Towards Executable Domain Models

Nitish Patkar
Software Composition Group
University of Bern, Switzerland
http://scg.unibe.ch/staff/NitishPatkar

Abstract

Lack of stakeholder commitment to project activities is an important cause of project failure. It leads to poor understanding of the problem domain and ultimately to incorrect domain modeling. We believe that combining *Goal Directed Design* methodology with *Domain Driven Design* using the *Naked Objects Pattern* would help stakeholders to become actively involved in important project activities such as requirements elicitation and rapid prototyping. This paper suggests how these approaches can gracefully be combined to facilitate active stakeholder involvement right from requirements elicitation until rapid prototyping to build a clear product vision, and realize it through an executable domain model.

1 Introduction

For software to be useful and usable, it must correctly reflect business needs, and that can only be achieved when the problem domain is sufficiently understood by all the stakeholders [Ver13, Bjo06]. This inherently demands active non-technical stakeholder participation and collaboration in project-related activities such as requirements elicitation and prototyping. Without non-technical stakeholder participation, the developers alone cannot accurately uncover the corresponding problem domain, which further makes it difficult to accurately and clearly formulate the software requirements. This ultimately hinders creation of an effective domain model.

The requirements elicitation phase of requirements engineering is mainly responsible for uncovering the requirements and the domain knowledge. The current requirements elicitation process faces several challenges such as ineffective stakeholder meetings, stakeholders focusing on solutions rather than underlying problems. Another challenge is that the complete domain knowledge might not be available right from the beginning, but has to be incrementally discovered and realized in software. Involving stakeholders directly in project activities also raises an important question, *how* can a domain model be represented to which each stakeholder can relate?

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2018 (sattose.org).
04-06 July 2018, Athens, Greece.

Two interesting approaches, when combined together, can answer the above question and address the challenge of stakeholder participation. We believe that combining the *Goal Directed Design* (GDD) process and *Domain Driven Design* (DDD) would help to uncover quality domain knowledge, and to rapidly build domain models through stakeholder collaboration.

The GDD process, introduced by Alan Cooper [CRC07], argues that most digital products fail because they fail to understand stakeholder and end-user goals, and hence put those at the center of requirements elicitation. It encourages requirements analysts to gather user goals and as much contextual data as possible to clearly understand stakeholders' vision of the software system. *Vision Backlog* [Pat18]—both a concept and a tool that leverages GDD—helps stakeholders to list their goals and provide contextual information on their own. From this information, it is possible to uncover the domain knowledge and identify domain entities as demonstrated later in section 4. On the other hand DDD, coined by Eric Evans [Eva03], puts the domain knowledge at the center of design and development activities. It encourages stakeholders to work closely to understand and model the problem domain correctly. Architectural styles like the *Naked Objects Pattern* [PM01] makes it easier to practice DDD.

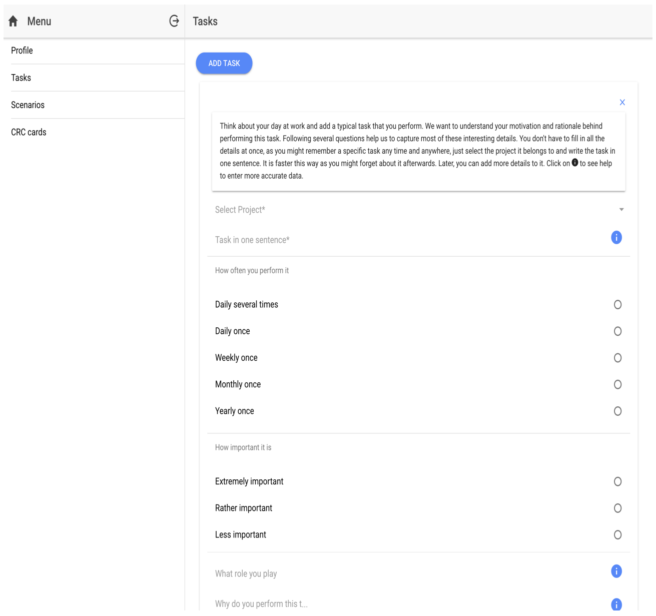
This paper demonstrates how these two approaches can gracefully be combined to facilitate active stakeholder involvement right from requirements elicitation until rapid prototyping in order to build a clear product vision and realize it through an executable domain model. Section 2 explores concrete problems in current requirements elicitation that hinders the stakeholder involvement and introduces a solution *Vision Backlog* and also a few other techniques. Section 3 explains the general idea behind executable domain models and lists various approaches which go in that direction such as DDD, Exploratory Modeling. It also shows how DDD using Naked Objects Pattern can help to build rapid prototypes that reflect the actual underlying domain model. Section 4 demonstrates the workflow that can be followed and the tools that can be used to combine GDD and DDD, and lists its benefits. Section 5 presents our plan to improve the stakeholder participation to uncover the domain knowledge and model it effectively, and further lists the research questions which we will be looking at. Section 6 summarizes our ideas and contribution.

