# Smart Decisions: An Architectural Design Game

Humberto Cervantes
UAM – Iztapalapa
San Rafael Atlixco 186, Col. Vicentina
CP 09340, Distrito Federal, Mexico
+5255-5804-4600 ext 1138
hcm@xanum.uam.mx

Serge Haziyev, Olha Hrytsay
Softserve, Inc.
111 Congress Avenue, Suite 2700
Austin, TX, 78701
+1-512-516-8880
shaziyev@softserveinc.com,
ohrytsay@softserveinc.com

Rick Kazman
SEI/CMU and University of Hawaii
2404 Maile Way,
Honolulu, HI, 96285, USA
+1-808-956-6948
kazman@hawaii.edu

## ABSTRACT
Architecture design is notoriously difficult to teach and to learn. Most competent architects in industry have deep knowledge won from long years of experience. But if we want architecture design to be methodical and repeatable, we need better methods for teaching it. Simply waiting for an aspiring architect to accumulate 10 or 20 years of experience is not acceptable if we believe that software engineering is a true engineering discipline. In this paper we describe our experiences with the development of a game that aids in teaching architecture design, specifically design employing the Attribute-Driven Design method. We discuss our approach to creating the game, and the "design concepts catalog" that provides the knowledge base for the game. Finally, we report on our experiences with deploying the game, and the (enthusiastic) assessments and feedback that we have received from industrial and academic participants.

## CCS Concepts
Software and its engineering → Software organization and properties → Software system structures → Software architectures; Software and its engineering → Software creation and management → Designing software → Software design engineering; Software and its engineering → Software creation and management → Software development process management → Software development methods → Design patterns

## Keywords
Software Architecture; Design Methods; Attribute Driven Design; Games for learning; Big Data.

## 1. INTRODUCTION
Among the different disciplines that exist in the field of software engineering, software architecture is one that has gained special importance in the last decade. As a matter of fact, Software Architect was ranked as the number one job in America in 2015 by CNN Money [6]. In [1], software architecture is defined as follows:

> *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*

At its core, the discipline of software architecture focuses on translating the most important requirements of the software system (also called *architectural drivers*) into a set of structures from which the system is developed. The translation of architectural drivers into structures is the process of *architectural design*. This process, as embodied in methods such as the Attribute-Driven Design (ADD) [13], is typically iterative: a subset of the drivers is selected at the beginning of an iteration and then design decisions are made to identify elements and create structures out of them to satisfy the selected drivers. Next, other drivers are selected and more structures are established, or existing structures are refined until an initial architecture is created. The design process is about making decisions, and often these decisions involve choosing among proven, documented solutions to recurring design problems. These proven solutions, which we call *design concepts*, are the building blocks for design. Design concepts can be conceptual, such as design patterns [7] or tactics [1], or more concrete such as application frameworks [5].

Although it is conceptually simple, the design process is notoriously difficult to learn and to perform with confidence. In practice there are many aspects that need to be considered when making design decisions—architecture is broad in scope and affects nearly all aspects of a project—and there are many decisions to be made. For instance, the selection of design concepts is difficult due to their large number and may be complicated by issues of cost, developer skills, standards compliance, as well as competitive and time-to-market pressures. Many times, architects rely on their experience when creating or modifying a design, and this is why the title of architect is not conferred to software developers at the beginning of their careers.

Given the importance of software architecture and its long-term consequences, it is critical that this discipline is taught to students of computer science and software engineering. However, students typically lack the experience needed to make decisions associated with the design of complex software systems. As we joke to our students: "you haven't shot yourself in the foot enough times". Due to the complex and contextual nature of software architecture, making decisions involves balancing many factors. But software architecture courses frequently rely on relatively simple examples, with relatively little context, as these are easier to describe and convey. Such examples may present the essential concepts of design, but they are also distant from real life system design experiences in terms of the limited number of design concepts, requirements, and constraints that are considered during the pedagogical design process. Furthermore, design exercises tend to require a considerable amount of time, particularly if several design iterations need to be considered, which is often a challenge in an educational setting. And, finally, feedback about the consequences of the decisions that are made as part of the


ICSE '16 Companion, May 14 - 22, 2016, Austin, TX, USA

ACM 978-1-4503-4205-6/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889184.

design process cannot usually be obtained in the short timeframe of a course. In the real world these consequences may only be felt after several months or even years of software development.

To address these issues we have created a game to more realistically simulate, teach, and stimulate discussion about the process of software design. Our game, which is called Smart Decisions, is a tool that can complement traditional lectures and exercises in software architecture courses. This game gives the players a simulated experience with the different techniques exercised in iterative design using ADD: architectural drivers, the selection of design concepts and the analysis of design decisions. A game session typically involves the simulation of several design iterations, each of which is scored, giving the students immediate feedback regarding the correctness of the decisions they have made. A score—a number—is beneficial because it is concrete and objective, but it also provides limited feedback. Hence a game session is followed by an in-depth discussion with the participants about what decisions they made and why.

