Journal of the ACS, Vol. 7, May 2013

ICONIX Approach to MVC: Applying Robustness Analysis on the Model–View–Controller Architecture

Ahmed El-Abbassy, Mohamed El-Zeweidy
ahmed_elabbassy@yahoo.com , melzeweidy@e-enjaz.org
Higher Institute of Computer Science &
Information Technology, El-Shorouk Academy

Abstract
ICONIX is a smart lightweight software process that is successfully used in both the academic and commercial software community. ICONIX is a use case driven software development methodology that is well suited to agile development. Due to its success, many adaptations have been proposed to fit with different contexts such as service oriented, embedded software and mobile development.
This paper investigates the application of ICONIX to the development of MVC (Model View Controller) applications. MVC is an emerging architectural pattern with objective to promote development of software that is flexible and easy to change. This is a quality requirement for most of software products and especially for web based applications.
The paper discusses and presents a proposed adaptation to ICONIX in order to help to fit with the MVC design. The proposed adaptation is illustrated by using a suitable example.

Key words:
Software Engineering, Software Development Process, MVC Architecture, UML, Object modeling, Robustness Analysis

1. Introduction

The traditional Plan-driven methods (such as PSP, TSP and RUP) have been challenged in recent years by the emergence of the Agile methods (such as Extreme Programming, SCRUM and CRYSTAL) [1, 2]. Traditional software development methodologies, usually designated as engineering methodologies, are very bureaucratic, in what documentation and rigid control mechanisms is concerned.
Agile methods afford more flexibility compared to traditional plan-driven approaches, which lock in the project details early and are less able to adjust to stakeholders' evolving needs, market changes, and unplanned technology challenges.

However in Agile methodologies, there is lack of emphasis on necessary designing and documentation.
In research work and literature, the Plan-driven methodologies are known as "Heavy-weight" methodologies or "Traditional" methodologies while agile methods are known as "light-weight" methodologies.

The ICONIX methodology is a medium-sized software development process lying between the heavy weight and the light-weight Methodologies [3].

The ICONIX process is use case driven and relatively small but it doesn't discard analysis and design like most of light-weigh methodologies do. Moreover, it makes streamlined use of the Unified Modeling Language (UML), while keeping a sharp focus on the traceability of user requirements [4]. ICONIX offers a streamlined approach software development that includes a minimal set of diagrams and techniques that a project team can use to get from use cases to code quickly and efficiently [9]. Because the process uses a minimal set of steps, it's also well suited to agile development, and can be used in tandem with test-driven development (TDD) to help "plug the gaps" in the requirements[5].
A principal distinction of ICONIX is its use of robustness analysis, a method for bridging the gap between analysis and design. This process makes the use cases much easier to design, test and estimate [6, 7].

ICONIX can be placed into the middle, between RUP, Extreme Programming (XP) and Agile Software Development. It takes the best of above mentioned methodologies and minimizes development process. ICONIX is based on UML diagrams like RUP, that's why it is so suitable for object-oriented development. ICONIX also supports iterative development and copes well with rapid changes of scope, design, requirements, and estimations. In ICONIX phases of analysis and design are exceptionally important and strongly bounded [4].

For many years the plan-driven approach was adopted to software development on Computer Studies and Software Engineering undergraduate courses. Now in both the academic and commercial software community,

the importance of an iterative and incremental approach to software development is deemed recognized [8, 9].
The development process currently in use in many undergraduate computing programs is ICONIX. ICONIX was designed with students and novice developers in mind with the purpose of providing a lightweight approach using only a subset of UML models. ICONIX provides sufficient structure, and an emphasis on the need for analysis and design upfront, as well as being scalable for the size of software development projects undertaken by students in their final year [3].

Due to its success, many adaptations have been proposed to fit with different contexts such as service oriented, embedded software and mobile development.

This paper investigates the application of ICONIX to the development of MVC (Model View Controller) applications. MVC is an emerging architectural pattern with objective to promote development of software that is flexible and easy to change. This is a quality requirement for most of software products and especially for web based applications [10].

The MVC pattern classifies the objects in three categories: view, model and controller objects. The classification criterion is given by the responsibilities of the objects from each category [11]. Successful use of this pattern isolates business logic from user interface considerations, resulting in an application where it is easier to modify either the visual appearance of the application or the underlying business logic without affecting the other [12].
This pattern decouples changes to how data are manipulated from how they are displayed or stored, while unifying the code in each component. The use of MVC generally leads to greater flexibility and modifiability. Since there is a clearly defined separation between the components of a program, problems in each domain can be solved independently. New views and controllers can be easily added without affecting the rest of the application [10].
The Model View Controller (MVC) architecture has been widely embraced as an approach for developing Web-based applications that contain a server-side programming component [13].

This paper discusses and presents a proposed adaptation to ICONIX in order to help to fit with the MVC design. The proposed adaptation is illustrated by using a suitable example.

