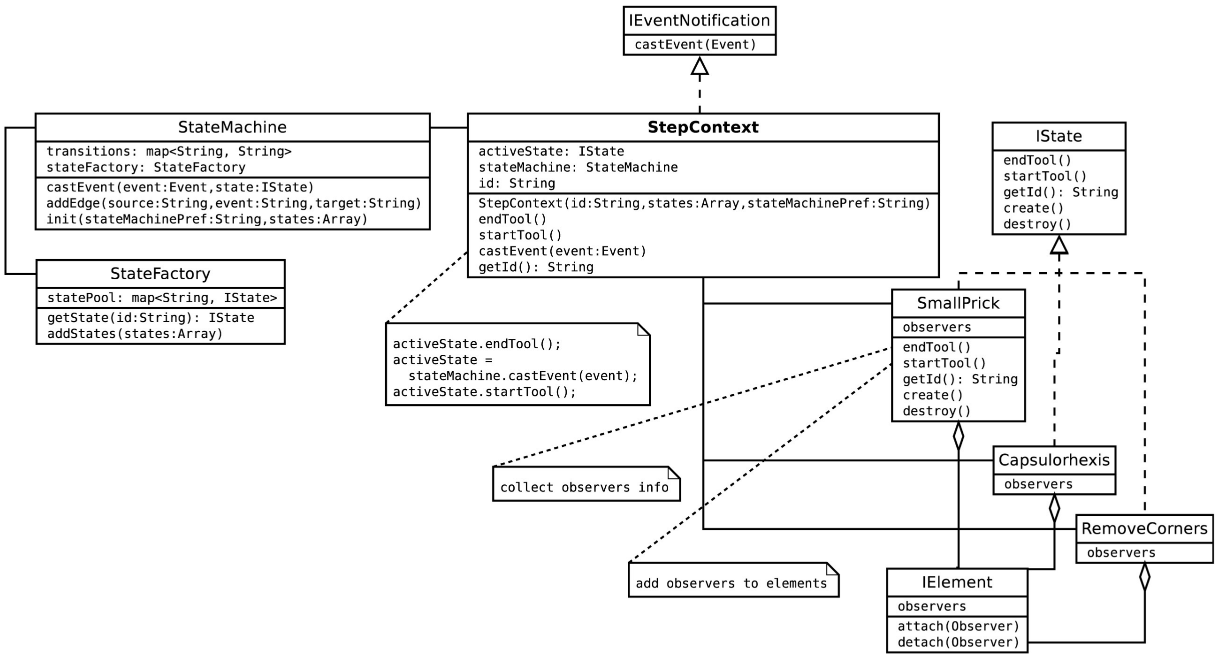
Table 6. Feedback for capsulorhexis step

States	Elements	Feedback	Outcome	Performance (– / = / +)
1	*all*	Just one tool has to be chosen	Error message if the right tool was not chosen	Negative if wrong tool chosen
2	*7*	Tool used: Position, angle and depth of the incision	Information about the quality of the incision	Position, angle and depth are ranked and graded accordingly
2	*not 7*	Should not be touched	Error message if touched	Negative if touched
3	*7*	Tools used: Shape of the opening and space trajectory of the tools used	Comments on the opening	Different shapes are ranked and negatives are assigned to each swap of the tools used
3, 4	*6*	Can be touched but not pushed hard	Error message if pushed hard	Negative if pushed and fatal if pushed hard
3, 4	*8*	Can be touched	No outcome	No grade influence
3, 4	*9*	Should not be touched	Fatal error if broken	Negative if touched and fatal if broken
3, 4	*not 6, 7, 8, 9*	Should not be touched	Error message if touched	Negative if touched
4	*7*	Time spent cutting the corners, number of cuts and shape of cuts	Messages about the performance with the scissors	Negative as more time is spent, negative for each new cut, and ranked shapes of cuts
5	*all*	Summarize information compiled during the step	Resume of the information compiled	Combined grade for this step
6	*all*	Wrong tool selected: which and when	When which tool was selected	Negative for every wrong tool selected

Although in Table 2 there are six states, only three are represented here. The other three are the `start state`, the `next step` and the `wrong state`. These three states are common for all the state machines, so they are not included in the diagram. The `start state` is an initial state for waiting for the user to choose one tool. The `user manually chooses the next step state` whenever she thinks she is done with the step, and the `wrong state` is not really a state. When the user chooses a wrong tool from any of the other states, the state machine swaps to the `wrong state`. Once there, the information about the initial state and the tool selected is stored for grading and feedback purposes, and then the state is swapped again into the initial one.

3.2 *Sprint N*

The goal of every Sprint is to implement, test and deliver one step of the simulator. Depending on the



```
activeState.endTool();
activeState =
  stateMachine.castEvent(event);
activeState.startTool();
```

Fig. 2. State machine design pattern with states for step 2.

complexity of the specific step, the Sprint length should be adjusted. Provided the fact that design patterns allow dividing coding tasks very clearly and quite efficiently, the size of the development team will also affect the length of each Sprint.

At the end of each Sprint, a retrospective meeting takes place during one of the regular practice sessions of the course. That meeting is conducted as a start-stop-continue meeting where the team has to identify specific things that they should start doing, stop doing and continue doing. During this meeting, the goal and the tasks for the next Sprint are defined and also assigned.

3.2.1 What does SHULE provide?

- The state machine esqueleton plus the combination between visitor and observer patterns, and information design.

The main core of the state machine has already been explained in the first meeting, so we are going to focus on the three remaining parts: visitor pattern, observer pattern and information design.

3.2.1.1 Visitor pattern

The role of the Visitor design pattern is to go over the different elements of the scene implicated in a concrete state. This is done in order to set the different observers that will later gather the information resulting from the user's interaction within the simulator.

