ReqCap: A Domain Specific Language for Capturing Requirements

Raunak Srikant Mokhasi
IIIT-Delhi
New Delhi, India
raunak17085@iiitd.ac.in

Rahul Mohanani
IIIT-Delhi
New Delhi, India
rahul.mohanani@iiitd.ac.in

Raghava Mutharaju
IIIT-Delhi
New Delhi, India
raghava.mutharaju@iiitd.ac.in

ABSTRACT

Generally, software requirements for a system are captured using informal high-level natural language descriptions. These requirements are shared with various teams that involve developers, designers, architects, quality control engineers and security experts. It will be helpful for these personnel if the requirements are more granular and in a language that they understand. We propose *ReqCap*, a Domain Specific Language (DSL) for capturing requirements. It helps stakeholders to formally capture the requirements in a consistent, systematic manner as opposed to the conventional methods of taking surveys or interviews and then writing the user requirements in a potentially ambiguous language like English. It also saves time and prevents conflicts in understanding the user-specified requirements. Here, we discuss the feedback on capturing requirements from various stakeholders involved in software development. This discussion forms the basis for our DSL. We then describe the syntax of ReqCap, along with security and architecture requirements that are captured using our DSL. ReqCap is under active development, and we are expanding its vocabulary to enable the stakeholders to capture the requirements effectively in a language that they are familiar with.

KEYWORDS

Domain Specific Language, Requirements Capture, Requirements Elicitation

ACM Reference format:
Raunak Srikant Mokhasi, Rahul Mohanani, and Raghava Mutharaju. 2019. ReqCap: A Domain Specific Language for Capturing Requirements. In *Proceedings of Innovations in Software Engineering Conference, Jabalpur, India, February 2020 (ISEC),* 5 pages.
DOI:

1 INTRODUCTION

Clear requirements mark the beginning of the software development process [4]. These requirements are in general informal, natural language descriptions of one or more features needed in a software system, and are often written from the perspective of an end-user of the system. They are an integral part of any software development process. Different stakeholders and categories of personnel, such as designers, architects, developers, quality control engineers, security engineers, user interface designers, etc., make use of requirements in different ways. Customers specify their requirements and validate the end product using the requirements given initially. Managers use the requirements to plan the complete software system, including the resources required. Software engineers use the requirements to understand the software system that needs to be developed, and the testing team uses the requirements to develop validation tests for the software.

Capturing the requirements for products involves information gathering, information transformation and requirements generation. Typically, these requirements for products are captured through surveys, interviews and usability tests. These methods of requirements elicitation are not only time-consuming and informal but sometimes even lead to conflicts. The requirements represent the customer's point of view, and thus are at a high-level for developers who would benefit from a more technical description of requirements that are suitable for implementation. Without this level of granularity, developers might miss some requirements or may interpret the requirements incorrectly. This, along with poor description of requirements, can lead to a huge loss for the organization [3]. The majority of projects that fail are due to poor description and interpretation of requirements [9].

We have addressed these limitations by capturing requirements using a DSL called *ReqCap*. Domain-Specific Languages [11] are focused on a particular domain and use a limited vocabulary and grammar to convey meaning. Our *domain* [2] involves capturing the requirements of users involved in the software development life cycle. The ReqCap grammar enforces discipline among product managers, as they have to capture user inputs using the structure of the DSL. This ensures consistency across the organization and reduces the ambiguity. ReqCap also helps in capturing the requirements more formally.

The idea for this DSL came about through a survey taken from different personnel in various roles in organizations that include a Fortune 500 FinTech company, a seed stage EdTech startup and an investment bank. These surveys highlighted that, first, there wasn't a single way to capture requirements in their respective organizations. Second, users tend to be very vague in their requirements of the software system. Finally, the ideas for products are often narrowed down or lost whenever we reach the final stages of development. Due to these limitations, architecture and security teams were unable to add their recommendations within the typical user story format. Our DSL helps in such cases.

ISEC, Jabalpur, India

DOI:

2 LITERATURE REVIEW

Requirements are specifications from stakeholders that they want accomplished during the process of development of software. In the software development process, requirements are fundamentally important because they provide a vision of the goals that are to be achieved. These requirements need to be specified accurately, unambiguously and managed because any lapse of a vital requirement can lead to increased product costs and sometimes even product failures. *Requirements Engineering* [8, 14] is the process of gathering, outlining, detailing and preserving requirements. It consists of the following steps:

- Requirements Elicitation. It is the process of capturing requirements and it allows stakeholders to provide their needs regarding the software product.
- Requirements Specification. It is the process of documenting requirements formally. They mostly contain models such as Use Case Diagrams, Entity Relationship diagrams and Data Flow Diagrams.
- Requirements Verification and Validation. In this step, it is ensured that the software has accurately implemented the documented requirements and that it is done precisely as per the needs of the customer.
- Requirements Management. In this step, the changing nature of requirements are monitored and continuous communication with the stakeholders is ensured.