The rest of this work is structured as follows: Section 2 presents an overview of ICONIX process and section 3 presents an overview of MVC architectural pattern. In sections 4 the proposed adaptation is discussed and an illustrative example is presented. Section 5 describes the conclusion on this topic.

2. Overview of the ICONIX Process

ICONIX is a medium-sized software development methodology whose analysis and design strength is based on UML. It is said to lie somewhere between RUP and XP. ICONIX offers a streamlined approach software development that includes a minimal set of diagrams and techniques that a project team can use to get from use cases to code quickly and efficiently [3].

Figure 1 illustrates the overall framework of the ICONIX process

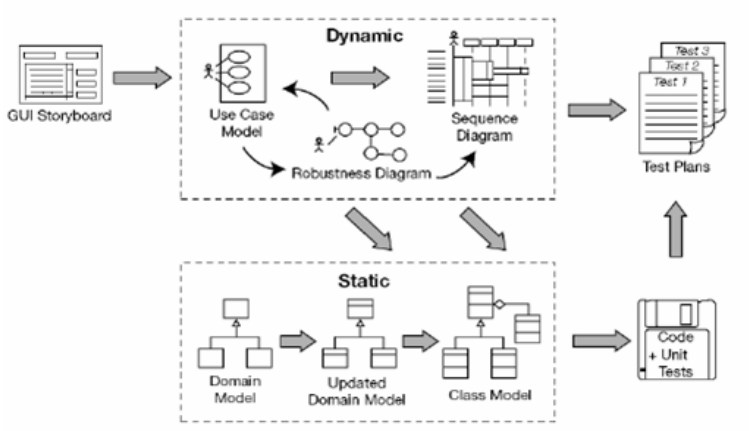
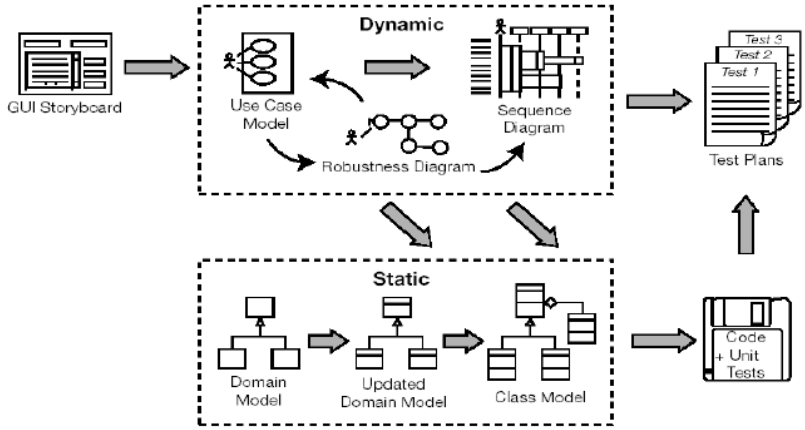


Figure 1: ICONIX Process



Following the ICONIX methodology, one has to identify the main ("real") problem domain entities, draw screen mockups and express the functionality of the application in terms of use cases using the drawn mockups and domain entities as vocabulary. The identified use cases should cover all required functionally. This ensures that the development process prioritizes a user-centered approach by expressing the functional requirements of the user in terms of a set of use cases that the final application will be tested against during system evaluation.

Based on the textual use case description, as well as the initial domain model, and with the purpose of refining the use case text and enriching the domain model with class attributes, a set of robustness diagrams are constructed bridging the gap between analysis and design.

Robustness analysis plays a central role in the ICONIX methodology. Robustness analysis allows a feasibility and sanity check of the analysis so far, which may also identify missing entities in the domain model (i.e., object discovery) and thus reassures that design is based on a solid set of participating classes. On a robustness diagram, actors are represented by stick-persons, application screen by boundary objects, system processing by controllers and domain classes by entity objects.

The next phase of the ICONIX methodology proceeds to the detailed design level of the application by constructing a sequence diagram for each robustness diagram (and thus for each use case).

Sequence diagrams contain all actors, screens and domain classes from the respective robustness diagrams and allocate behavior to them by translating each control object to one or more messages exchanged between actors, screens and domain classes.

This process results in assigning methods to classes in the domain model and eventually transforms the initial domain model (which was revised in the robustness analysis phase and extended with class attributes) to a detailed design level class diagram that can drive the generation of the application coding [3, 5].