Furthermore, the game is designed with a clear separation between the game mechanics on one hand, which involves the rules and steps to follow, and the game contents on the other hand, which involves a game context and a catalog of design concepts. We did this so that it would be easy to create different game scenarios involving different design problems. We have developed and published an initial offering of the game for the domain of Big Data systems. Also, we have tested this game in different settings, including playing with university students, educators, and industrial practitioners and we have received excellent feedback and encouragement from these participants. Finally, the game materials (describing the mechanics) and game content (describing Big Data design concepts) have been released publicly on the web-site smartdecisionsgame.com, as open source artifacts.

In the rest of this paper we discuss the Smart Decisions game in more detail. In section 2, we present the Attribute-Driven Design Method whose steps establish a basis for the game mechanics. In section 3, we present the details of the game starting with the motivation and the requirements (or drivers) that were used in the design of the game and the resulting game design (or architecture). In section 4, we present an evaluation of the game based on an analysis of the data and feedback that we have received in different sessions where it has been played. In section 5, we discuss related work and finally, in section 6 we conclude and present our thoughts on future work.

# 2. ARCHITECTURAL DESIGN

In this section we describe the concepts that are used in the Smart Decisions game.

## 2.1 Architectural Drivers

Architectural drivers are the input to the design process. The first type of driver that needs to be considered is the design purpose, since design may be carried out for different reasons. For example, a design might be created to support early cost and schedule estimation of the system. Other types of drivers are a subset of the system's requirements and typically include:

- **Primary use cases.** They are considered as primary as they are the raison d'être for the system. For example, in designing an ATM a primary use case would be withdrawing funds.
- **Quality attributes.** These are characteristics that define "quality" for a particular software system. They may include aspects—such as performance, availability, usability, and security—which are important to customers, or maintainability and testability—which are important to the developers. Quality attributes need to be expressed in a quantitative, objective, falsifiable way because requirements such as "the system shall be fast" or "the system shall be easy to learn" are subjective and can not be tested.
- **Constraints.** These are decisions over which the architect has little or no control. These may be technical (e.g. "use only open source technologies" or "integrate with the company's legacy LDAP server) or they may be organizational (e.g. 50% of the development must be offshored to our Bangalore office).
- **Architectural concerns.** These encompass additional aspects that need to be considered as part of architectural design but which are not expressed as traditional requirements [4]. Examples include general concerns such as creating an overall system structure, or more specific concerns such as managing exceptions or generating logs. Other architectural concerns include internal requirements, which are seldom expressed by customers, or issues resulting from analysis activities such as architectural evaluations.

## 2.2 Design Concepts

Design concepts are the building blocks for design. They are proven solutions to specific problems that appear repeatedly as part of designing software systems. These solutions can be of a conceptual or more concrete nature. The types of design concepts that we use in the game include:

- **Reference architectures.** These are blueprints for structuring systems in particular domains such as web or mobile applications. A catalog of Reference Architectures appears in [9].
- **Patterns**. These are conceptual solutions for problems at different levels of granularity and across different areas such as controlling the behavior of objects, addressing security, interoperability or deployment [7]. There is an extensive number of pattern catalogs and thousands of patterns have been documented.
- **Tactics.** These are design decisions that influence the control of a quality attribute response [1]. Examples of tactics are the use of redundancy to promote availability, or the use of record/playback to aid in testability, or the use of user models to improve usability.
- **Technology families.** These are representatives of a group of more-or-less equivalent technologies. A technology family choice can serve as a placeholder until a specific product or framework is selected. An example of this is a Relational Database or an Object-Oriented to Relational mapper.
- **Frameworks.** They are reusable software elements that provide generic functionality addressing recurring concerns across a broad range of applications [4]. An example framework is Hibernate, which is used to perform object-oriented to relational mapping in Java.

ADD 3.0, which we discuss in the next section, promotes the use of *design concepts catalogs*. These catalogs group design concepts associated to particular application domains and makes design concept selection simpler and more repeatable.

## 2.3 Attribute-Driven Design (ADD)

While there exist a number of methods that provide guidance with respect to the complete architecture development lifecycle, few are focused solely on the design activity. The Attribute-Driven Design (or ADD) method was developed by the Software Engineering Institute (SEI) in an attempt to systematize the design process . It was originally developed in the year 2000, version 2.0 appeared in 2006, and version 3.0 will be released in 2016 [4].

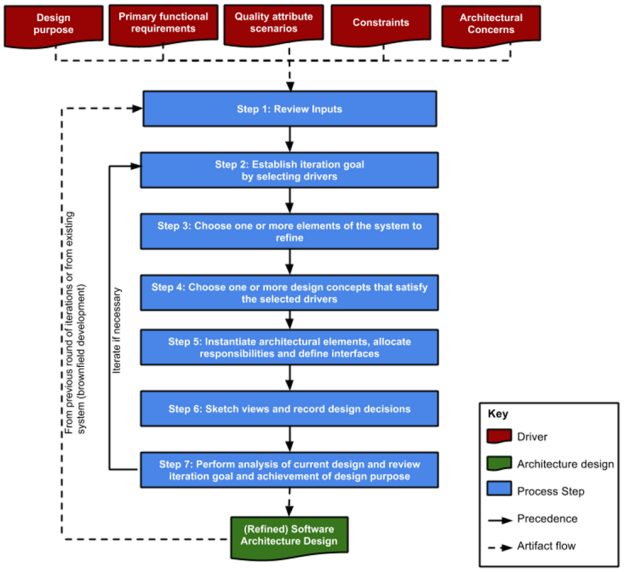


**Figure 1. Steps and Artifacts of ADD Version 3.0**

The steps of ADD 3.0 are illustrated in Figure 1. The activities performed in these steps are as follows:

**Step 1 – Review inputs.** The architect reviews and makes sure that the inputs to the design process, i.e. the drivers, are well understood.