Test Driven Development (TDD) [1] is the process of writing automated test cases for software and then producing and refactoring the code to pass the test. Agile methodology [12] is an iterative process of project management in software development. In the Agile methodology, work sequences called sprints that are incremental and iterative are involved. It involves the delivery of updates frequently by every team in a company and also promotes the alliance of business and development teams. Behaviour Driven Development (BDD) [15] is an agile software development process that generates test cases for software through conversion of natural language statements. BDD is assisted through domain specific languages like Gherkin that employ structured (and restricted) natural language statements to convey meaning. Compared to TDD, BDD involves writing the specification in simple restricted English statements, which in turn are automatically converted to test case stubs. ReqCap follows BDD and a Gherkin based approach where we provide the vocabulary for specifying the requirements. Test case stubs are generated based on the requirements specified using our DSL.

3 METHODOLOGY

We conducted a survey that included 10 participants of distinct demographics. The participants were between the age group of 20 years to 40 years, of all genders and were working in different roles such as product managers, software engineers, architects, and security experts at startups and well-established multinational corporations. Our survey questions are as follows.

Q1) Have you ever used user stories to get new ideas or updates for your software/products?
Q2) Do you know the process involved in requirements capturing?
Q3) Does your role involve capturing user requirements for software systems?
Q4) What are some examples of the requirements that your team is working on?
Q5) What are some common issues faced by you while capturing requirements or while conducting user studies?
Q6) What are your suggestions to improve the requirements capturing process?
Q7) Have you ever tried using a restricted English language that is understood by business, developers, and testers based on your user stories and having the standard who-what-why questions?
Q8) Do you have an idea about Behaviour Driven Development? What is your opinion about it?
Q9) What are the tasks that developers, testers, UI designers, and security engineers have to do that are not captured in requirements?
Q10) What are some non-functional requirements that might be missed by product managers?
Q11) Do you think it is better if there is a formal way to capture requirements through a Domain Specific Language?

The summary of the response for each question is as follows.

For Q1 Almost all the participants said that user stories are used to capture the requirements. The product managers across different teams write the user stories in different levels of detail. Some of them prefer putting everything in text, while the others prefer keeping it minimal and explaining the user stories verbally. The development team therefore always has a learning curve associated with adjusting to the style of a new product manager which can lead to confusion, bugs and delays.

For Q2 Most of the participants knew about the process involved in capturing requirements. They also said that requirements are captured using several mechanisms such as surveys, interviews, and prototype feedback.

For Q3 Most of the respondents said that their role involved capturing requirements. The respondents who said "no" mentioned that they got their requirements not directly from the end-user but from in-market teams who interacted with them like a business product manager, customer support or technical account manager. In this case, some of the details in the requirements often get lost in translation.

For Q4 The requirements from the survey could be categorized into user requirements, system requirements, and software requirements. The software requirements are the services that the system should provide and the expected outcome when a particular input is given to the software. System requirements mainly describe the functionality of what is required from the system services. The user requirement was that the software be usable by everyone from different domains, even those who did not have any technical knowledge. Apart from these, it also included constraints or limitations of the application being developed.

For Q5 The answers to this question had lot of variations. There were several issues with the current methods used for capturing requirements such as different formats, different

jargon, and use of same words to mean different things in different contexts.

For Q6 The responses varied from training to stronger guidelines while capturing requirements. The major theme was "*uniformity*" although no one explicitly mentioned using a DSL.

For Q7, Q8 Few of the participants had used a restricted English language earlier and said that it contributed to uniformity. So even though they do not recall a DSL, they thought it made sense when we brought it up. Some even mentioned the limitations of a DSL and Behaviour Driven Development.

For Q9, Q10 The tasks that are not captured in the requirements and/or are non-functional requirements that might be missed by product managers are as follows. Provisioning the development machine, setting up the local database, building the code and pushing it to the master repository so that it can be deployed, running deployment verification tests to ensure correct production copy, getting the new code's binaries deployed onto the quality assurance environment and running build verification tests to ensure that the code is deployed correctly. Along with that security requirements like preventing XSS (cross-site scripting), preventing SQL injections and having proper authentication for APIs, are also generally not explicitly mentioned in the requirements.