The approach in a Nutshell [14,15]

1. Step1
   1.1 Identify your real-world objects and the generalization + aggregation relationships between them (Domain model static diagrams)
   1.2 (Possibly perform a rapid prototyping of the proposed system)
   1.3 Identify your use cases (Use Case Diagrams)
   1.4 Organize the use cases into package diagrams
   1.5 Allocate functional requirements to use cases and domain objects

Milestone 1: Requirements Review

2. Step2
   2.1 Write a description of the use cases, both "mainstream" + alternative courses
   2.2 Perform Robustness Analysis:
      - identify objects which accomplished a stated scenario
      - update the domain model with this objects
      - Finish the analysis class diagram

Milestone 2: Preliminary Design Review

3. Step3
   3.1 Allocate behavior. For each use case: (Sequence diagrams)
   3.2 identify the messages between different objects
   3.3 (if needed collaboration diagrams to show key transactions between objects)
   3.4 (state diagrams to show the real-time behavior)
   3.5 Finish the static model with the detailed design (Detailed design static diagrams)

Milestone 3: Detailed Design Review

4. Step4
   4.1 (If needed produce deployment + component diagrams to help in the implementation phase)
   4.2 Write/generate code
   4.3 Perform unit + integration testing
   4.4 Perform system + user-acceptance testing

Milestone 4: Delivery

3. The MVC Architectural Patten

In general, the software is designed by applying an architectural pattern. This describes the kind of components, their relations, their constraints, the design and the composition rules of the components [11].
One of the most common software architecture design pattern is the Model View Controller (MVC) paradigm [16].
According to the MVC software pattern, the application should have at least three components as illustrated in figure 2. The Model component includes the core of application data and logic domain functionality. The View obtains data from the Model and displays them to the user. The Controller receives and interprets input into the requirements for the Model or the View. One of the first uses of the MVC software pattern in object oriented software applications was in the SmallTalk programming language [14].

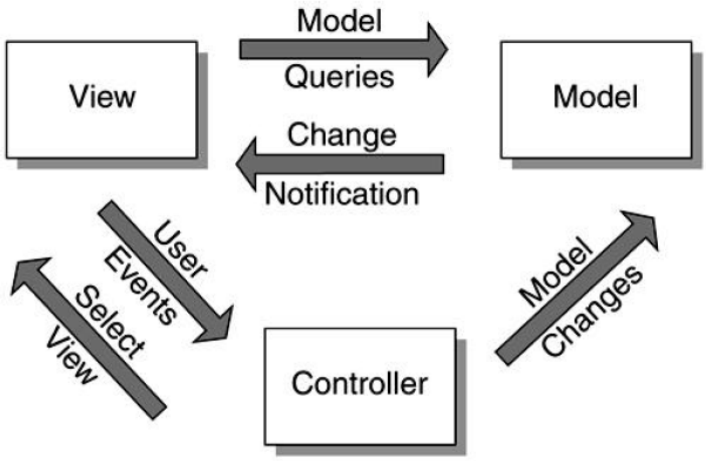


Figure 2: MVC Software Pattern

The architectural pattern has wide acceptance in the software development of GUI applications. GUI models the interface as a composition of interacting objects that present the internal state to the user [17]. MVC appears popular because it directly applies an object-oriented approach to separate the components of an interactive system. Specifically, as the name

implies, the components are the model, the view and the controller. Separation of the components allows for independent development, testing and maintenance of each component [16].

Figure 3 illustrates an MVC architecture created using Java-based technologies. In this architecture, user HTTP requests are routed through a controller, which is typically implemented as a servlet [13].

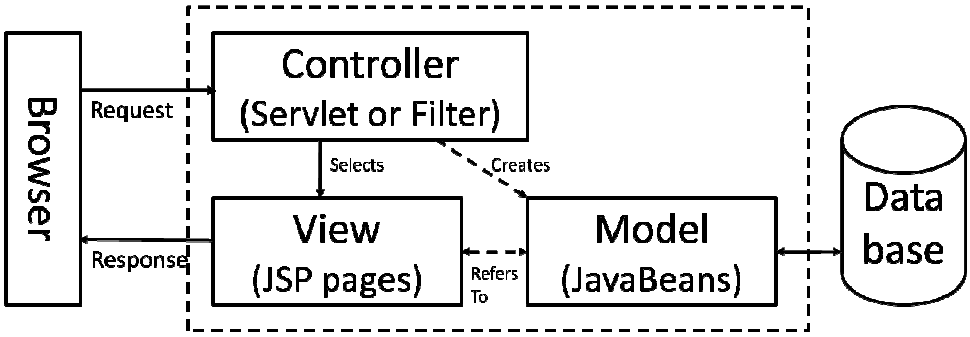


Figure 3 MVC using Java-based technologies

This approach effectively separates the program code and HTML code and allows the programmers and designers to work far more independently. The programmers can write the controller and model code without interleaving the HTML, which is found in the view. Similarly, the view contains far less programming code than in non-MVC approaches. Although the JSP Expression language and tag libraries are a form of coding, they are less intrusive than adding scriptlets and directives to a JSP page [13].

The MVC basic idea is to minimize the coupling among objects in a system by aligning them with a specific set of responsibilities in the area of the persistent data and associated rules (Model), presentation (View), and the application logic (Controller) [18].
The solution suggested by the model-view-controller pattern is to localize the responsibilities of

- hosting business logic and data,
- presenting information to users and

- reacting to user events

on separate, loosely coupled, components called the model, the view and the controller. This is illustrated in Figure 4.