**Step 2 – Establish iteration goal by selecting drivers.** This involves selecting a subset of the drivers that will be addressed in the iteration.

**Step 3 – Choose one or more elements of the system to refine.** Design proceeds by refinement of elements which are chosen in this step. Refinement can involve decomposing an element into sub-elements and relationships, or by further designing and elaborating elements previously identified. At the beginning of the design process of greenfield systems, the only element that can be selected is the system itself which is conceived as one big element. In further iterations, or in the design of changes for an existing system, an existing element or elements are selected.

**Step 4 – Choose one or more design concepts that satisfy the selected drivers.** This step involves the identification and selection of design concepts which will help address the drivers selected in the iteration. Design concepts can be selected from a design concept catalog if it is available.

**Step 5 – Instantiate architectural elements, allocate responsibilities and define interfaces.** Once design concepts have been selected, elements derived from them are instantiated and connected. This is where new architectural structures are created or existing ones are elaborated. For example, a new layer might be defined to address a modifiability driver, or an existing layer might be refined. Also, the responsibilities for these elements are established and the interfaces exposed by the elements are defined.

**Step 6 – Sketch views and record design decisions.** The architecture of the software system needs to be communicated to other stakeholders. While formal documentation may occur after the design process, some information should be recorded during design, when it is fresh in the mind of the architect and is less likely to be forgotten. This information includes sketches of the views that document the structures created in the previous step along with the design decisions and their rationale.

**Step 7 – Perform analysis of current design and review iteration goal and achievement of design purpose.** In this step, the architect makes an analysis of the design decisions that were made during the iteration and also of the design process as a whole. As a result, a decision is made to continue performing more iterations or to stop design and proceed to implementation.

# 3. SMART DECISIONS

In this section we describe the motivation for creating the Smart Decisions game, and the details concerning the game mechanics and game contents.

## 3.1 Why create a game?

Smart Decisions was conceived to *illustrate* important concepts associated with architectural design to a target audience of students and practitioners who do not have much experience with architecture design. Some of the key concepts that are illustrated by the game are the following:

- Architectural design is performed iteratively and it can be realized in a systematic way using a method such as ADD.
- Architectural design starts with drivers and involves making design decisions. Some of these decisions include the selection of design concepts, which is usually difficult, particularly for beginners faced with "the blank page".
- Every design concept has an influence on the drivers, and more specifically, on quality attributes.
- Design decisions have consequences: some decisions are better (hence the name "Smart Decisions") and some are worse. Also, the selection of a design concept in an iteration may have far-reaching consequences later on.
- Decisions should be recorded and analyzed as part of the design process.

The game was not conceived of as a substitute for "traditional" instruction about design, but rather a *complement* to such instruction. Consequently, we do not expect players to learn in detail how to design, or how to make optimal design decisions, by just playing the game. Instead, the game can be used a starting point for deeper discussions about the complexities of architectural design, or to practice various aspects of the design process. We have found that game participants can understand, in a short time and in an entertaining and compelling way, how design is performed and the different concepts and activities associated with this crucial activity.

## 3.2 Game drivers

Similar to the drivers that guide the design of a software architecture, we identified drivers that guided us in the process of designing the game. Our main or primary 'functional' requirement was to simulate the design process using ADD and to illustrate the concepts associated with it. Our quality attributes were the following:

- **Simplicity.** The rules of the game shall be simple to understand and to play. A new user should be able to learn to play the game with 30 minutes of instruction or less.
- **Extensibility.** The game shall support new game content to be added without changes to the game mechanics.

Finally, our constraints were that:

- No actual architectural design experience or detailed knowledge about the design concepts is needed to play the game.
- The game has to be played in a limited amount of time (1 - 2 hrs).

## 3.3 Game mechanics

Game mechanics include the game rules, steps, and scoring. Smart decisions requires a facilitator that guides the players in understanding the game mechanics using a presentation. The game requires a minimum of two "players" and a maximum of six that compete against each other. These players can be individuals or teams. The game is played in a series of *rounds* where each round represents an iteration in the design process of a greenfield system.