2 Building a product vision

The stakeholders' contributions in understanding the problem domain and forming requirements are vital, as they have the clearest idea about their own needs and expectations of the perceived system [ZC05, Yoz14]. Requirements Engineering (RE) activities, such as elicitation, strive to uncover domain concepts and stakeholder goals. They are iterated until the problem domain is thoroughly understood and documented [ZZ07].

The current requirements elicitation process, however, has its own shortcomings which eventually compromises the result of elicitation. First, non-technical stakeholders possess very little knowledge of the elicitation process. It requires special skills and among all the stakeholders only requirements engineers have those [Yoz14]. There is a risk of requirements engineers becoming a bottleneck if stakeholders start depending on them for elicitation. Second, elicitation is traditionally done mostly through formal meetings which are always a tough choice as the stakeholders are relatively busy. The results of these meetings are heavily affected by the fact that people are mostly not able to express themselves clearly when asked direct questions [MT13, Bea01]. Another challenge is that the stakeholders tend to focus on probable solutions rather than the underlying problems [ZC05]. These shortcomings eventually lead to a poor understanding of the problem domain by requirement engineers and developers, and affects the domain model that would be built.

Vision Backlog, a concept and a tool that was built during our previous research [1], tackles these obstacles by encouraging stakeholders to describe on their own their goals, motivations, expectations *etc.* This tool is built as a web application that stakeholders can use on the go. It gathers information about the tasks the stakeholders perform, why they perform them, the tools, techniques and knowledge they require to perform those tasks, information about the people involved, existing workflows, and flaws in the current software system—if it exists—with possible improvements. It serves as an assistant system to the requirements engineer that gathers stakeholder goals and surrounding contextual information outside of meetings. This information is then presented as analytics from which domain entities and their relationships can easily be identified. This information forms the initial product vision directly contributed by stakeholders. It can even further assist in building personas[2].



Think about your day at work and add a typical task that you perform. We want to understand your motivation and rationale behind performing this task. Following several questions help us to capture most of these interesting details. You don't have to fill in all the details at once, as you might remember a specific task any time and anywhere, just select the project it belongs to and write the task in one sentence. It is faster this way as you might forget about it afterwards. Later, you can add more details to it. Click on ❶ to see help to enter more accurate data.

Select Project*

Task in one sentence*

How often you perform it

Daily several times ○
Daily once ○
Weekly once ○
Monthly once ○
Yearly once ○

How important it is

Extremely important ○
Rather important ○
Less important ○

What role you play

Why do you perform this t...