The implementation of this design pattern requires defining the `IStateVisitor` interface including a specific method for visiting each of the elements of the scene. One class will implement this interface for each one of the states of the present step. Moreover, in order to "accept" the Visitor in an element of the scene, the class `IElement` has to include the `accept (IStateVisitor)` method.

Each time a concrete "visitor" visits a concrete element, the visitor substitutes the current observer with another one that collects the information required for that new state. This action takes place each time the user chooses a different tool, which in turn generates a change in the active state of the state machine. However, during the implementation of this pattern, a new issue arose about collecting the information captured from the observers during a simulation session. To solve this need, the class `InformationVisitor` was added. As a result, each time a visitor "visits" a concrete element of the scene for changing its observer, the visitor starts to process the information generated by the old observer from that element before removing it. The whole process corresponds to the `starTool()`and `endTool()` methods of each `IState` (see Fig. 3 and Fig. 4).

3.2.1.2 Observer pattern

The Observer design pattern is the link between the framework and the information resulting from the user's interaction. For its implementation, the `IOb-`

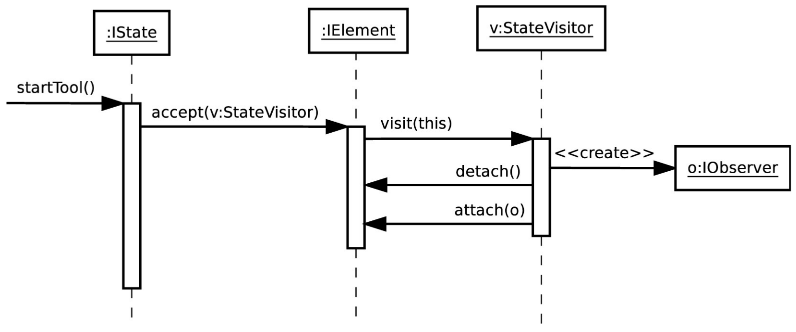


Fig. 3. Sequence diagram—`startTool` process.

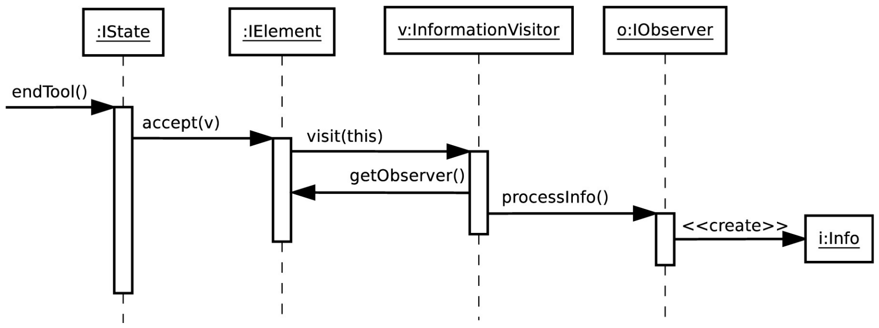


Fig. 4. Sequence diagram—`endTool` process.

server interface has been defined. One class will implement this interface for each one of the states of the present step. Besides those ones, the NoActionObserver is provided. This observer handles the case in which the user should not interact with an element of the scene in a particular state. In addition, the methods attach(IObserver) and detach(IObserver) are located in the IElement interface for it to act as the subject of the observer pattern.

For the part of the observer that offers the service of collecting information, the method processInfo() was added to the IObserver interface. This method is called from the InformationVisitor and is responsible for creating the information that assesses the user's performance within the simulator (see Fig. 4). More details of this method will be described in the next subsection.

3.2.1.3 Information

One of the most important aspects of SHULE is information modeling. Information obtained from user interaction with scene elements has to be modeled. In order to evaluate the user, the first step is to have detailed and structured information about that user interaction. This information is obtained from the data collected during simulation operation. Both info and data are described in the next subsections.

Info

In the design explained by now, observers are in charge of getting information, but it has not been detailed yet; how is this info defined? How is it stored? Who has the responsibility to control and process it?

First of all, different info is obtained for each particular scene element in each state. During the interaction process with the element, basic data is generated for later producing the info. This is raw data, that is, it has not yet been classified and it is related to atomic operations, for instance, take or leave a tool, move a tool through a region, touch a specific part of an element of the scene, etc. Those data, once processed, will be the basis for the info. In the next subsection, the data involved is detailed and it is shown how it is processed.

In order to evaluate a student, a final mark is obtained from the partial ones obtained for every step of the procedure. In turn, in order to obtain the mark for one step, all its states are involved. This means that there must be information units for each state. Each step will store its own information unit that composes from the individual units of each state.

Once the info is defined, the framework has to store and manage it from somewhere. Taking into account the overall structure of the architecture, that information should be kept inside every step because that is the place where each state is included. In the end, the simulator will manage the whole information because it is aware of the active step in each particular moment.

Figure 5 shows the class diagram modeling the information. The actors involved are:

- **Info**. Represented by the container Infos that aggregates Info objects. The Info class represents the information unit obtained from a particular element of the scene during a specific state and includes the set of processed data. The Infos class is only in charge of storing and managing Info objects for an individual step.
- **Simulator**. Represented by the Simulator class. This class controls the simulation and stores a reference to the active step all the time.
- **Step**. Represented by the StepContext class.