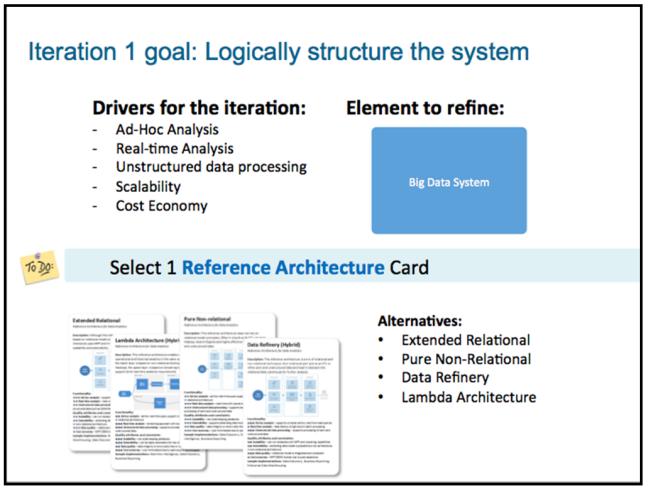


**Figure 2. Information presented at the beginning of a game round, which corresponds to an iteration of ADD**

In the first iteration, the facilitator explains how each step of ADD is reflected in the game. Step 1 of ADD (Review inputs) is only performed once, at the beginning of the game. In this step the facilitator presents the system context and the list of drivers that will need to be taken into account. Once the inputs have been reviewed, the game rounds commence. Each round begins with a description of the goal for the iteration, the drivers to consider, the element or elements to refine, the type (or types) of design concept(s) that needs to be selected along with a list of possible alternatives from which to choose from, as shown in Figure 2. This information helps to guide and structure the design step for the player and it represents the results of steps 2 and 3 of ADD which are to 'establish iteration goal by selecting drivers' and to 'choose one or more elements of the system to refine', respectively.

After the players are provided with this information, it is their turn to select design concepts from a catalog, which is embodied in a set of game cards, such as the one shown in figure 3. Each card contains information about a particular design concept and its influence (or consequences) on various drivers. The influence on drivers is rated using one to three stars where each star counts as a point. One point means that the design concept has a "low influence" on the driver (it does not contribute very much, or at all, to satisfying the driver) and three points represents "high influence" (it contributes significantly in satisfying the driver). This is a direct representation of what happens in real design—an architect would choose among design concepts based on knowledge of the strengths and weaknesses of those concepts. In the game, players select the cards using both the descriptions of the design concept and the ratings about their influences on drivers. The selection of cards represents step 4 of ADD, namely 'choose one or more design concepts that satisfy the selected drivers'.

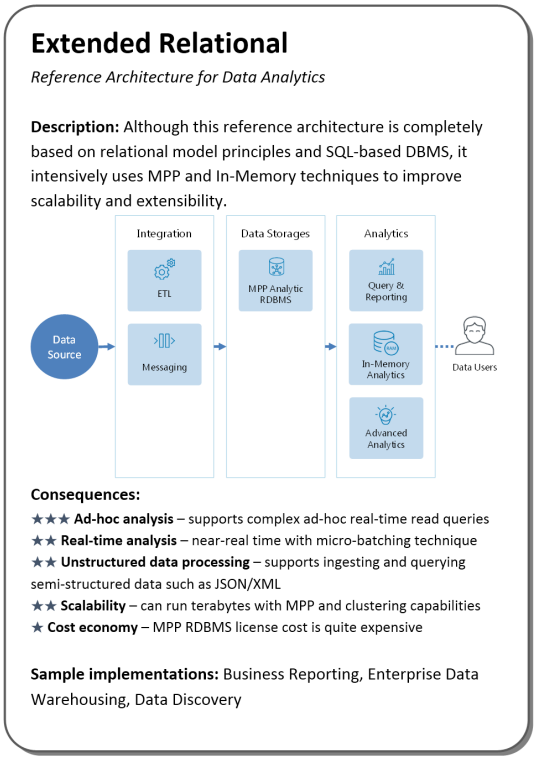


**Figure 3. A design concept card**

Once the players have selected a particular card they add points listed on the card for the drivers associated with the iteration. The resulting number represents the "fitness" of the design concept with respect to the drivers selected for the iteration. For example, if the drivers for the iteration were "(support for) Ad-hoc analysis" and "Scalability" and a player selected the "Extended Relational" reference architecture, as shown in Figure 3, they would score 3+2=5 points. This information would then be filled in row *b* of the game scorecard, as shown in Figure 4.

The next step of ADD, step 5, involves 'instantiating design elements, allocating responsibilities and defining interfaces'. Performing these tasks as part of the game would be difficult and far too time consuming, so we decided to simulate this activity through a throw of dice. A dice number between 4 and 9, the most probable result, represents a "correct" and appropriate instantiation of the selected design concept and is given no bonus or penalty points. Less probable results represent either "poor" instantiation (2 or 3) or "excellent" instantiation (10 to 12). These result in a penalty or a bonus of 2 points, respectively, which must also be recorded in the scorecard (in row *c*). Also, the players need to record their design decisions, that is the name of the design concepts that they selected, in the scorecard (in row *a*). This represents the recording of design decisions in step 6 of ADD: 'sketch views and record design decisions'.

Step 7 of ADD, 'perform analysis of current design and review iteration goal and achievement of design purpose' is guided by the facilitator who presents a list of the design concepts that could be selected for a particular iteration and bonuses or penalties associated with the selection. Bonuses and penalties represent 'good' and 'bad' decisions, respectively, and they allow the impact of design decisions to be felt by game players. For example, a penalty may occur if the player selected a design concept that was not compatible with a design concept selected in a previous iteration, or if the design concept is not compatible with the constraints presented at the beginning of the iteration. Bonuses or penalties associated with this step are recorded in row *d* of the scorecard. The first iteration analysis is performed at the end of iteration 1, as a group exercise, to explain to the players how to fill out the scorecard. The rest of the analyses, however, are performed when the players have finished performing all of their iterations. This allows the players to perform iterations at different speeds without having to wait for each other at the end of each iteration. At this point, the facilitator can promote short discussions regarding the alternatives selected by the participants.