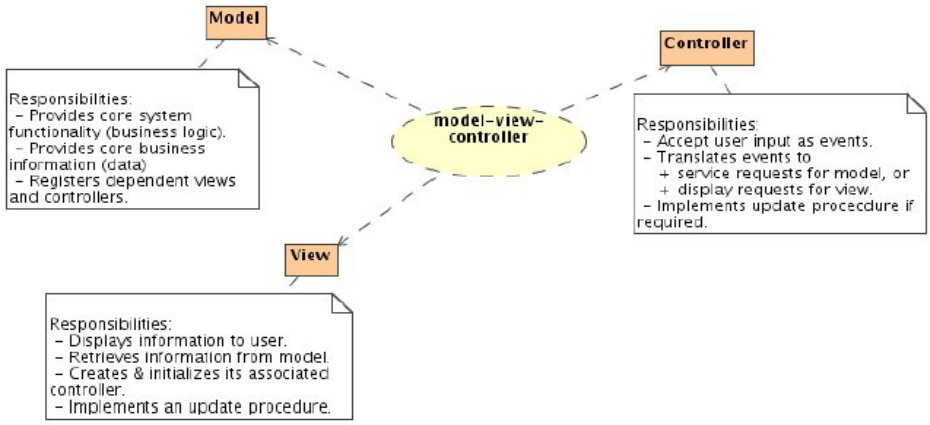


Figure 4 Responsibility allocations in the MVC pattern

Major Benefits and Potential Problems

Benefits

- Simpler maintenance.
- Supports concurrent modular development with clearly defined developer roles (view designers, front-end developer, back-end developer, ...).
- Simpler testing.
- Reusable business logic and presentation layer components.
- Supports multiple synchronized views on same data.
- Business logic and presentation/controller may be hosted on different nodes (machines).

Potential problems

- In distributed implementations the communication overheads may lead to performance problems.
- Both, the controller and the view are dependent on the model API which may reduce re-usability of view and controller elements.

4. The Proposed Adaptation

4.1 Robustness Analysis:

Ivar Jacobson introduced the concept of robustness analysis to the world of OO in 1991 [4]. It is an intermediate level of design, between Use Cases and the software design level. Robustness analysis acts as a mediator to bridge the gap between modeling use case diagram and sequence diagram [17].

As a non-core element of UML, robustness analysis bridges the gap between analysis and design in software process, and it is a key technique to realize transformation and trace ability of models in ICONIX software process [19]

By analyzing each use case, robustness analysis identifies a set of objects that will participate in the use case, and classifies them into one of three stereotypes as shown in Figure 5:

(1) Boundary objects, which the actors use when communicating with the system

(2) Entity objects, which are usually objects from the domain model

(3) Control objects, which server as the "glue" between boundary objects and entity objects

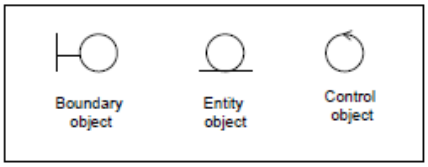


Figure 5 Robustness diagram Stereotype symbols

Entity objects represent data stored in a database. Boundary objects are deemed user interfaces and triggered by users to communicate with control objects, which capture application logics and act as bridges between boundary objects and entity objects.

As illustrated in figure 6, the interaction rules among these objects can be summarized as follows:

• Actors can only communicate with boundary objects.

• Boundary objects can only communicate with control objects and actors.

• Entity objects can only communicate with control objects.

• Control objects can communicate with boundary objects, entity objects and the other control objects, but not with actors.

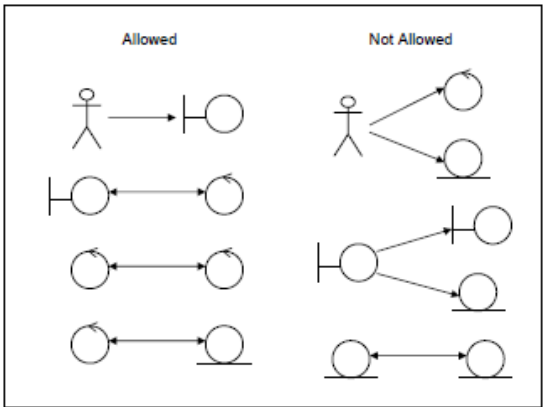


Figure 6 Robustness diagram interaction rules

When applying robustness analysis for Web-based systems we find:

(1) Boundary objects are the objects that the users will use to interact with the system. These are elements that compose a web page, such as hypertext, forms, menus, buttons, and so on.
(2) Entity objects often map to the database tables and elements in legacy systems. They represent resources required by use case execution.
(3) Control objects embody mostly application logic. They serve as mediator between the users and the stored data. This is where one captures the frequently changing business rules and policies.