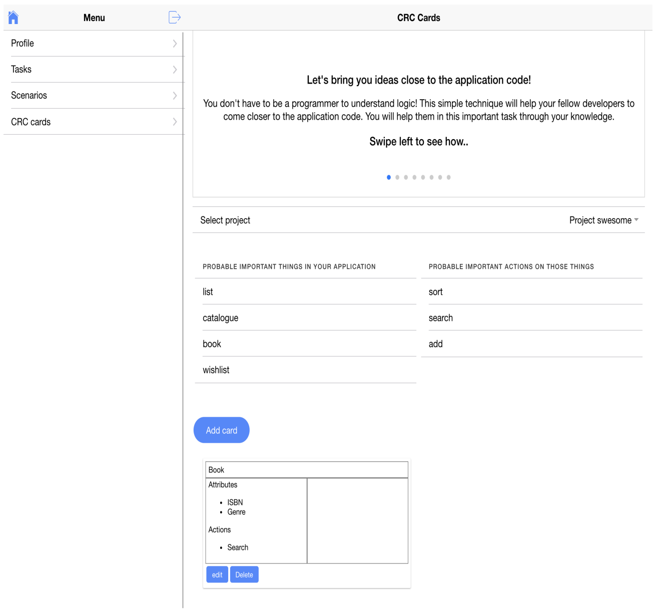
Figure 1: Vision Backlog adding a task

Vision Backlog is currently only a prototype (see Figure 1 and Figure 2), which was tested for its usability and usefulness, and the results were largely positive, suggesting only a few improvements in its UI. It largely reduces the frequency of meetings, allowing analysts to ask fewer, but more specific questions based on the knowledge gathered by stakeholders in Vision Backlog.

Apart from Vision Backlog, a few other techniques can also help to uncover and structure the domain knowledge. CRC-card modeling technique from Responsibility-driven design (RDD) [WBW89] is used for brainstorming to develop a clear understanding of the domain classes and their responsibilities. Six hat thinking technique [DB17] helps to un-bundle thinking and explore different perspectives on the problems.

[1]http://scg.unibe.ch/archive/external/Patk18a.pdf

[2]A Persona represents a user model. How users behave, how they think, what they wish to accomplish *etc.* Personas are not real people, but they are based on the behaviors and motivations of real people we have observed. Persona also serves as a powerful communication tool between developers, managers and other stakeholders that helps them to understand the logic behind design decisions.



Let's bring you ideas close to the application code!

You don't have to be a programmer to understand logic! This simple technique will help your fellow developers to come closer to the application code. You will help them in this important task through your knowledge.

Swipe left to see how..

Select project: Project swesome

PROBABLE IMPORTANT THINGS IN YOUR APPLICATION	PROBABLE IMPORTANT ACTIONS ON THOSE THINGS
list	sort
catalogue	search
book	add
wishlist	

Book
Attributes
- ISBN
- Genre

Actions
- Search

Figure 2: Vision Backlog using natural language processing to extract entities and methods

3 Making domain models executable

To bridge the communication gaps between business and IT, the role of *Business Analyst* has long been introduced to assume the responsibility to understand both IT and business aspects of a project. In the United States alone, 130 000 professionals are employed as IT Business Analysts [Len15]. To assist Business Analysts in their work, frameworks like Enterprise Architect [Ent10] have been used to let them model the system using more than 15 different UML diagrams.

A model, or specifically a domain model, is not a specific diagram but a set of concepts and relationships between them. A good domain model would capture an abstraction of the business in a form defined well enough to be coded into software [Eva03]. Even better, it should be possible to represent domain concepts as classes and interfaces easily demonstrable back to non-technical stakeholders. A working prototype built out of such a domain model is a true representation of a model, which every stakeholder can relate and respond to. Since this model forms an integral part of the running software system—unlike, say, static UML diagrams—we call it an *executable domain model*.

One of the approaches to build such executable domain models is to use Domain Driven Design along with Naked Objects Pattern. DDD places vigorous efforts in understanding the problem domain and encourages non-technical stakeholders and software developers to work closely. DDD encourages all the stakeholders to collaboratively decompose the monolithic problem domain into smaller domains until each of them can have its own *ubiquitous language* with no ambiguous terms. A ubiquitous language consists of the concepts used in the corresponding problem domain, and those concepts should literally be reflected in the code as classes and interfaces [Eva03]. DDD expects the domain model to have a literal and straightforward representation in the design and implementation of the system, that is if the code changes the model changes and vice versa [Hay09].