For Q11 Most of them indicated that having a formal language to capture requirements will be very helpful. Some of the participants had reservation as they were not sure what this new language would look like. They knew that the currently existing DSLs had many limitations.

We will illustrate the issues faced by the survey respondents with the help of the following example. Imagine your portal has a searchable table with 10 columns of customer details, and you can search for a customer using a string/regex for any of those 10 columns. Later, if you want to add a new searchable column to that table, the user story is just as follows - "As a user of the portal, I want to search for customers using a new column for 'pincode', so that I can pull a list of all customers who live in the same pincode".

Conventional methods of capturing requirements will let you capture these high-level requirements. However, for developers, this entails three tasks:

(1) For the back-end function, create an API to read 'pincode' data from the database. After this, we should add pincode as an option that can be searched on.
(2) For the UI function, add a table column to read the pincode value from API and display it. Also, ensure that the pincode shows up as an option to be searched on.
(3) For the search function, which is often facilitated through separate data stores like Cassandra, and not through the RDBMS using SQL, we have to ensure 'pincode' is now always also written to the Cassandra "search database", and not just to the regular SQL "customer info database".

These details are often lost or not captured when effort estimates are being made and tracked.

4 FINDINGS

ReqCap uses a Behaviour-Driven Development [13] approach where it reads the requirements from a text File (called Objective file) and then generates *Test Stubs* [7] in Java from it. Here, ReqCap is a Ubiquitous language [5] which is a language understood by the developers, business team, testers and the users. It attempts to answer the standard who-what-why questions in a uniform way. The default vocabulary of ReqCap is:

- **Objective**: Text File where the requirements are written
- **Goal**: Outcome of the scenario taking place
- **Knowing**: Preconditions or explanation of the preliminary background
- **If**: If the occurrence of a particular event takes place
- **Do**: The Expected output that should take place when the event occurs

We will use the following example to illustrate ReqCap. The product requirement is to have a login page (Figure 1), post which there would be several categories to choose from. Upon choosing a category, well known quotes from that category will show up on the screen.

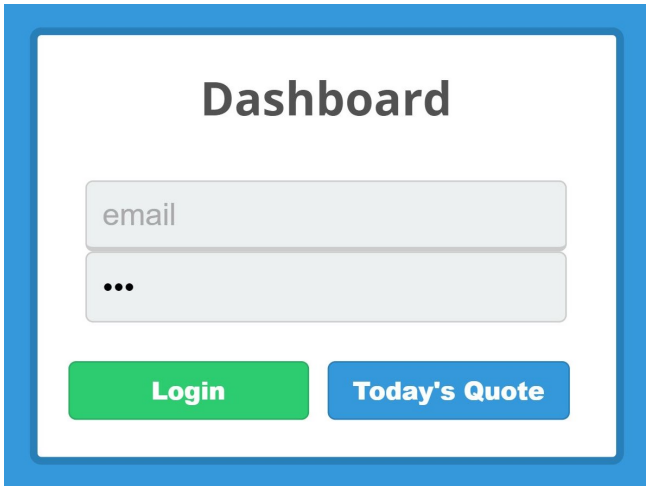

Figure 1: Login Page

When the user clicks on the quote button, we fetch the quote and display it (Figure 2). At this point, our Objective file would look like Figure 3.

So far, only the product manager has captured user requirements using the keywords made available by default by our DSL. In addition to this, there will be recommendations from the security team and architecture team. The security recommendation would be to prevent XSS and SQL injection. The architecture recommendation would be to not call this REST API every time the user clicks on the button on our website, but rather be as follows:

(1) Pre-fetch all the quotes available on the REST [10] API daily using a job.
(2) Store all these results in an internal database.
(3) Whenever the user clicks on the get quote button on our website, data should be fetched from the internal database