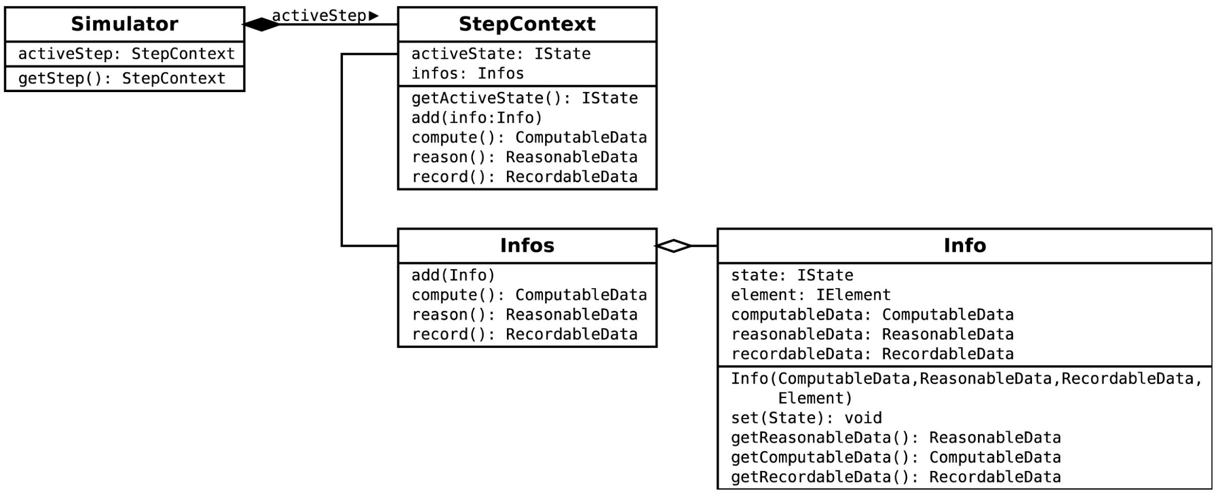


Fig. 5. Class diagram of the information of the framework.

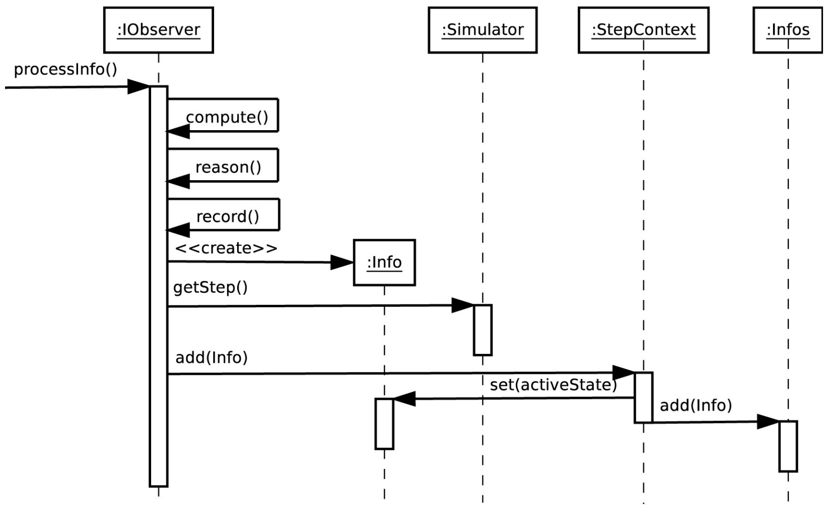


Fig. 6. Sequence diagram—creating an `Info` object.

This class represents one step of the procedure. It is in charge of storing all the information obtained during its execution. It also maintains a reference to the active state.

The process by which the visitor gets the information by means of the observer is detailed in Fig. 6. When the `InformationVisitor` invokes the `processInfo()` method of an `IObserver`, it is the moment for it to process all the information stored by calling the methods `compute()`, `reason()` and `record()`. As a result, the `IObserver` will be able to create a new instance of Info. Next thing to do is to store that new information in the correct place. For doing that, the IObserver has to know which one is the active state. The active state has to be included in the information itself before it is stored in the active step infos.

Data

Prior to generating a user evaluation, the information itself has to be built. During the simulation, the interaction process generates a lot of raw data that has to be classified in order to obtain an information unit useful for both the framework and also the final users.

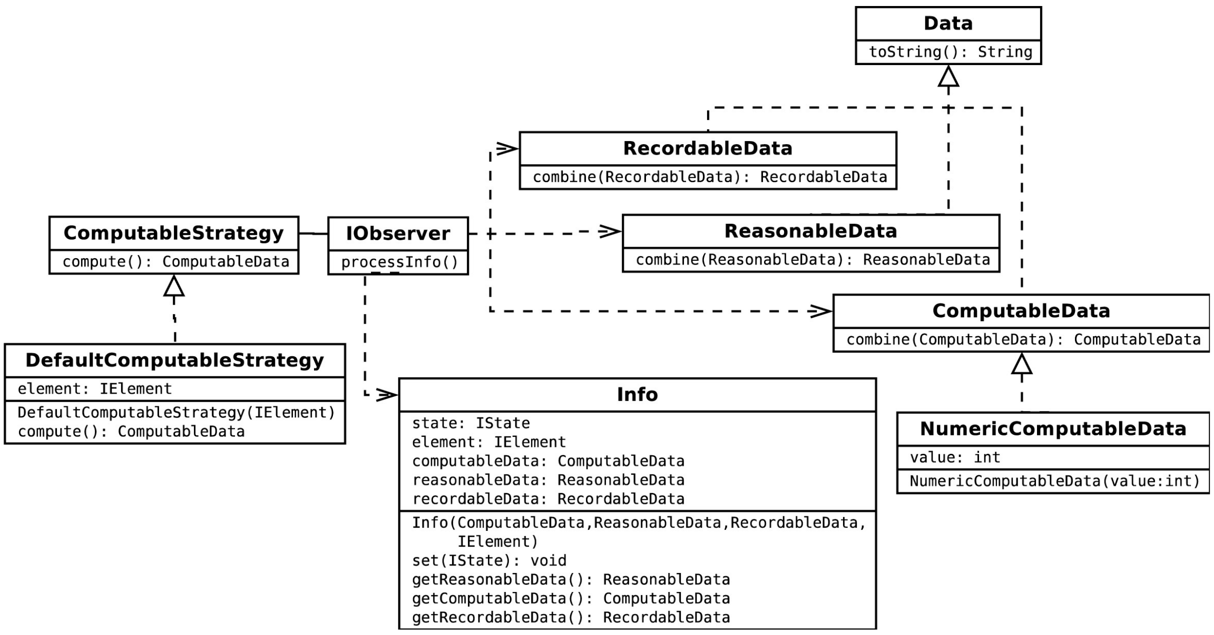
In the definition of the model [9], it is stated that each training session with the simulator includes three main elements:

1. On the one hand, a mark to evaluate the student performance has to be obtained. This mark comes of a series of calculations comparing initial determined parameters with the data obtained during the interaction (incision angle, length or shape of cut, depth of cut, etc.). This type of element is named `Computable` meaning that this data has to be computed.
2. On the other hand, the expert obtains information about the student's performance during a training session. This way, the expert can verify or reconsider the design of the surgical procedure. This information is obtained by processing technical data to make it meaningful to the user. Data processed include information about actions as the following ones: time to complete one step, number of times that a particular part of the anatomy has been touched, type of tools used, etc. This element is named `Recordable` referring to the fact of recording the user interactions.
3. Finally, together with the mark, the student receives detailed information about her training session. This feedback can be useful for future sessions. The information, in a similar way that the last one, is obtained by processing technical data, but this time, the rationale behind the mark is explained to the user. Data processed includes references to technical data such as: number of errors, appropriate time to complete a step, degree of precision during the procedure, etc. An element of this type is named `Reasonable` because, unlike `Recordable`, this one is enriched with some logic for adding a meaning to the data obtained.

Each of these elements works with different data. The combination of all of them is an information unit. Nevertheless, before that information becomes useful, data has to be processed. Taking into account that each data is different, the way they are processed should also differ.

The modeling of the requirements described above is shown in the class diagram of Fig. 7. The actors are:

- **Data**. Represented by the Data interface. There



Data
toString(): String
RecordableData
combine(RecordableData): RecordableData
ReasonableData
combine(ReasonableData): ReasonableData
ComputableData
combine(ComputableData): ComputableData
NumericComputableData
value: int
NumericComputableData(value:int)
ComputableStrategy
compute(): ComputableData
IObserver
processInfo()
DefaultComputableStrategy
element: IElement
DefaultComputableStrategy(IElement)
compute(): ComputableData
Info
state: IState
element: IElement
computableData: ComputableData
reasonableData: ReasonableData
recordableData: RecordableData
Info(ComputableData,ReasonableData,RecordableData, IElement)
set(IState): void
getReasonableData(): ReasonableData
getComputableData(): ComputableData
getRecordableData(): RecordableData

Fig. 7. Class diagram of the data comprising an information unit.

are three abstract classes that implement it (`RecordableData`, `ReasonableData` and `ComputableData`), one for each type that forms an information unit. Sometimes data has to be expressed as a combination of several ones of the same type. For that season, every concrete class includes a method named combine to represent that possibility. Besides, each type of data can be specialized in order to offer a more detailed version, as for instance the `NumericComputableData` class.

- **Observer**. Represented by the `IObserver` interface. All through this work it has been stated that the observer is the one in charge of getting the interaction information. For that information to be used during the evaluation phase, the interface includes a method named `processInfo` that allows processing data to be transformed into instances of the `Info` class, as it has been shown earlier.
- **Info**. Represented by the Info class. This class represents the information unit once the `IObserver` has processed data. Its task consists in storing all the processed data obtained from an element of the anatomy in a specific state of one step of the surgical procedure.
- **Strategy**. Represented by the `ComputableStrategy` interface. This interface and its concrete class, `DefaultComputableStrategy`, correspond to the Strategy design pattern [1]. This pattern allows maintaining a set of algorithms encapsulated in order to make them interchangeable. For the data, the pattern allows defining different calculation strategies.

Figure 8 shows the sequence diagram for computing a `Computable` data using a Strategy design pattern. Data obtained by the `IObserver` is raw data that has to be processed. To achieve this goal, the instance of the `IObserver` makes a call to its

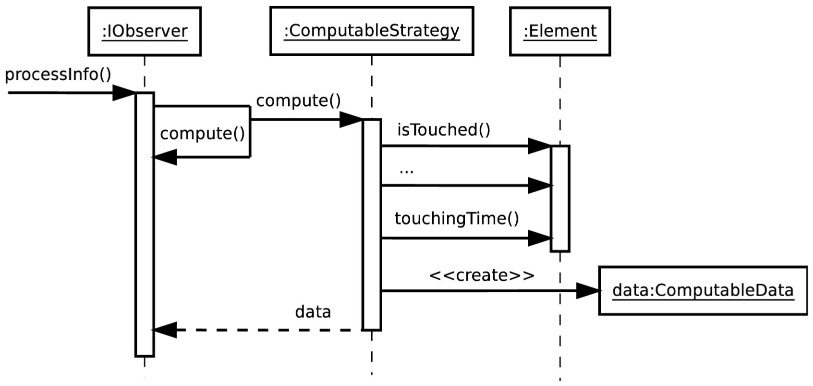


Fig. 8. Sequence diagram—data calculation.

compute method as part of the processInfo() method (see Fig. 6). That method delegates into a concrete strategy the task of obtaining and calculating the specific data. The instance of the strategy, ComputableStrategy, is in charge of obtaining the parameters needed from the element that the IObserver is observing, IElement. From this IElement, the strategy is able to create a new instance of the ComputableData class containing the data just obtained. At the end, this instance is returned to the IObserver, who will use it to build an information unit.

3.2.2 What is left?

- Define specific states, visitors, observers and data from user stories.