The Naked Objects Pattern invented by Pawson, is a system design approach, wherein core business objects are directly shown through the user interface (UI) [PM01]. The UI is automatically generated from the definition of the domain objects, and all the interactions possible within system are basically invoking methods on those objects. It is implemented using different frameworks such as Naked Objects Framework [3] and Apache Isis [4]. This makes the system very expressive from the user's point of view. Since the underlying model is reflected in the UI, stakeholders can intuitively recognize if it is incorrect. It can then be easily and quickly fixed by developers.

Using DDD along with the Naked Objects Framework can enable rapid prototyping, where a prototype is the literal manifestation of the underlying domain concepts. Such a domain model is understandable by all the stakeholders. It is better than static UML diagrams as it is directly maintained in the code.

Another such approach is Exploratory Modeling [Tön07] wherein a model is experimentally created in a suitable programming language instead of creating extensive UML diagrams. It relies on cycles of continuous feedback loops between experiments and current model documentation.

4 An example

Figure 3 depicts a workflow that could go from the self elicited goals by stakeholders to a working prototype, through an active collaboration between stakeholders and developers. We will explain this workflow with the help of a toy example.

Consider Alice who owns a small book library with 2000 books, and which has a total of 150 current readers. Alice has identified certain problems such as the readers in her city have no or little time to physically visit the library, and hence she is thinking of a different business model where books are delivered to a reader's doorsteps when requested. To experiment with this business model, the library's book list has been distributed to each reader, and they call or SMS the library to request a book or return one.

This business model still has some shortcomings, as the readers end up calling the library just to realize that the book they want is already borrowed by someone else.

[3]NakedObjectsFramework
[4]http://isis.apache.org/

Apart from this, the library is still managing all the transactions manually by recording reader activities *i.e.,* borrowing a book and returning a book in a big physical catalog. It is relatively hard to search for specific books or to keep a high level overview of the entire operation.

To tackle these challenges, the Alice is planning to build a mobile application for its readers and librarians. The readers would simply be able to browse through the list of books or search for a specific book. They would be able to see the current status of the book *i.e.,* available or borrowed, and should be able to request the book through the app if they wish. The Librarian, on the other hand, would see who has borrowed which book and when it is due for a return, as well as any new book requests.

Building such an application would require some straightforward upfront effort. There is clearly domain knowledge to uncover, and it involves different stakeholders *i.e.,* the Librarian and Readers with varying interests. It involves workflows and several tasks to achieve a certain goal.

To approach this problem, we can start with Vision Backlog (see Figure 3) where librarians and readers who are our non-technical stakeholders, would enter the information about the tasks they perform, their motivation behind it, and terms or concepts involved *etc.* A sample task entered by a reader could be: *As a Reader, I want to search Books according to Authors.* A Reader's primary goal is to be able to efficiently search for books, and a subgoal is to be able to search according to specific criterion. This little information entered in Vision Backlog tells the requirements analysts and developers the possible domain entities involved *e.g., Reader*, *Book*, *Author* along with possible relationships among them *e.g., Book* has an *Author*.

It is possible to uncover most of the domain entities and their relationships through Vision Backlog with much less overhead of actual meetings. It additionally helps the analysts and developers with two things. First, it helps them to identify the user goals and the problems they are facing. Second, it helps them to build *Personas* from the personal attributes of the stakeholders, which enables them to think of more accurate solutions for problems of a specific Persona.

Having uncovered important domain entities and their relationships, developers can take advantage of the frameworks such as Naked Objects Framework to rapidly build prototype as a generic UI. The uncovered domain entities become classes, and the UI is automatically generated by the underlying framework, which enables the stakeholders to interact directly with those entities. This allows a rapid feedback, because if the underlying domain model— which is nothing but code, is inaccurate—it is reflected on the UI so that the developers can react much more quickly to fix it.