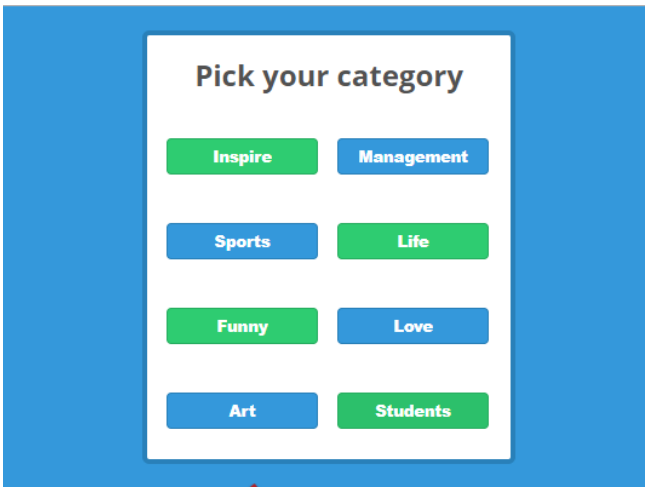

Figure 2: Quotes Page

```
Objective: Login functionality on login page
  Goal: Verification of Correct Login
    Knowing Open the Chrome Browser and launch the Login page
    If Enter Raunak as Username and Mokhasi as Password
    && Click the login button
    Do Log the user in and load Quotes page
  Goal: Verification of Invalid Login
    Knowing Open the Chrome Browser and launch the Login page
    If Click the login button
    Do Show invalid login alert
  Goal: Verification of Get Quote button
    Knowing Open the Chrome Browser and launch the Login page
    If Click Get Quote button
    Do Show quote in alert
```

Figure 3: Objective File

and not any external REST API. This would make our website performance much faster.

(4) In addition, it should follow architecture guidelines from a URL

ReqCap can support these requirements. The internal engineering teams like security and architecture have the option to add their own keywords to our DSL. They get to specify what these keywords are and what code is supposed to be rendered when these keywords are encountered in the Objective file. Once they have added the keywords and provided their specification (i.e. what Java code to render when these words are seen), the security and architecture teams will modify the Objective file itself to include the new keywords that they have just added. Whatever follows these keywords will be parsed by our DSL and rendered according to the specification provided by these teams for that keyword.

ReqCap will also allow *reference* to other existing functions so there is no code duplication. For example, if login is a prerequisite for some other functionality it will directly refer to the login functions. It will also allow project management to link the Objective file to the APIs provided by project management software (like Rally or Jira) so that the data there is in sync with your Objective files.

The Objective file with the above extensions is shown in Figure 4.

```
Objective: Login functionality on login page
  Goal: Verification of Correct Login
    Knowing Open the Chrome Browser and launch the Login page
    If Enter Raunak as Username and Mokhasi as Password
    && Click the login button
    Do Log the user in and load Quotes page
  Goal: Verification of Invalid Login
    Knowing Open the Chrome Browser and launch the Login page
    If Click the login button
    Do Show invalid login alert
  Goal: Verification of Get Quote button
    Knowing Open the Chrome Browser and launch the Login page
    If Click Get Quote button
    Do Show quote in alert
  *security_xss prevent cross-site scripting
  *security_injection ensure no sql injection is possible in any text input
  *security_general
  *project_mgmt deliver across login and quotes team in 2 sprints
  *architecture set up job to retrieve quotes at start of day
  *architecture query internal db instead of calling api at runtime
  *architecture_general
```

Figure 4: Objective File with Recommendations

Based on this Objective File, now the test stubs are generated for the Login Page. The generated Java code is shown below.

```
public class TestStubs{
  // Objective: Login functionality on login page

  public void Chrome_Browser_Opened_and_Login_Page_launched(){}

  public void Enter_Raunak_as_Username_and_Mokhasi_as_Password(){}

  public void Click_the_login_button(){}

  public void Click_Get_Quote_button(){}

  public void Ensure_No_Sql_Injection(){}
  // security: prevent XSS and SQL injection
  /* check general security guidelines at
     www.demoquotes.com/wiki/reqCap_security */

  public void create_quotes_retrieval_job(){}
  /* project_mgmt:
     deliver across login and quotes team in 2 sprints */
  /* architecture: query the internal database
     instead of calling API at runtime */
  /* review general architecture guidelines at
     www.demoquotes.com/wiki/reqCap_architecture */
}
```

5 DISCUSSION AND CONCLUSION

The basic flow in ReqCap is as follows

(1) The product manager writes requirements using primary keywords and grammar that come out of the box with ReqCap.
(2) Developers and architects write lower-level tasks based on requirements using new keywords and unit test stubs are generated from this DSL.
(3) Teams would provide keywords to specify the tasks for different teams. This will annotate the functional and unit tests with team name so that the correct team works, or reviews and signs off on their components. Thus, progress can be tracked team-wise and at a more granular level since there are tasks instead of user stories.