Specific tasks have to be established for the team to estimate effort and assign them. Depending on the size of the development team, the size of those tasks can vary. For instance, one task can involve the development of one of the states of the state machine, or it can comprise the development and integration of all of them in the state machine structure. The list of tasks will include but not be limited to the following ones:

- State machine states: One state class for each new state defined for this step.
- Concrete visitors: One visitor class for each new state defined for this step.
- Concrete observers: One observer class for each new state defined for this step.
- Info: Specific classes that inherit from ComputableData, ReasonableData and RecordableData and represent the information to be obtained.
- Concrete strategies: One ComputableStrategy, ReasonableStrategy and RecordableStrategy class for each way defined to compute, reason and record the information obtained.

Each task, of course, includes coding, unit testing and documenting. It is important to keep in mind the need for the build master role in charge of building and deploying each version of the final application.

Besides what can be considered pure coding, there is also another important task for the final product, which are the anatomy and the tools that the user can choose. A graphical representation has to be developed using some 3D tool. The tools will be shown as the haptic cursor each time the user chooses a different one and will also cause a different behavior for the haptic cursor.

3.2.3 Example: Cataract surgery

The whole design of the state machine for the second step of the cataract surgery was already presented in Fig. 2. The main three states included in this step: SmallPrick, Capsulorhexis and RemoveCorners, are the basis for what is left of the visitor and observer patterns.

Figure 9 shows the class diagram for the visitor pattern of the step 2. The visitor interface includes one method for visiting each element of the anatomy involved in the surgery procedure. In this case, only four elements can be affected by this step, so the visit method is overloaded for visiting the anterior chamber, the cornea, the lens and the sclera. In Fig. 10, the sequence diagram of the SmallPrickVisitor is shown. When the active state is SmallPrick, a small incision in the cornea has to be done, but the sclera should not be touched. For this reason, a SmallPrickObserver is attached to the cornea element and a NoActionObserver is attached to the sclera element.

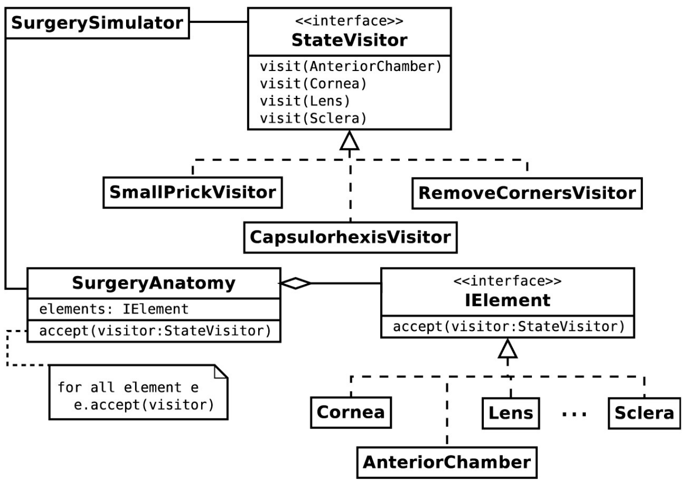


Fig. 9. Visitor pattern for step 2.

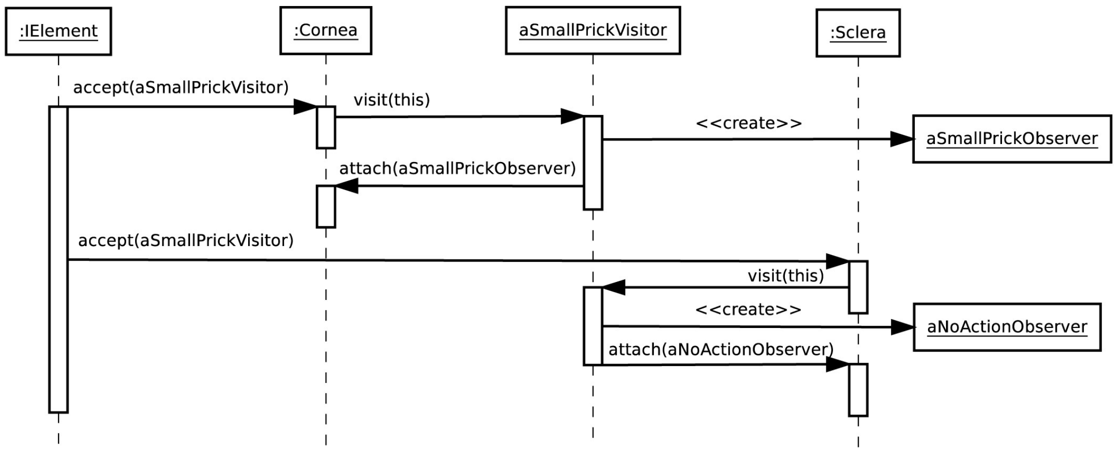


Fig. 10. Sequence diagram—`SmallPrickVisitor`.

Figure 11 shows the design of the observer pattern for the step 2. One observer is needed for every state, so that they can be placed in some elements of the anatomy. The visitor is the one that knows which observers have to be placed in which elements. The role of each observer is shown in Fig. 12. Whenever the cornea is touched, the observer is notified so it can update the information related, for instance, to the cut position. This information will later be collected in order to create objects of the `Info` class.

Currently, there are issues to define suitable metrics for assessing the surgical skills of a medical trainee. Regardless of the metrics used, the framework needs to be able to automatically collect and process the information resulting from a training session with the simulator. The evaluation of students' performance in a surgical environment is not new [16, 17]. Traditionally, this was based in the review of a sequence of actions and in the observation of how users carry them out [18–20]. With the popularization of the use of simulators and virtual worlds in surgical environments, the landscape has changed. Simulators facilitate the evaluation of several issues that are not easy to assess in real experiences. The "information" gathered through the simulation can be used to carry out different kind of tests [17, 21, 22]. Some examples of the "information" to study can be: the force employed with a surgical tool, the angle used to make a cut, the skill in the use of the instrument (movements/time), how the users deal with the tissue, etc. [22, 23].