The principles are the same as those underlying the component-based reference model for web-based systems presented in [20]:

Presentation component ~ Boundary object
Control component ~ Control object
Resource component ~ Entity object

4.2 Role of Robustness Analysis in ICONIX Methodology

Robustness analysis plays several essential roles within the ICONIX process.

Robustness analysis fills the role of preliminary design, by closing the gap between analysis and detailed design as illustrated in figure 7. Robustness analysis is really preliminary design, during this phase, you start making some preliminary assumptions about your design, and you start to think about the technical architecture and to think through the various possible design strategies. So it's part analysis and part design.

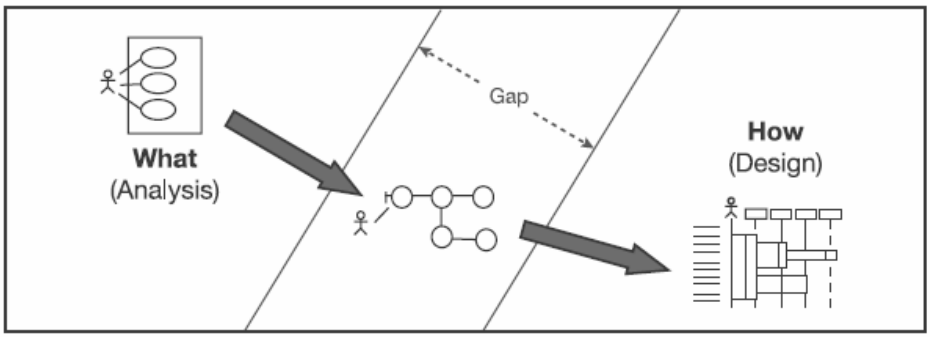


Figure 7 Role of Robustness analysis within the ICONIX process

A robustness diagram shows conceptual relationships between objects. Because it's an "object drawing" of the use case text, it occupies a curious space halfway between analysis and design. Nevertheless, mastering robustness analysis is the key to creating rigorous designs from clear, unambiguous use cases.

4.3 Robustness Analysis and Model-View-Controller:

MVC can be best described visually, using robustness analysis. As illustrated in figure 8, we can see how robustness diagram objects are related to MVC paradigm. MVC have one-to-one mapping with the objects, derived from robustness analysis [21]:

- Entity object maps to Model object,
- Boundary object maps to View object, and
- Controller is same in both.

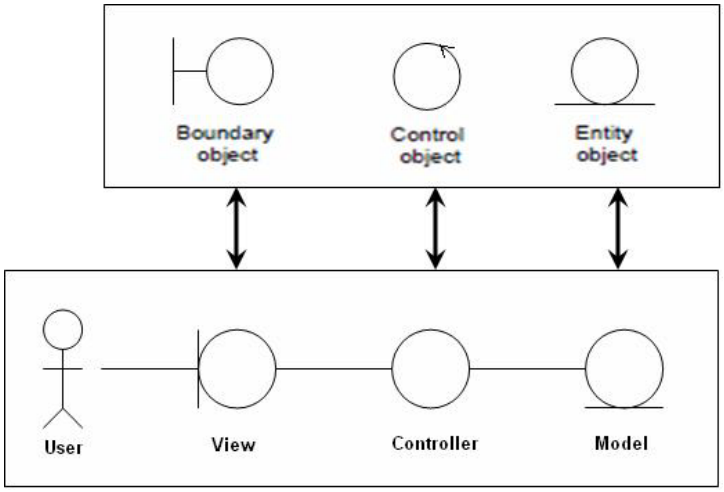


Figure 8 Robustness analysis and MVC objects relationship

This means, when we are doing Robustness Analysis, we can use Model-View-Controller objects in place of Entity-Boundary-Controller objects.

4.4 The proposed adaptation

It is clear from the analysis of models that robustness diagram is the most suitable model to represent MVC programs.
So we propose to use the robustness model in the detailed design phase for MVC systems.
By this way robustness diagrams will be used in both the preliminary design and the detailed design phases.
With this proposed extension to the robustness analysis role, the overall framework of the ICONIX process can be revisited as shown in figure 9

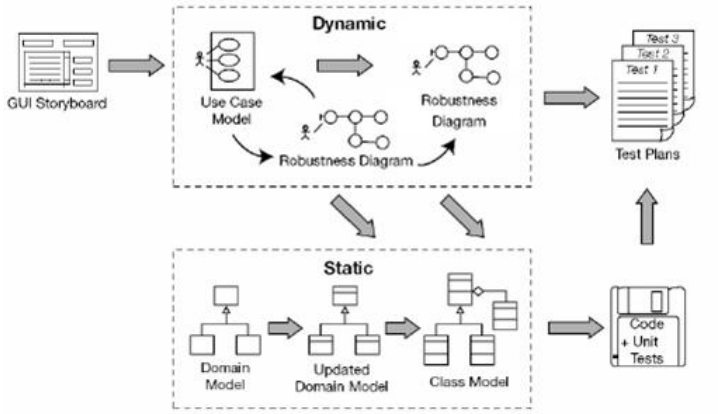


Figure 9: The ICONIX process revisited

Robustness analysis will fill the role of preliminary design and detailed design as shown in figure 10

