Capstone Project

Software Architecture

Semester 1, 2022

Richard Thomas & Brae Webb

Summary

Throughout the software architecture course, you have learnt about a subset of quality attributes of concern to software architects. You have also been exposed to a number of techniques to satisfy these attributes. Now, as the capstone project, you are required to

- propose a non-trivial software project,
- identify the primary quality attributes which would enable success of the project,
- design an architecture suitable for the aims of the project,
- deploy the architecture, utlising any techniques you have learnt in or out of the course, and
- evaluate and report on the success of the software project.

The successful completion of the project will result in three deliverables, namely,

i a proposal of a software project, the proposal must clearly indicate and prioritise two or three quality attributes most fundamental to the project's success,

ii the developed software, as both source code, and a deployed artifact, and

iii a report which evaluates the success of the developed software relative to the chosen quality attributes.

Your software deliverable must include all supporting software (e.g. test suites or utilities) that are developed to support the delivered software.

1 Introduction

We have looked at several core quality attributes in this course, and will continue to look at more over the remainder of the semester. These attributes were selected because they are key concerns of many real-world software projects. In this project, you will have an opportunity to explore some of the fun of industry. You will take the role of an entrepreneur, software architecture, developer, and operations team.

As entrepreneur, you have proposed a project idea. These proposals have been evaluated by your peers and the teaching team. You have been allocated to a project based on interest you have indicated by voting on proposals and on your previous coursework experience.

As a team you now need to perform the roles of software architect, developer, and operations team. You should design the basic structure of the initial software architecture, based on the scope, functionality, quality attributes, and evaluation plan from the proposal. The details of the architecture are expected to evolve as you start implementing parts of the system.

Part of the assessment will be how the architecture evolves in response to what you learn during development. You need to write Architectural Decision Records (ADRs) for each decision you make about the design of the architecture [1]. These are to be recorded in your GitHub repository so that the marker can see how your architecture evolved and the reasons for the decisions you made.

2 Software

You need to implement a software system that delivers a Minimal Viable Product (MVP)[1]. The MVP needs to implement a usable core of the system's functionality, which demonstrates that the architecture could deliver the full system functionality. The MVP also needs to allow the software architecture to be tested to determine if it can deliver the project's important quality attributes.

You may renegotiate the scope of the system during the project, if you determine that certain aspects of the original scope are not feasible within the project time constraints. The earlier you do this, the less it will impact on your final result. You will not explicitly lose marks for renegotiating scope, unless the revised scope limits your ability to adequately test important quality attributes. But, late changes to scope are likely to have a flow-on effect that could reduce the quality of your final deliverables. This means that you should attempt to implement some of the riskier parts of the project early.

3 Evaluation

You need to test the software system that you implement to demonstrate how well its architecture supports delivering system functionality and its quality attributes. This evaluation should be based on the proposal's evaluation plan, but should *not* be limited to only what is in that plan. You will be assessed on how well you test your system in terms of both functionality and quality attributes. Discovering issues with the system or its architecture during testing will not adversely affect your marks for the evaluation component of the assessment.

A section of your project report needs to summarise the test results and provide access to the full suite of tests. You should automate as much of the testing as possible. Any manual tests need to be documented so that they can be duplicated. The results of all manual tests need to be recorded in a test report. This may be a section of the project report, or it may be a separate document with a link to it from the project report. You need to include test code and test infrastructure in your project's repository.

4 Report

The report should include the following content.

Title Name of your software project.

Abstract Summarise the key points of your document.

Changes Describe and justify any changes made to the project from what was outlined in the proposal.

Architecture Describe the MVP's software architecture in detail.

Trade-Offs Describe and justify the trade-offs made in designing the architecture.

Critique Describe how well the architecture supports delivering the complete system.

Evaluation Summarise testing results and justify how well the software achieves its quality attributes.

Reflection Lessons learnt and what you would do differently.

You do not need to have sections for each topic above, though your report needs to contain the content summarised above. For example, the description of the architecture could include discussion of trade-offs. Similarly, the critique and evaluation could be combined so that both are discussed in relation to an ASR.