| | Iteration #1 | Iteration #2 | Iteration #3 | Iteration #4 | Iteration #5 | |
|---|---|---|---|---|---|---|
| **(a) Design Decisions** *(Names of selected design concept(s))* | | | | | | |
| **(b) Driver selection points** *(from cards)* | | | | | | |
| **(c) Instantiation points** *(from dice)* | | | | | | |
| **(d) Analysis bonus points** *(from review)* | | | | | | **Final score:** |
| **(e) Iteration total** *(b + c + d)* | | | | | | |

**Figure 4. The game scorecard**

Once the analyses for the iterations have been performed and the players have filled in columns *a* to *d* in the scorecard, they can calculate the iteration total and the final score as a sum of the points of all of the individual iterations. Since the players are competing against each other, the first player to finish the iterations receives a bonus while the player that finished last receives a penalty. This game feature was added to reflect the reality that time-to-market is a concern for the architect.

Finally, the facilitator presents the results and promotes a discussion among the participants.

## 3.4 Game content

The game content is independent from the game mechanics. The game content consists of two elements: the system context and the set of cards, which embodies the design concepts catalog for the domain associated to the system context. Currently, we have one completed set of game content associated with the application domain of Big Data. This game content was created by architects from Softserve, a company with deep expertise in the field of Big Data and its associated technologies.

The system context for the Big Data example, which is described at the beginning of the game by the facilitator, is derived from a real system whose purpose is to collect extensive amounts of data from web server logs (a detailed description of the design of this system can be found as a case study in [4]). The system that is described supports the analysis of logs in both real-time and historical form, as illustrated by the diagram used to present the system use cases shown in Figure 5.

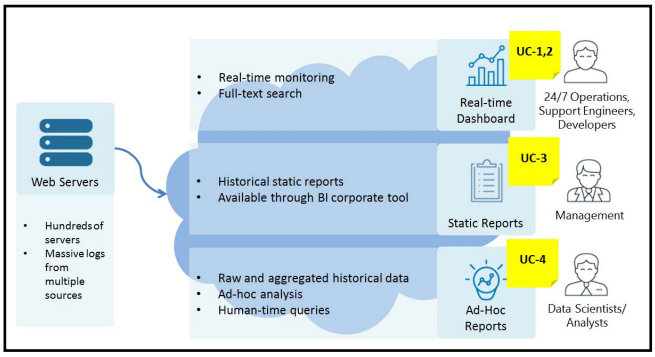


**Figure 5. Use cases for the Big Data game system context**

The system context also includes a description of the required quality attributes. These quality attributes are described as raw scenarios in the categories of Performance, Compatibility, Reliability, Extensibility, Scalability and Availability. The following are two examples of the scenarios that are presented:

- Q1 (Performance): The system shall collect 10000 raw events / sec on average from up to 300 web servers
- Q7 (Reliability): The data collection and event delivery mechanisms shall be reliable (no message loss)

In addition to use cases and quality attributes, constraints are also part of the system context, an example being the following:

- C1: The system shall be composed primarily of open source technologies.

During the presentation of the game context, quality attributes are presented in detail, including their measures, to illustrate ways in which they should be described. During game play, however, only the quality attribute categories, such as Performance, are used.

The design concepts catalog for the Big Data system is provided as a set of 28 cards, of which 11 correspond to 'abstract' design concepts including reference architectures, patterns and technology families and 17 correspond to 'concrete' design concepts, namely specific real-world technologies associated with the different technology families, as summarized in Table 1.

**Table 1. The design concepts catalog included in the Big Data game content**

| Type of design concept | Design concept card |
|---|---|
| Reference Architectures | ● Data Refinery<br>● Extended relational<br>● Lambda Architecture<br>● Pure non-relational |
| Patterns / Technology Families | ● Column-Family NoSQL Database<br>● Data Collector<br>● Distributed File System<br>● Distributed Message Broker<br>● Distributed Search Engine<br>● Document-Oriented NoSQL Database<br>● Interactive Query Engine |
| Specific technologies | ● Apache Kafka<br>● RabbitMQ<br>● Amazon SQS<br>● Apache ActiveMQ<br>● Apache Flume<br>● Fluentd<br>● Logstash<br>● Apache HBase<br>● Apache Cassandra<br>● Apache CouchDB<br>● Mongo DB<br>● Apache Hive<br>● Impala<br>● Spark SQL<br>● Apache Solr<br>● Elasticsearch<br>● Splunk |

The Big Data game content revolves around one of the reference architectures that has to be selected in the first iteration (we won't reveal which one!) to address the general architectural concern of *structuring the system*. Subsequent iterations are focused on refining the elements derived from this reference architecture, and satisfying the drivers associated with them. We are aware that the design concepts associated with a particular application domain, such as Big Data, may be unknown to the players. This could complicate the selection of alternatives during the iterations and slow down the game as players review all of the cards to figure out what to select. To address this situation, the description of each iteration goal also provides a list of alternatives to be considered (as shown in Figure 2). In this way, players only need to review a few cards in a particular iteration but the game can contain a considerable number of design concepts, reflecting the situation that occurs in real life.