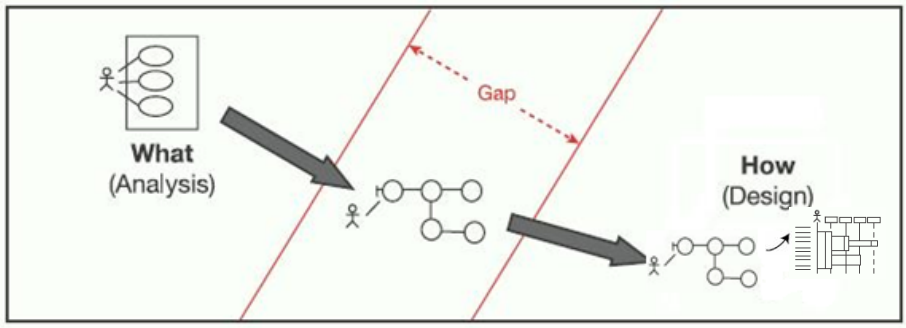


Figure 10: New Role of Robustness analysis within the ICONIX process

4.5 Example

In the following example we present how robustness analysis may be used in the detailed design.
To create the sketch I merely followed the logic of the use case and applied the following heuristics were used to apply robustness analysis in the detailed design:

- Add a boundary element for each major user interface element such as a screen or a report.
- Add a controller to manage the overall process of the scenario being modeled.
- Add a controller for each business rule.
- Add a controller for activities that involve several other elements.
- Add an entity for each business concept.
- Add a use case whenever one is included in the scenario

Example: Robustness Analysis of student registration use case
Figure 11 illustrates the detailed design of the student registration use case.

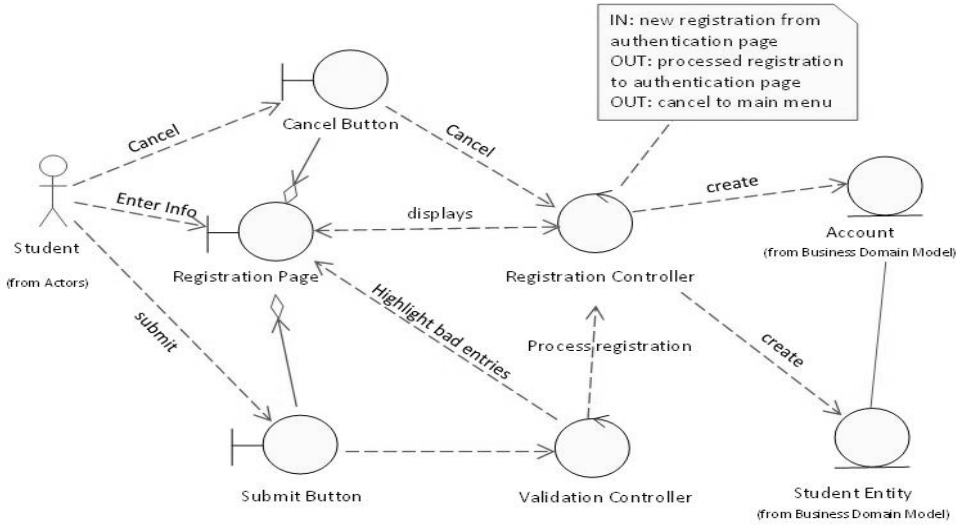


Figure 11: Student registration detailed design using robustness diagram

Figure 12 illustrates the object discovery during the detailed design corresponding to the student registration use case