[1] https://www.agilealliance.org/glossary/mvp/

When writing your report, you may assume that the reader is familiar with the project proposal. You will need to describe any changes your team has made to the original proposal. A rationale should be provided for each change. Small changes only need a brief summary of the reason for the change. Significant changes to functionality of the MVP, or changes to important quality attributes, need a more detailed justification for the change. You should provide a reference and link to the original proposal.

The full architecture of your MVP needs to be described in enough detail through views, diagrams, and commentary to give the reader a complete understanding of the architecture's design. You should describe parts of the detailed design that demonstrate how the architecture supports delivering key quality attributes. (e.g. If interoperability was a key quality attribute, you would need to describe the parts of the detailed design that support this. For example, how you use the adapter design pattern to communicate with external services.)

Any trade-offs made during the design of the architecture need to be described. Explain what were the competing issues[2] and explain why you made the decisions that resulted in your submitted design.

When describing the architecture and trade-offs, you should summarise and/or reference ADRs that relate to important decisions that affected your architecture.

Your critique should discuss how well the architecture is suited to delivering the full system functionality and quality attributes. You should use test results to support your claims, where this can be shown through testing. For quality attributes that cannot be easily tested (e.g. extensibility, interoperability, ...), you will need to provide an argument, based on your architectural design, about how the design supports or enables the attribute. Some quality attributes (e.g. scalability) may require both test results and argumentation to demonstrate how well the attributed is delivered.

Test plans, and test results, should be summarised in the report. Links need to be provided to any test plans, scripts or code in your repository. Where feasible, tests should be automated with information about how to run the tests. Ideally, you should use GitHub Actions[3] to run tests and potentially deploy artefacts.

Your report should end with a reflection that summarises what you have learnt from designing and implementing this project. It should include descriptions of what you would do differently, after the experience of implementing the project. Describe potential benefits or improvements that may be delivered by applying the lessons you have learnt during the project.

5 Repository

Your team will be provisioned with a repository on GitHub. All development work and documentation are to be committed to the repository. All project artefacts are to be submitted via this repository.

- Model artefacts (e.g. PUML or Structurizr DSL files) should be stored in the `/model` directory.
- ADRs are to be stored in the `/model/adrs` directory.
- The report must be stored in the `/report` directory.

Do not commit large binary files to the repository. (i.e. Do *not* commit Word documents or frequently changing PDF files to the repository.) It is recommended that you use markdown or LaTeX to write your report. If you use LaTeX, you should use GitHub actions to produce a PDF of the report.

Your final submission will be what is in your repository at the due date, 16:00 (AEST) June 10.

[2] "Forces" in design patterns terminology.

[3] https://docs.github.com/en/actions

6 Academic Integrity

As this is a higher-level course, you are expected to be familiar with the importance of academic integrity in general, and the details of UQ's rules. If you need a reminder, review the Academic Integrity Modules[4]. Submissions will be checked to ensure that the work submitted is not plagiarised.

All code that you submit must be your own work, or must be appropriately cited. If you find ideas from online sources (e.g. Stack Overflow), you must cite and reference[5] these sources. Use the IEEE referencing style[6] for citations and references. Citations should be included in a comment at the location where the idea is used in your code. All references for citations must be included in a file called `refs.txt`. This file should be in the root directory of your repository.

You may use libraries to help implement your project. The library's license must allow you to use it in the context of your project. All libraries used in your project must be listed in a file called `libs.txt`. This file should be in the root directory of your repository.

Uncited or unreferenced material will be treated as not being your own work. Significant amounts of cited material from other sources will be considered to be of no academic merit.

7 Demonstration

Your team needs to demonstrate your project's functionality and how well it achieves its goals. This should include a demonstration of how quality attributes are achieved, or a brief summary of how the architecture facilitates delivering a quality attribute.

You may use this booking page[7] to schedule a time to demonstrate your project between June 10 and 17. The demonstration is to take a **maximum** of 20 minutes. Your system must be deployed and set up so you can start your demonstration **immediately**. You should not take demonstration time to do any deployment or initialisation.

When you book the demonstration time, you will be sent a confirmation email with a Zoom link for the demonstration. Only **one person** from your team should make a demonstration booking. Ensure you discuss with your other team members to find suitable times that are available for booking, and for which at least all of your key team members are available.

8 Marking Criteria

20% Extent to which project's scope was delivered.

20% Suitability of architecture to deliver system goals.