It is worthwhile to emphasize that in addition to learning about the architecture design process, the participants also learn about the specific application domain of the game content.

## 3.5 Game design rationale

Table 2 summarizes the primary design decisions that we made to satisfy the game drivers.

**Table 2. Summary of the primary Design Decisions**

| Driver | Design decisions |
|---|---|
| Simulate the design process using ADD and illustrate the concepts associated with it. | ● The game is played in rounds that simulate design iterations<br>● All of the steps of ADD are represented in the game<br>● The game starts with a set of drivers<br>● A design concepts catalog is provided as a set of cards<br>● Design decisions are recorded and analyzed |
| The rules of the game should be simple to understand and to play. A new user should be able to learn to play the game with 30 minutes of instruction or less.<br>(Simplicity) | ● A facilitator explains the game mechanics and guides the players through a first iteration<br>● The game provides guidance at the beginning of iterations in terms of the alternatives to consider<br>● The influence of design concepts on quality attributes is simplified through the use of a point-based rating system<br>● Design concept cards provide only essential information and only quality attribute categories are considered (not the detailed scenarios)<br>● The first round, guided by the facilitator, lasts less than 30 minutes |
| The game shall support new game content to be added without changes to the game mechanics.<br>(Extensibility) | ● Game mechanics and game content are completely independent |
| No actual architectural design experience or detailed knowledge about the design concepts is needed to play the game. | ● The facilitator and the design concepts cards provide all the information required to play the game |
| The game has to be played in a limited amount of time (1 - 2 hrs) | ● The game is played in a limited number of iterations (5)<br>● Design alternatives that need to be considered are presented at the beginning of an iteration<br>● Step 5 of ADD is simulated through dice throws |

# 4. EVALUATION

The development of the game began in June of 2014. To gather early feedback, we presented the game concept at the ACE (Architecture-Centric Engineering) Workshop for Educators at the Software Engineering Institute in August 2014[1]. Afterwards, we performed several pilot game sessions with prototypes that were refined after each session. These pilots were performed with practitioners at Softserve in October 2014, Siemens in November 2014, and also with master's students at UAM (Universidad Autónoma Metropolitana) in March 2015. The full game was officially presented and played at the practitioner-oriented SATURN conference in April 2015[2]. In August 2015 we played the game again at the ACE Educators workshop at the SEI[3]. At that time we also published the game web-site, where the game is described and the complete set of materials can be downloaded.

To better understand the *impact* of our game, we have begun collecting data since March 2015. We have now collected data from three game sessions (1.5 hours each) including the ones with students at UAM, educators at the 2015 ACE workshop, and practitioners at SATURN 2015. In the games played by the educators and practitioners, the players were teams of people (3 and 6 player teams of 4 to 5 people respectively), while three students at UAM played as individuals.

The data that we collected includes the players' scorecards and also data from feedback forms. We have now collected 12 scorecards from the three game sessions, and we have also collected 41 feedback forms, from 28 SATURN and 13 ACE participants. A noteworthy aspect of the three game sessions that we have conducted is that the participants have represented very different groups including practitioners, students, and educators.

Due to the limited amount of data, we have only made a simple analysis of the decisions made by the players in the different iterations of the game. Table 3 shows the percentage of players that made the most "appropriate" decision in a particular iteration (note that some iterations involve two decisions). Appropriateness is established based on the information used during the analysis of decisions and these decisions may provide bonuses that are added to the players' scores. These results show that, for certain decisions (e.g. decision 1 in iterations 1 and 5), a majority of the players chose the right option, while in others (e.g. decision 2 in iteration 2 and decision 1 in iteration 3) only a minority chose the right option. This suggests that in these cases selection may be too simple or too difficult respectively, so we will need to review these decisions. Such feedback is not only useful for planning future versions of the game, it is crucial feedback to an educator, to understand which architectural decisions may need further explanation and discussion.

Interestingly, there does not seem to be any observable difference between the quality of the decisions made by the less experienced students and the more experienced educators and practitioners. However, it is worth mentioning that none of the students had in-depth knowledge about Big Data, which levels the playing field somewhat. In the end this result is encouraging, as it confirms our driver that no architectural experience (in this case, with Big Data) was required as a prerequisite to play the game.

**Table 3. Percentage of players who make the most appropriate decisions per iteration**

| It. | Number of alternatives (Decision 1) | Percentage of players who made the most appropriate decision | Number of alternatives (Decision 2) | Percentage of players who made the most appropriate decision |
|---|---|---|---|---|
| 1 | 5 | 91.7% | - | - |
| 2 | 2 | 66.7% | 7 | 33.3% |
| 3 | 3 | 33.3% | - | - |
| 4 | 3 | 83.3% | 7 | 66.7% |
| 5 | 3 | 90.9% | 7 | 54.5% |

Feedback from players is collected at the end of the game session using a questionnaire that includes the player's role (Architect, Leader, etc…), a series of ten questions, and a free text comment section for additional comments and suggestions to improve the game. The questions were answered using a scale with five options: Totally Disagree, Partially Disagree, Neutral, Partially Agree and Totally Agree. By assigning a value of 1 (Totally Disagree) to 5 (Totally Agree) to these options, we computed the mean and standard deviation for each of the questions and the results are displayed in table 4. Although the amount of data gathered at this point is relatively small, the feedback we have obtained is uniformly positive. Both the practitioner and the educator populations gave strongly positive ratings in response to all ten questions. Both populations agree on most questions, but there are two questions where responses are slightly divergent. The first concerns whether the game simulates the design process realistically; here the educators were less positive than practitioners. The second is whether the scoring technique was appropriate, with a similar discrepancy. All other things being equal one would expect that practitioners' judgments with respect to realism would carry more weight than that of educators. As to the source of the differences, we can only speculate. Perhaps the educators were simply harsher judges than practitioners. But we note that even given these differences, the responses were generally strongly positive.