Thus, ReqCap captures not only high-level user requirements but also captures the low-level requirements and team information. The DSL is currently being developed using Xtext [6] which is a framework for the development of domain-specific languages. We are also planning to add more features, that include the following:

(1) We are in the process of implementing *partial code generation* on reading the Objective File (using Natural Language Processing techniques), and not just basic test stubs generation.
(2) We will seek community feedback on ReqCap's vocabulary and look into mechanisms to encourage commnity participation in shaping the vocabulary of ReqCap. For example, if we do not include some common keyword that is used by almost every company in some niche industry, they might want to contribute it back into the core grammar. So, we would do research around the best way to solicit this information and achieve community consensus around it.
(3) We are also working on internationalization, where we would be providing translations for ReqCap keywords in international languages so that it can be used in a uniform way by non-English speakers and enhancing the parsing logic to work with any of these non-English languages.
(4) We also plan to extend test-stub generation to languages other than Java, since there is a huge development ecosystem today and developers use a plethora of programming languages. Hence, the test stubs that ReqCap generates should be available in as many of these languages as possible.

Some organizations use Gherkin to capture requirements, which is the domain-specific language used for Cucumber specifications. It also generates test stubs reading a text file using the concept of Behaviour Driven Development. However, Gherkin has many limitations that ReqCap solves such as

- **Directives**: In ReqCap, teams can modify the Objective file and add their recommendations to the document. Instead of generating functions from their instructions (like it happens for Gherkin), ReqCap will provide those as code snippets and comments.
- **Extensibility**: Each project can define its own set of keywords. The action for these keywords will be a sentence that comes up in the comments or sometimes with a link to an internal guidelines wiki.
- **Reference**: ReqCap will allow a reference to other existing functions, to avoid code duplication. For example, in Gherkin, if you are testing an internal page, you will have to rewrite all the login steps (this can lead to inconsistency). In ReqCap, we will be able to merely reference another test so that we can maintain consistency and reduce duplication.
- **Mutually Exclusive Conditions**: Gherkin does not make it easy to specify mutually exclusive conditions. ReqCap allows it by providing an else-if condition in the vocabulary to test events that cannot occur at the same time.
- Generally, a project management software like Rally or Jira is used in many organizations. Executives and project managers do not look at the user stories that developers refer to in code. So if the project management software is out of sync with the requirements due to an update or increase in scope, this can lead to a problem in communication and Gherkin has no way to check this. ReqCap will solve this problem by comparing the Objective files against the data offered by the Rally/Jira APIs to ensure that the project management software reflects the latest features required.

Gherkin has an ecosystem and has already been adopted by many players. It has been localised in many different languages, and it works seamlessly with BDD systems like Cucumber. So even if it has flaws, there will be a cost for moving away from it that companies might not be willing to pay.

In this work, we showed how ReqCap ensures the formal capturing of requirements and is consistent when compared to conventional methods of capturing requirements. The proposed approach saves time and prevents conflicts in understanding what the user is trying to convey.

ACKNOWLEDGMENTS

We would like to thank the participants of the survey for their contribution to the research paper, as it provided useful inputs such as low-level requirements or granular implementation details not being captured and also the cost of poor or incomplete requirements in their companies.

REFERENCES

[1] Kent Beck. 2003. *Test-driven development: by example.* Addison-Wesley Professional.
[2] Dines Bjørner, Chris W George, and Søren Prehn. 1995. Domain Analysis-a Prerequisite for Requirements Capture. (1995).
[3] Barry W Boehm and Philip N. Papaccio. 1988. Understanding and controlling software costs. *IEEE transactions on software engineering* 14, 10 (1988), 1462–1477.
[4] Rachel Cooper, Andrew B Wootton, and Margaret Bruce. 1998. fiRequirements capturefi: theory and practice. *Technovation* 18, 8-9 (1998), 497–585.
[5] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley Professional.
[6] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion.* ACM, 307–309.
[7] Martin Fowler. 2007. *Mocks Aren't Stubs.* https://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs
[8] Gerald Kotonya and Ian Sommerville. 1998. *Requirements engineering: processes and techniques.* Wiley Publishing.
[9] Dean Leffingwell and Don Widrig. 2000. *Managing software requirements: a unified approach.* Addison-Wesley Professional.
[10] Mark Masse. 2011. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* " O'Reilly Media, Inc.".
[11] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.
[12] Ken Schwaber and Mike Beedle. 2002. *Agile software development with Scrum.* Vol. 1. Prentice Hall Upper Saddle River.
[13] Carlos Solis and Xiaofeng Wang. 2011. A study of the characteristics of behaviour driven development. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications.* IEEE, 383–387.
[14] Ian Sommerville and Pete Sawyer. 1997. *Requirements engineering: a good practice guide.* John Wiley & Sons, Inc.
[15] Matt Wynne, Aslak Hellesoy, and Steve Tooke. 2017. *The cucumber book: behaviour-driven development for testers and developers.* Pragmatic Bookshelf.