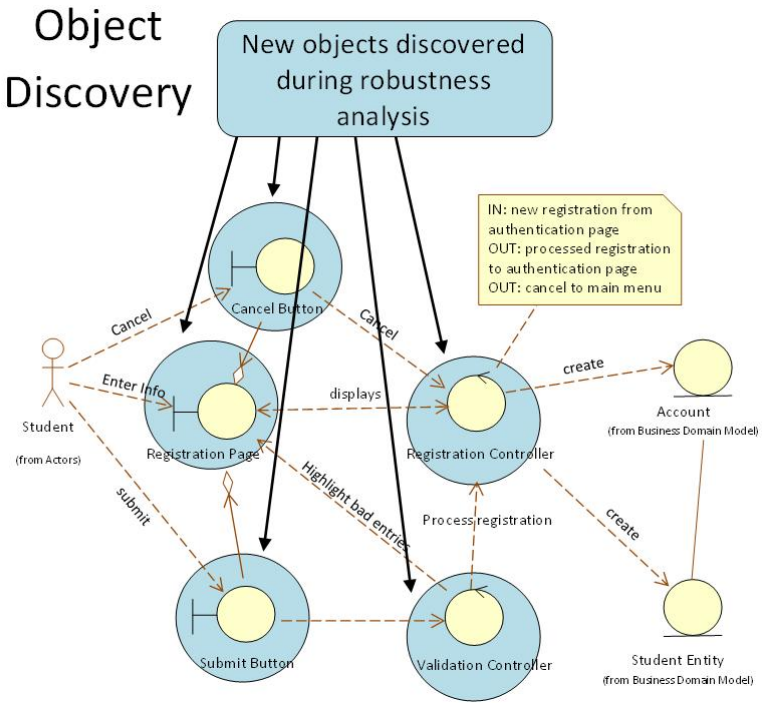


Figure 12: Object discovery during the detailed design process

4.6 Comparative analysis and evaluation

In this section we discuss the importance of robustness analysis with respect to MVC design pattern and the ICONIX modeling process. We also provide a comparative analysis between robustness diagrams and sequence diagram which are the main behavioral models used during ICONIX design phase.

4.6.1 Robustness Analysis and Model View Controller design Pattern

Robustness diagram is the most simple and important Model-View-Controller (MVC) diagram in the software business.

Model-View-Controller can be best described visually, using robustness analysis: they have one-to-one mapping with the objects and the same rules are applied to these objects too. Robustness analysis model helps to partition objects within a Model-View-Controller paradigm and we can use Model-View-Controller objects in place of Entity-Boundary-Controller objects.

Robustness diagram will also be good to discover any missing object and overview of the MVC communication.

A robustness Diagram succinctly tells us how to implement an MVC design in code. As illustrated in figure 13, this diagram states the following MVC pattern rules:

- Users interact with View objects.
- View objects and Controller objects talk to each other.
- Different Controller objects talk to each other.
- Controller objects talk to Model objects.
- No other forms of communication between objects are allowed.

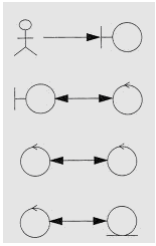


Figure 13: MVC allowed communication

If we understand these rules and commit to them, MVC concept becomes very simple.

4.6.2 Robustness Analysis and ICONIX modeling process

Robustness analysis streamlines the ICONIX modeling process, and delivers the following added values:

- Use the DDT (Design Driven Testing) process.

We have the flexibility to test against Requirements, against Controllers ("logical functions") and to do true unit testing at the Sequence message level. All of these abstraction levels should be verified by our testing.

- Generation of robustness diagrams from use cases:

Having the Requirements on the Robustness diagram helps us to make sure we haven't forgotten any Requirements as we analyze the use case.

- Generation of sequence diagram structures from robustness diagrams

In addition to being useful during Requirements definition and Analysis/Conceptual Design phases of a project, the Robustness model delivers value when transitioning from the Analysis phase to Detailed Design, by generating skeleton Sequence diagrams from Robustness Diagrams. All Boundary, controller, and Entity objects are brought from the Robustness diagram onto the Sequence diagram. Once the Sequence diagram is completed, we can generate unit tests from the messages on the Sequence Diagram. Note that "controllers" on Robustness diagrams tend to reflect a slightly higher abstraction level than Sequence messages; that is, a controller may be implemented as multiple messages on a Sequence diagram.

- Transformation of robustness control elements to test diagrams:

By analyzing our use case using the Robustness Analysis technique, we've added a lot of information to the model that can be used to drive the testing of the application. We can generate test cases for each "controller" on the Robustness diagram. The "controller tests" are performed by the development team while an independent QA team might own the "requirement tests". We can easily generate test plan reports. So driving our testing activity from the model is very straightforward.

4.6.3 Robustness Diagrams versus Sequence Diagrams

A robustness diagram is an interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.

a) Robustness Sequence and Sequence diagram Similarities

- Semantically equivalent

- Can convert one diagram to the other without losing any information
- Model the dynamic aspects of a use-case scenario
- Sequence diagrams and robustness diagrams are semantically equivalent: Both sequence and robustness diagrams allow you to capture semantics of the use-case flow of events. They help identify objects, classes, interactions, and responsibilities, as well as validate the architecture.
- As a result, you can take a diagram in one form and convert it to the other without any loss of information.

b) Robustness Sequence and Sequence diagram differences

- Robustness diagrams emphasize the structural communication of objects and show a clearer picture of the pattern of relationships and control that exist among the objects participating in a use case. Robustness diagrams also show more structural information, such as the relationships among objects. Robustness diagrams are better for understanding all the effects of a given object and for procedural design.
- Sequence diagrams show the explicit sequence of messages and are better for real-time specifications and complex scenarios. A sequence diagram also includes chronological sequences but does not include object relationships. On sequence diagrams, the time dimension is easier to read, the operations and parameters are easier to present, and the larger number of objects are easier to manage than in robustness diagrams.

5. Conclusion

This paper investigated the application of ICONIX, a use case driven, object-oriented analysis and design methodology that makes streamlined use of UML, for the development of MVC applications.