**Table 4. Feedback form questions and responses collected from practitioners (SATURN) and educators (ACE). Data includes the mean and (standard deviation)**

| # | Question | SATURN | ACE | Total |
|---|---|---|---|---|
| 1 | The game helped me improve understand how to perform architectural design? | 4.18 (0.85) | 4.31 (0.61) | 4.22 (0.78) |
| 2 | The game simulates the design process realistically? | 4.46 (0.68) | 3.92 (0.73) | 4.29 (0.74) |
| 3 | The game is simple to play? | 4.04 (0.98) | 4.15 (0.77) | 4.07 (0.92) |

[1]https://www.sei.cmu.edu/community/edworkshops/2014/?location=secondary-nav&source=983706

[2] http://sched.co/2Nyn

[3]https://www.sei.cmu.edu/community/edworkshops/2015/?location=secondary-nav&source=847612

| | | | | |
|---|---|---|---|---|
| 4 | The game is sufficiently challenging? | 4.36 (1.01) | 4.31 (0.72) | 4.34 (0.93) |
| 5 | The information provided on the cards is helpful in selecting the design concepts? | 4.64 (0.85) | 4.38 (0.49) | 4.56 (0.77) |
| 6 | The scoring technique was appropriate? | 4.39 (0.86) | 3.77 (0.8) | 4.2 (0.89) |
| 7 | The example system used in the game is interesting? | 4.57 (0.82) | 4.62 (0.49) | 4.59 (0.73) |
| 8 | The game is entertaining and kept my attention? | 4.64 (0.85) | 4.62 (0.49) | 4.63 (0.76) |
| 9 | The game made me think about the process of design? | 4.46 (0.91) | 4.69 (0.61) | 4.54 (0.83) |
| 10 | I would encourage peers in my organization / team to play this game? | 4.39 (1.01) | 4.38 (0.74) | 4.39 (0.93) |

Some of the free-text comments made by the participants support the data in that the appreciation of the game is generally positive but scoring needs to be improved:

- "LOVED IT! CAN'T WAIT TO BUY IT!" (SATURN player)
- "Overall, a very interesting way to introduce concepts of attributes, trade-offs, how to think about technology etc! It will be a valuable asset for learners and practitioners can be supplemented with a lightweight catalog of choices. Thank you! " (SATURN player)
- "Fun game. Scoring was confusing at times. Sometimes hard to see if any constraints existed round to round, really nice cards." (SATURN player)
- "Good game, seems like we need a lot more information for other kinds of architectural styles. Great work will improve arch knowledge." (SATURN player)
- "Although the game doesn't encourage deep thinking about the design tradeoffs, I think it's still really helpful to reinforce the framework for reasoning about design" (ACE Workshop player)
- "First time playing skewed my results, probably better for second run! Thought provoking for me as an educator." (ACE Workshop player)

While current results are encouraging, we intend to collect additional data to attempt to better understand the factors that determine positive game outcomes and responses. Once we collect more data, we plan to refine the game further. Also, we did not gather data about *a priori* knowledge of the players with respect to ADD and Big Data but we plan to collect this information in the future.

# 5. RELATED WORK

The use of games for teaching software engineering concepts has been promoted by Kruchten and colleagues in [3] and [8]. Their games *Hard Choices* and *Mission to Mars* teach concepts related to agile software development such as the benefits and consequences of taking "shortcuts" during development, and concepts related to iteration planning. These games use a board game format and the game sessions are always followed by discussions among the participants, these ideas were reused in Smart Decisions.

There have been other proposals to use video game-like formats to teach software engineering concepts. An example of this is SimSE [10], which is a software engineering simulation environment for software process education.

Game concepts have also been used to improve actual development practices; this is the concept of *gamification*. Gamification represents the incorporation of game elements into an activity that people do not typically consider a game [12], and it is different from playing a game to teach particular concepts. In a systematic mapping study of software engineering gamification [11], the researchers noted some important gaps—that a number of critical stages of the software development lifecycle, including architecture design, have been ignored in prior gamification attempts. Although our game was not originally created with the goal of gamification of the actual design process, it should be possible to use the game materials for this purpose. For example, a development company may decide to establish a design concepts catalog that is presented as set of cards similar to the ones used in Smart Decisions, as a way of establishing a common vocabulary and common set of design primitives. During design architects could, for example, browse this design concepts card catalog and favor the selection of the design concepts that are present there. This would promote the use of common patterns and technologies in designs and would also facilitate teaching newcomers. Even though the idea of gamification of the architectural design process sounds compelling, there has also been some work examining, and questioning, the value of this practice. For example, in one study [2] reports that students in a gamified software engineering course "did not receive the gamification ideas in a positive light" and suggested improvements, although many of these had more to do with administration of the course than the game play itself.