Therefore, by using the information gathered through the simulation, it is possible to enrich student's assessment. However, objective metric definition is a very complex process [24, 25], because it may depend on the type of surgery to be performed and on experts' expertise [17]. For the specific case of a haptic simulator, three types of information should be stored:

1. Information about user's action in each step. That is, useful information about student performance in a specific step. This information could include: the time used to complete the step; issues related to the surgical technique (pressure applied, tissue damage, depth, number of cuts, surgery accuracy, proper use of tools); procedural issues (if the actions were carried out in the right order); and issues related to success of the surgery (if the final outcome is successful, in what percentage, degree of improvement, etc.).
2. Feedback information to show to the users depending on how they carry out the step. The expert provides them with several possible workflows for each surgery. Depending on how the students complete the steps, the simulator will send them a different feedback. The feedback could change depending on user's gathered information. The framework aims to use this to increase student success.
3. Information about the students' performance or success in the activities that they carry out. The expert sets up criteria and depending on the information about how the users progress in the step, they achieve a grade.

All the information has been encapsulated into the Info class, which is composed by `ComputableData`, `ReasonableData` and `RecordableData`. The format of each of them has to be defined, but both ReasonableData and RecordableData will mainly be just strings provided by the expert. Their combine method would be in charge of removing the replicate messages and reorganize the other ones following the order established by the lead user. The case of the `ComputableData` is different because is information that has to be computed. A strategy pattern has been used to allow the computation algorithm to be the one

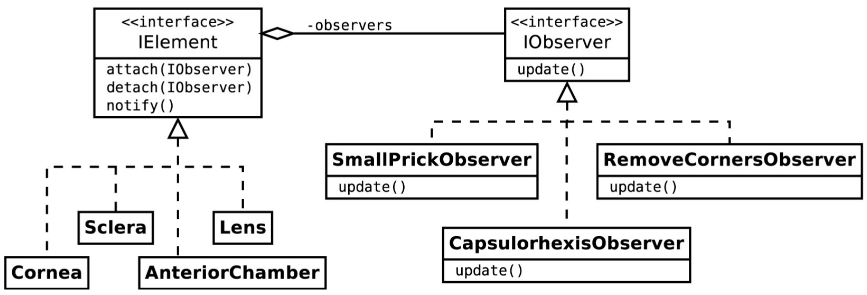


Fig. 11. Observer pattern for step 2.

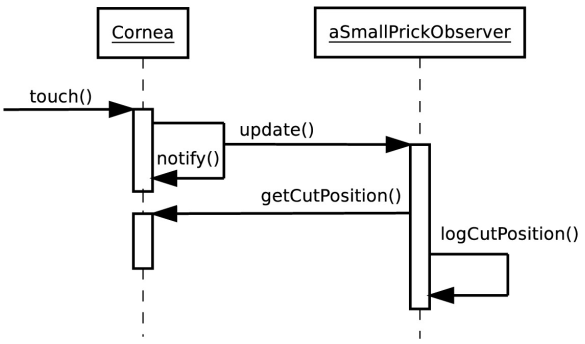


Fig. 12. Sequence diagram—`SmallPrickObserver`.

needed for every situation. A `DefaultComputableStrategy` is provided for this case. This strategy uses `NumericComputableData` to represent the negative and positive information assigned to every action by the lead user (see Table 6).

The information provided about users progress provided by the simulator may be used in other performance assessment systems used with surgery students, such as: OSATS [26] and HMA [27]. This guarantees the simulator portability, so they could be used in other contexts and for other surgery activities.

The only part that is left is the surgery anatomy and the graphical tools. Table 7 shows some of the tools developed for this simulator and a screenshot of the eye anatomy development. The models were developed by some students that had previous experience with 3ds Max[1] software. This task is only performed once for all the steps.

3.3 Tools

This section includes the tools used for students to work with the haptic devices and for teachers to evaluate students' performance.

3.3.1 HBOgre

HBOgre is a software library developed by our research group. It uses an open source 3D engine, Ogre[2], for graphics, and a real-time physics library, Bullet[3]. The haptic part of the library abstracts the haptic device and all the other parts are synchronized. The result is a library that offers a simplified interface for creating new haptic simulators by abstracting low level details.

This is one of the key parts of the development of haptic simulators using SHULE. Without HBOgre, the rapid development needed for accomplishing this project in one semester would not be possible.

3.3.2 Support and evaluation tools

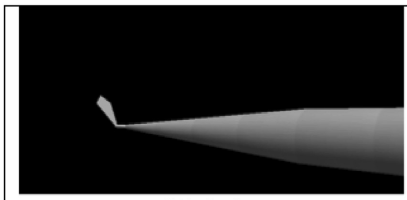
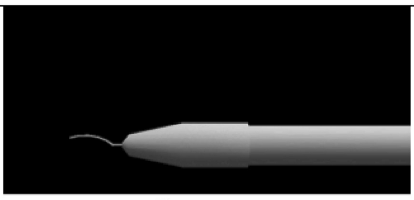
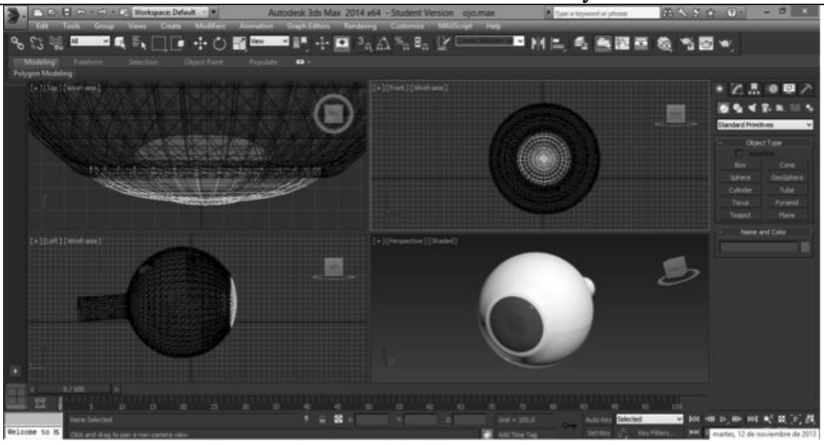
One important part of the process is how to evaluate the students that take part in the development. These are the tools used for both supporting the process and also for evaluating the students' work.