The study concluded that by extending the role of robustness analysis to cover both the preliminary and the detailed design phases, the adaptation of the ICONIX methodology to MVC is easily recognized.

6. References

[1] Sutap Chatterjee, (2010), "The Waterfall That Won't Go Away", ACM SIGSOFT Software Engineering Notes, Volume 35 Number 1

[2] Li Jiang, and Armin Eberlein, (2008), "Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies", APSO'08, Leipzig, Germany.

[3] DOUG ROSENBERG, 2010, ICONIX PROCESS ROADMAPS, Fingerpress LTD Londen

[4] Evanthia Faliagka, Petros Karkoulias, Maria Rigou2, Spiros Sirmakessis, Giannis Tzimas, and Athanasios Tsakalidis, (2012), Applying an OO Modeling Methodology for the Design, Implementation and Testing of a Smart Phone, Received: November 07, 2011 / Accepted: November 18, 2011 / Published: February 25, 2012, Computer Technology and Application 3 (2012) 105-113, David Publishing.

[5] Doug Rosenberg, Matt Stephens and Mark Collins-Cope , (2005), Agile Development with ICONIX Process, Methods & Tools, Global knowledge source for software development professionals, ISSN 1661-402X, Summer 2005 (Volume 13 - number 2)

[6] Bhushan Thakare, Bhushan Bhokse, Laxmi Thakare , (2012), Deriving Best Practices from Development Methodology Base (Part 1), International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 1 Issue 6, August – 2012.

[7] Bhushan Thakare, Bhushan Bhokse, Laxmi Thakare , (2012), Deriving Best Practices from Development Methodology Base (Part 1), International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 1 Issue 6, August – 2012

[8] Bernard Chalk, Karen Fraser, (2005), A Survey on the teaching of introductory programming in Higher Education, ICS Higher Education Academy Website in September 2005.

[9] Mark Cranshaw1, John Flackett, (2006), Using process miniatures as an aid to teaching software development in Java, The 10th Java & the Internet in the Computing Curriculum Conference (JICC 10), held at GC108, North Campus, London Metropolitan University on Friday 3rd February 2006.

[10] Banani Roy and T.C. Nicholas Graham, (2008), Methods for Evaluating Software Architecture: A Survey, Technical Report No. 2008-545, School of Computing, Queen's University at Kingston, Ontario, Canada

[11] Alexandru Florin Pavel, Crengu¸ta M˘ad˘alina Bogdan , (2008), Object-Oriented Construction of Portals Using AJAX, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844, Vol. III (2008), Suppl. issue: Proceedings of ICCCC 2008, pp. 442-447

[12] Yanfang Wang, Chunyan Guo, Lei Song , (2009), Architecture of E-Commerce Systems Based on J2EE and MVC Pattern, 2009 International Conference on Management of e-Commerce and e-Government

[13] Nick Heidke, Joline Morrison, and Mike Morrison , (2008), Assessing the Effectiveness of the Model View Controller Architecture for Creating Web Applications,

[14] D. Rosenberg, M. Stephens, M. Collins-Cope, (2005), Agile Development with ICONIX Process: People, Process, and Pragmatism, Apress, 2005.

[15] Morteza Poyan rad, Homayon Motameni, (2011), Improving Web Engineering and Agile Iconix Process, Middle East Journal of Scientific Research 8(1): 274-281, 2011, IDOSI Publications, 2011.

[16] Thuan Pham, (2010), Model-View-Controller Design, University of Washington – Bothell, CSS 555 - Evaluating Software Design, Mark Kochanski, May 16, 2010

[17] Wu J-H, Shin S-S, Chien J-L, Chao WS, Hsieh M-C (2007) An Extended MDA Method for User Interface Modeling and Transformation. In Proceedings of the Fifteenth European Conference

on Information Systems (Österle H, Schelp J, Winter R eds.), 1632-1642, University of St. Gallen, St. Gallen.

[18] Khawar Zaman Ahmed, Cary E. Umrysh, 2001, Developing Enterprise Java Applications with J2EE and UML, Addison-Wesley Pub Co; ISBN: 0201738295; 1st edition(December 15, 2001)

[19] Yang, Deren, Su, Fulin ; Zhou, Tao , (2012) , Applying robustness analysis to MDA software paradigm, This paper appears in: Instrumentation & Measurement, Sensor Network and Automation (IMSNA), 2012 International Symposium on Date of Conference: 25-28 Aug. 2012, Volume: 2, Page(s): 419 - 422

[20] Philippe Dugerdil, Javier Belmonte, and David Kony, (2009), Using Robustness Diagrams to Help With Software Understanding: an Eclipse Plug-in, Int.J. of Software Engineering, IJSE Vol.2 No.3 December 2009

[21] Shams Mukhtar, (2004), "Applying Robustness Analysis on the Model-View-Controller (MVC) Architecture in ASP.NET Framework, using UML", www.codeproject.com August 2004