# 6. CONCLUSION AND FUTURE WORK

In this paper we have presented Smart Decisions, a game created to teach about the process of designing a software architecture using the ADD method. The design of our game is modular: it makes a clear separation between the game mechanics and the game contents, allowing the game to be used in different application domains such as Big Data or Enterprise Applications with relatively little change.

We have released our game publicly and conducted a number of game sessions with practitioners, educators and a few students where we collected data to understand the impact of the game and to collect feedback from players. The data that we have gathered to this point is very positive, and this makes us believe that the Smart Decisions game can indeed be a useful tool to complement teaching software architecture design. One benefit of using this game is that it allows the design process to be simulated quickly, as opposed to more traditional exercises that are used in architecture design courses which can take a considerable amount of time. Here, the consequences of design decisions are

immediately tangible—the participant gets rapid feedback in terms of a score—which is not true of design in general, and this is a great pedagogical aid. Furthermore, we believe that our game is not only useful to teach about software design, but it is also a useful tool to get acquainted with the design concepts that are specific to a particular application domain such as Big Data. At this point, more game sessions need to be conducted with students and additional data needs to be collected from these sessions to better understand how much they can learn from playing the game.

Our future work includes refining the game mechanics and creating additional game content, based on the data we are collecting. Regarding game mechanics, we have received several ideas from player feedback forms, as discussed in the evaluation section. One aspect that has been requested several times is the introduction of additional dependency constraints between iterations. This means that a decision made in one iteration constrains the decisions that can be made in later iterations. Another area that has been mentioned several times by game participants is the improvement of the scoring method. While these aspects can certainly be improved, we must point out that some of the simplicity in the game is deliberate as we had to make a tradeoff between simplicity and "playability": more realism tends to add complexity which impacts negatively on how easy it is to learn and play the game. Two interesting lessons learned from this project are that 1) the development of a compelling game for teaching software engineering concepts is truly a challenging endeavor and 2) that following a method similar to the one used to design a software architecture is useful to design a game.

Regarding the game contents, for this initial release the materials were created by practicing architects who are specialists in the Big Data domain, which adds considerable realism to the game. Smart Decisions was the fruit of a successful collaboration between academia and industry. Finally, it is worth mentioning that the entire game is open source: we are open to receive contributions from the software engineering community, either new game contents, or suggestions concerning modifications to the game mechanics. The only aspect that needs to be taken into account when proposing changes to the game mechanics are the constraints discussed previously. We would also like to encourage educators to use our game and provide us with feedback about their experiences of using it with their students.

The complete set of game materials can be downloaded from http://www.smartdecisionsgame.com and a brief webinar that presents the motivations and mechanics of the game can be viewed at: https://www.youtube.com/watch?v=8LhLsunwPJ8

## 7. ACKNOWLEDGMENTS

We would like to acknowledge and thank the people who have helped in the development of this game, including:

- The many students, educators and practitioners who played the game and provided us with valuable feedback.
- Softserve, for sponsoring the development of the game cards and the smartdecisionsgame.com web site.
- The Software Engineering Institute for letting us play the game at the ACE Educators Workshop.

## 8. REFERENCES

[1] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 3rd edition, Addison-Wesley Professional, 2012.

[2] K. Berkling, C. Thomas, "Gamification of a Software Engineering course and a detailed analysis of the factors that lead to it's failure", *International Conference on Interactive Collaborative Learning*, 2013.

[3] N. Brown, R. Nord, I. Ozkaya, P. Kruchten, and E. Lim, "Hard Choices: A game for balancing strategy for agility", *Proceedings of 24th Conference on Software Engineering Education and Training*, Honolulu, HI, USA, 2011.

[4] H. Cervantes, R. Kazman, *Designing Software Architectures: A Practical Approach*, Addison-Wesley Professional, 2016

[5] H. Cervantes, P. Velasco, R. Kazman, "A Principled Way of Using Frameworks in Architectural Design", *IEEE Software*, March/April 2013, 46-53.

[6] CNN Money, "Best Jobs in America", http://money.cnn.com/gallery/pf/2015/01/27/best-jobs-2015/

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.

[8] P. Kruchten and J. King, "Mission to Mars: An Agile Release Planning Game," *Proceedings of 24th Conference on Software Engineering Education and Training*, Honolulu, HI, USA, 2011.

[9] Microsoft, *Application Architecture Guide*, 2d. Edition, Microsoft Press, 2009

[10] E. Navarro, and A. Van der Hoek, "Comprehensive Evaluation of an Educational Software Engineering Simulation Environment", *Proceedings of the Twentieth Conference on Software Engineering Education and Training*, Dublin, Ireland, July 2007.

[11] O. Pedreira, F. García, N. Brisaboa, M. Piattini, "Gamification in software engineering – A systematic mapping", *Information and Software Technology*, Vol. 57, January 2015, pp. 157–168.

[12] W. Snipes, A. R. Nair, E. Murphy-Hill, "Experiences Gamifying Developer Adoption of Practices and Tools", *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014

[13] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, W. Wood, *Attribute-Driven Design (ADD), Version 2.0*, CMU/SEI-2006-TR-023.