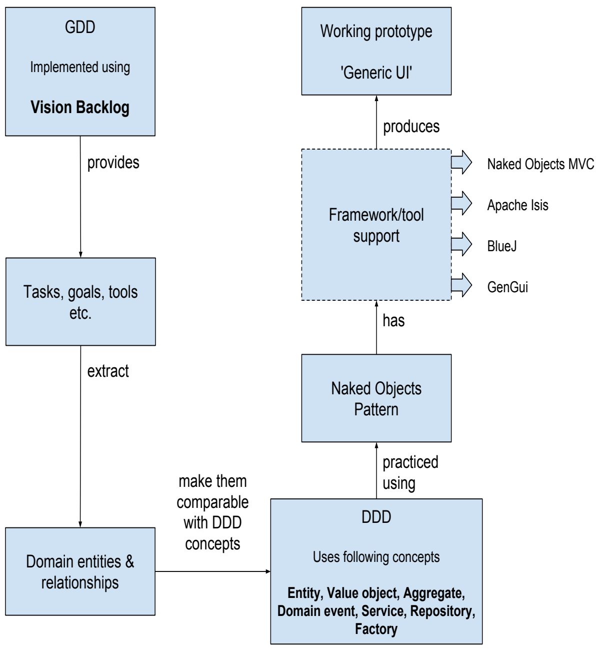


Figure 3: Combining GDD with DDD

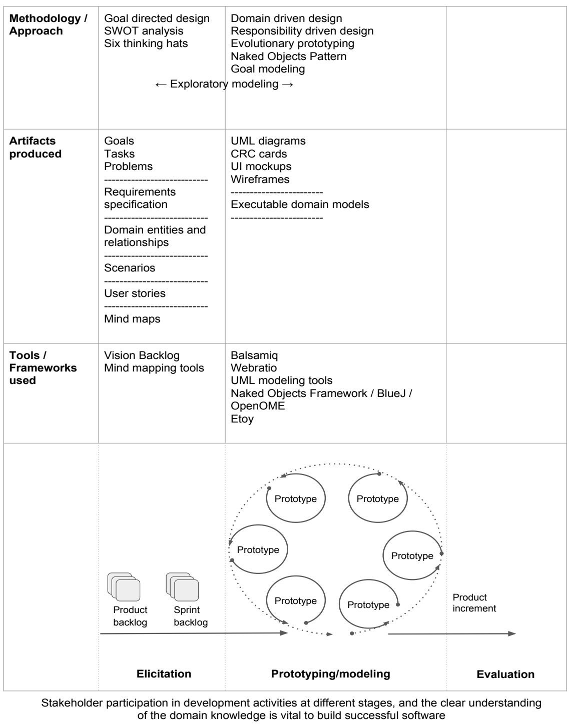
5 Research plan

The research question that I would like to address in my PhD is, "*How can domain models be specified and deployed as executable software artifacts suitable for testing, expressing requirements and driving design and implementation?*"

We have argued that stakeholder participation and the domain knowledge play a vital role in building successful software. Our focus of interest is how to enable the active contribution of non-technical stakeholders in order to uncover domain knowledge through elicitation, prototyping, and evaluation. To facilitate this we would like to figure out how we can design innovative development loops to support these activities. We plan to investigate which methodologies and approaches could be of interest in this regard, which artifacts can be used so that stakeholders can naturally relate to them, and which tools and frameworks could be used.

In Figure 4 we propose such a development cycle. We imagine a Sprint [Sch97] and divide it into three phases *i.e.,* Elicitation, Modeling/Prototyping, and Evaluation in which we should actively involve stakeholders. For each of these phases we have listed the approaches and methodologies that encourage stakeholder participation which could be of interest, the artifacts that are produced in these phases, and the tools and frameworks which could be used. Keeping this diagram in perspective, we find the following questions worth looking at:

- *What could be a visual alternative to UML, more suit-*



Methodology / Approach	Goal directed design SWOT analysis Six thinking hats	Domain driven design Responsibility driven design Evolutionary prototyping Naked Objects Pattern Goal modeling	
	← Exploratory	modeling →	
Artifacts produced	Goals Tasks Problems --------------------------- Requirements specification --------------------------- Domain entities and relationships --------------------------- Scenarios --------------------------- User stories --------------------------- Mind maps	UML diagrams CRC cards UI mockups Wireframes ------------------------ Executable domain models ------------------------	
Tools / Frameworks used	Vision Backlog Mind mapping tools	Balsamiq Webratio UML modeling tools Naked Objects Framework / BlueJ / OpenOME Etoy	

Stakeholder participation in development activities at different stages, and the clear understanding of the domain knowledge is vital to build successful software

Figure 4: Bird's eye view on the research plan

able for non-technical stakeholders?