- PivotalTracker[4]—This tool was used for supporting the Scrum methodology. This web application includes as its functionalities creating a project, including stories, estimating points and velocity, modifying the story workflow, receiving notifications and obtaining monitor charts. Tracker is free for public projects, non-profits and academic institutions.
- GitHub[5]—Code host for both git and subversion repositories. Using a history statistics generator such as GitStats[6], the activity of each student working in the project can be traced.
- Jenkins[7]—This open source continuous integration server was used combined with GitHub and Sonar for the project development.
- Sonar[8]—For code inspection, Sonar offers plugins to work with several programming languages and also for checking individual contributions to a project using Developer Cockpit[9].

[1] http://www.autodesk.com/products/3ds-max

[2] http://www.ogre3d.org
[3] http://bulletphysics.org
[4] https://www.pivotaltracker.com
[5] https://github.com/
[6] http://gitstats.sourceforge.net/
[7] http://jenkins-ci.org/
[8] http://www.sonarqube.org/
[9] http://www.sonarsource.com/products/plugins/developer-tools/developer-cockpit/

Table 7. 3D Surgery tools and eye anatomy

 Slit knive	 Cystotome
 Eye anatomy in 3ds Max environment	

4. Conclusions

The paper presents an approach for using the SHULE framework in a software engineering course to develop a haptic simulator as a teaching/ learning tool using an Agile development approach. The main advantage of this proposal is that students do not start a project from scratch, nor in its design, nor in its implementation, so that valuable results can be achieved in a semester. The core of this framework combines several design patterns that students have to complete during the development. A Scrum methodology has been used to accomplish this project because it fits perfectly with the tasks separation that the design patterns included in the framework provided. Besides, including a lead user has provided an environment much closer to the real world than most traditional projects accomplished in software engineering courses. This final product can be used as a teaching/learning tool in another environment. Although it has been applied to building a surgical simulator, SHULE can be used for any application where expertise related to the sense of touch and also spatial sense are involved. If a real lead user can be involved, the experience turns out to be much more enriching because students have to be able to communicate ideas to a person from outside their field, and also have to be able to understand a lead user, which rarely happens during college education.

SHULE is shown as a valuable tool that complements lecture classes about design patterns. The Scrum process has been supported by the use of several tools that help the teacher evaluate both individual contributions to the project and also the final product quality. These tools have played an important role in the whole development. First, the web environment that supports the Scrum process has allowed students to have up to date information about the work of the rest of the team, and the teacher has been able to monitor the whole process. Second, using tools to analyze the data obtained from the repository has helped the teacher to know the code contribution of each particular student. Last, the use of a continuous integration server together with code quality analysis provides many important metrics to measure the final product in terms of lines of code, unit test coverage, documentation, duplications, complexity, violated rules, etc.

Although no formal questionnaire has been used, the experience has showed mixed results for students and for the teacher. For students, teamwork has shown to be a pending task once more. Although the use of tools to support the agile development has helped, they are not used to have individual assessments when working in teams. Particularly, they were quite surprised about all the information that can be obtained from their repository user, as graphics showing the periods and times of more activity, usually at the end of the semester, or their specific contribution to the system's code. For the teacher, the experience has shown very promising results, especially concerning assessment process. The use of a continuous integration server has proved to be essential to know the state of the projects in any particular time. The combination of this server with the tools that perform quality analysis releases the teacher from some tedious tasks about code conventions, for instance, while ensuring code quality and offering objective measures about it.

Acknowledgements—Special thanks to professor José Manuel Gonzalo Órden for being the lead user.

The authors would like to thank *Cátedra Telefónica de la Universidad de León* for its support. This work has also been partially funded by the Spanish *Ministerio de Economía y Competitividad* under grant DPI2013-40534-R.