20% Quality and thoroughness of testing.

20% Clarity, accuracy and completeness of architecture's description.

20% Insightfulness of architecture's evaluation.

[4] https://web.library.uq.edu.au/library-services/it/learnuq-blackboard-help/academic-integrity-modules

[5] https://web.library.uq.edu.au/node/4221/2

[6] https://libraryguides.vu.edu.au/ieeereferencing/gettingstarted

[7] https://calendly.com/richard-thomas-uq/csse6400-demo

Criteria	Standard				
	Advanced (35)	**Proficient (28)**	**Developing (23)**	**Emerging (16)**	**No Evidence (0)**
System Scope 20%	MVP's originally proposed functional & non-functional requirements, or those agreed and documented early in the project, are fully delivered.	MVP's originally proposed functional & non-functional requirements, or those agreed and documented early in the project, are delivered with small variances.	MVP's functional & non-functional requirements were revised and documented later in the project, and are mostly delivered.	MVP's functional & non-functional requirements underwent significant revision late in the project, and are mostly delivered. Or, little revision was done, but some significant requirements are delivered.	MVP's functional & non-functional requirements underwent significant revision late in the project, and are partially delivered. Or, little revision was done, and significant requirements are not delivered.
Architecture Suitability 20%	Delivered architecture, supplemented by the design reflection, is very well suited to delivering all key functional & non-functional requirements.	Delivered architecture, supplemented by the design reflection, is mostly suited to delivering all key functional & non-functional requirements.	Delivered architecture, supplemented by the design reflection, is fairly well suited to delivering most key functional & non-functional requirements.	Delivered architecture, supplemented by the design reflection, should deliver some key functional & non-functional requirements.	Delivered architecture, supplemented by the design reflection, may deliver one or two key functional & non-functional requirements.
Testing Quality 20%	All key functional & non-functional requirements, & key architectural components are well tested (or are described adequately in a test plan) and, where feasible, are mostly automated.	Most key functional & non-functional requirements, & key architectural components are well tested (or are described adequately in a test plan) and, where feasible, are mostly automated.	Most key functional & non-functional requirements, & key architectural components are fairly well tested (or are described fairly adequately in a test plan) and, where feasible, many are automated.	Main test cases for a few key functional & non-functional requirements, & key architectural components are fairly well tested (or have some meaningful description in a test plan) and, where feasible, some are automated.	Testing is poor, superficial or extremely limited. There are very few, if any, automated tests.
Architecture Description 20%	Accurate, clear and complete description of all aspects of the architecture. Narrative text and diagrams complement each other. Views support describing all aspects of the architecture. Decisions about trade-offs are well described within the context of the design discussion.	Accurate, clear and mostly complete description of the architecture. Narrative text and diagrams complement each other. Views support description of the architecture. Decisions about important trade-offs are well described within the context of the design discussion.	Mostly accurate and fairly complete description of the architecture. Narrative text and diagrams support each other. Decisions about some important trade-offs are adequately described within the context of the design discussion.	Some parts of the description are inaccurate or incomplete. Most diagrams are relevant to the narrative text or a necessary diagram is missing. Few trade-offs are adequately described within the context of the design discussion.	Many parts of the description are inaccurate or incomplete. Few diagrams are relevant to the narrative text or many necessary diagrams are missing. Trade-offs are poorly described.
Architecture Evaluation 20%	Critique & evaluation clearly demonstrate that the delivered architecture, varied a little by the reflection comments, can deliver all functional & non-functional requirements of the full system.	Critique & evaluation fairly clearly demonstrate that the delivered architecture, varied by the reflection comments, can deliver all functional & non-functional requirements of the full system.	Critique & evaluation imply fairly well that the delivered architecture, varied by the reflection comments, might deliver most functional & non-functional requirements of the full system.	Critique & evaluation demonstrate that the delivered architecture, varied by the reflection comments, is unlikely to deliver some key functional or non-functional requirements of the full system.	Critique & evaluation demonstrate that the delivered architecture, varied by the reflection comments, is unlikely to deliver most functional or non-functional requirements of the full system. Or, they are too unclear to determine.

References

[1] R. Thomas, "Architectural decision records," March 2022. https://csse6400.uqcloud.net/handouts/adr.pdf.