- *How could we represent stakeholder goals and scenarios in our code?*
- *How could we extract the domain entities and relationships from stakeholder goals and tasks specified in a natural language?*
- *How could we relate stakeholder goals and tasks to the user stories and further to the executable domain model?*
- *Could we combine methodologies like GDD, DDD, Exploratory modeling and RDD to build viable softwares?*

6 Conclusion

In this paper, we address the challenge of the lack of stakeholder participation in the project activities at different phases, and how the domain knowledge can effectively be modeled. We argue that stakeholder participation is crucial in order to uncover the domain knowledge and mention various tools and techniques which help to handle this challenge.

We mention a tool called Vision Backlog that assists non-technical stakeholders to uncover domain knowledge themselves. We also argue in the favor of executable domain models, which are literal representations of the underlying domain concepts in the code, and are more maintainable and understandable for all the stakeholders.

We also propose a development cycle and list methodologies, artifacts, and tools which could help to involve stakeholders actively in the development activities. We also present several questions which we will be looking at in the future.

Acknowledgments

The authors gratefully acknowledge the funding of the Swiss National Science Foundations for the project "Agile Software Analysis" (SNF project No. 200020_162352, Jan 1, 2016 - Dec. 30, 2018, `http://p3.snf.ch/Project-162352`).

References

[Bea01] K. Brown and G. Craig et al. *Enterprise Java Programming with IBM Websphere*. Addison Wesley, 2001.

[Bjo06] D. Bjorner. *Software Engineering 3: Domains, Requirements, and Software Design (Texts in Theoretical Computer Science. An EATCS Series)*, volume 10. Springer, 2006.

[CRC07] Alan Cooper, Robert Reimann, and David Cronin. *About face 3: the essentials of interaction design*. John Wiley & Sons, 2007.

[DB17] Edward De Bono. *Six thinking hats*. Penguin UK, 2017.

[Ent10] Enterprise Architect - UML design tools and UML CASE tools for software development, 2010.

[Eva03] Eric Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Hay09] Dan Haywood. *Domain-driven design using naked objects*. Pragmatic Bookshelf, 2009.

[Len15] Yin Leng. AN INVESTIGATION OF THE ROLE OF Full paper 1 Literature Review. *ECIS 2015 Proceedings*, pages 1–14, 2015.

[MT13] Walid Maalej and Anil Kumar Thurimella. *Managing requirements knowledge*. Springer, 2013.

[Pat18] Nitish Patkar. Vision Backlog. Master's thesis, University of Paderborn, Germany, 2018.

[PM01] Richard Pawson and Robert Matthews. Naked Objects: a technique for designing more expressive systems. *SIGPLAN Notices*, 36(12):61–67, 2001.

[Sch97] Ken Schwaber. SCRUM Development Process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[Tön07] Andreas Tönne. Exploratory modeling with SAP NetWeaver (white paper), 2007.

[Ver13] Vaughn Vernon. *Implementing Domain-Driven Design*. Addison-Wesley, 2013.

[WBW89] Rebecca Wirfs-Brock and Brian Wilkerson. Object-oriented design: A responsibility-driven approach. In *Proceedings OOPSLA '89*, pages 71–76, October 1989. ACM SIGPLAN Notices, volume 24, number 10.

[Yoz14] Kadir Yozgyur. A proposal for a requirements elicitation tool to increase stakeholder involvement. In *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*, pages 145–148. IEEE, 2014.

[ZC05] Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.

[ZZ07] Hongyu Zhang and Xiuzhen Zhang. Comments on "data mining static code attributes to learn defect predictors". *IEEE Trans. Softw. Eng.*, 33(9):635–637, September 2007.