References

1. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional, Boston, MA, USA, 1994.
2. I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
3. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Pearson Education, Limited, 2008.
4. T. Dybå and T. Dingsøyr, Empirical studies of agile software development: A systematic review, *Information and software technology*, **50**(9), 2008, pp. 833–859.
5. S. Jalali and C. Wohlin, Global software engineering and agile practices: a systematic review, *Journal of Software: Evolution and Process*, **24**(6), 2012, pp. 643–659.
6. N. Salleh, E. Mendes and J. Grundy, Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review, *IEEE Transactions on Software Engineering,* **37**(4), 2011, pp. 509–525.
7. T. Chow and D-B. Cao, A survey study of critical success factors in agile software projects, *Journal of Systems and Software*, **81**(6), 2008, pp. 961–971.
8. N. B. Moe, T. Dingsøyr and T. Dybå, A teamwork model for understanding an agile team: A case study of a Scrum project, *Information and Software Technology*, **52**(5), 2010, pp. 480–491.
9. G. Esteban, C. Fernández, M. A. Conde and V. Matellán, Design of a Haptic Simulator Framework for Modelling Surgical Learning Systems, In *Proceedings of the First International Conference on Technological Ecosystem for Enhancing Multiculturality (TEEM'13)*, Salamanca, Spain, November 14–15, 2013, pp. 87–94.
10. T. Coles, D. Meglan and N. John, The Role of Haptics in Medical Training Simulators: A Survey of the State of the Art, *IEEE Transactions on Haptics*, **4**(1), 2011, pp. 51–66.
11. F. J. García-Peñalvo, J. Cruz-Benito, C. Maderuelo, J. S. Pérez-Blanco and A. Martín-Suárez, Usalpharma: A Cloud-Based Architecture to Support Quality Assurance Training Processes in Health Area Using Virtual Worlds, *The Scientific World Journal*, **2014**, Article ID 659364, 10 pages, 2014. doi:10.1155/2014/659364.
12. P. J. Fager and P. von Wowern, The use of haptics in medical applications, *The International Journal of Medical Robotics and Computer Assisted Surgery*, **1**(1), 2004, pp. 36–42.
13. G. S. Ruthenbeck and K. J. Reynolds, Virtual reality surgical simulator software development tools, *Journal of Simulation*, **7**(2), 2013, pp. 101–108.
14. A. Shalyto, N. Shamgunov and G. Korneev, State machine design pattern, In *Proceedings of the 4th International Conference on .NET Technologies*, 2006, pp. 51–57.
15. J. Sutherland and K. Schwaber, The scrum guide. *The Definitive Guide to Scrum: The Rules of the Game*, 2011.
16. E. R. Petrusa, T. A. Blackwell, L. P. Rogers, C. Saydjari, S. Parcel and J. C. Guckian, An objective measure of clinical performance, *The American Journal of Medicine,* **83**(1), 1987, pp. 34–42.
17. R. Kneebone, Simulation in surgical training: educational issues and practical implications, *Medical Education,* **37**(3), 2003, pp. 267–277.
18. R. W. Barnes, Surgical handicraft: Teaching and learning surgical skills, *The American Journal of Surgery,* **153**(5), 1987, pp. 422–427.
19. R. M. Epstein, Assessment in Medical Education, *New England Journal of Medicine,* **356**(1), 2007, pp. 387–396.
20. A. R. Pulito, M. B. Donnelly, M. Plymale and J. R. M. Mentzer, What Do Faculty Observe of Medical Students' Clinical Performance?, *Teaching and Learning in Medicine,* **18**(2), 2006, pp. 99–104.
21. M. Tavakol, M. A. Mohagheghi and R. Dennick, Assessing the Skills of Surgical Residents Using Simulation, *Journal of Surgical Education,* **65**(2), 2008, pp. 77–83.
22. N. E. Seymour, A. G. Gallagher, S. A. Roman, M. K. O'Brien, V. K. Bansal, D. K. Anderse and R. M. Satava, Virtual Reality Training Improves Operating Room Performance: Results of a Randomized, Double-Blinded Study, *Annals of Surgery*, **236**(4), 2002, pp. 458–464.
23. R. V. O'Toole, R. R. Playter, T. M. Krummel, W. C. Blank, N. H. Cornelius, W. R. Roberts, W. J. Bell and M. Raibert, Measuring and developing suturing technique with a virtual reality surgical simulator, *Journal of the American College of Surgeons,* **189**(1), 1999, pp. 114–127.
24. A. M. Paisley, P. J. Baldwin and S. Paterson-Brown, Validity of surgical simulation for the assessment of operative skill, *British Journal of Surgery,* **88**(11), 2001, pp. 1525–1532.
25. A. G. Gallagher, K. Richie, N. McClure and J. McGuigan, Objective Psychomotor Skills Assessment of Experienced, Junior, and Novice Laparoscopists with Virtual Reality, *World Journal of Surgery*, **25**(11), 2001, pp. 1478–1483.
26. K. Moorthy, Y. Munz, S. K. Sarker and A. Darzi, Objective assessment of technical skills in surgery, *British Medical Journal*, **327**(7422), 2003, pp. 1032–1037.
27. S. Mackay, V. Datta, M. Mandalia, P. Bassett and A. Darzi, Electromagnetic motion analysis in the assessment of surgical skill: Relationship between time and movement, *ANZ Journal of Surgery*, **72**(9), 2002, pp. 632–634.

Camino Fernández received her degree in Computer Science from Universidad Politécnica de Madrid in 1994. M.S. degree in Knowledge Engineering in 1995 and Ph.D. in Computer Science in 2000 from the same University. She joined the Computer Science Department of the Universidad Carlos III de Madrid in 1995 as an assistant lecturer in Computer Science. In 2008 she joined University of León where she is part of the Robotics Research Group. Her main interests include haptic simulators, simulators for e-learning and serious games.

Gonzalo Esteban received his degree in Computer Science in 2009 and a MSc. degree in Cybernetics in 2011, both from the University of León (Spain), where he is currently pursuing his Ph.D. degree. Since 2011, he has been working as a researcher in the Mechanical Engineering, Computer Science and Aerospace Engineering Dept. of the University of León and is a member of the Robotics Research Group. His current research interests include haptic simulators and serious games.

Francisco J. Rodríguez-Lera is a Ph.D. student. He received the B.E. degree in Computer Science from the University of León, Spain in 2006, and the D.E.A in intelligent systems from the same university in 2009. He joined the Robotics Group, University of León in 2009 as part of SPL (Standard Platform League) team focused in Robocup competition. His

research interests focus on the human-robot interaction: verbal and non-verbal dialogue supported in Augmented Reality technologies and control architectures oriented to interaction. He is also involved in robot benchmarking in robotics challenges, telerobotics and assistive robotics.

Francisco Rodríguez-Sedano holds a Ph.D. in Computer Science (2010, University of León). In 2014 he joined the Robotics Research Group of the University of León. His research interests focus on the human-computer interaction, accessibility and design of user graphic interfaces.

David Díez holds a MSc in Computer Science and Technology (2007) and a Ph.D. Thesis in Computer Science (2009) from the Universidad Carlos III of Madrid. From 1998 to 2005, he worked as software engineering and project manager for different multinationals companies. Currently, he works as visiting professor at Universidad Carlos III de Madrid. His research interests are related to bridge the gap between design and engineering by using Agile development methodologies.