# Software Architecture Documentation for<project name>, revision #

### Team <#><Authors><Date>

# 0. Guidelines to using this template

1. **Lifetime:** This starts out as an architecture document. At some point, you may extend it develop a design document from it, or you may instead leave it separate as the “high level” info and make what amounts to a new, “detailed design” document. Your choice![[1]](#footnote-2) In either case, the high level design needs to be kept up to date. The rest of this discussion refers primarily to what would be in the initial, high-level design document. Once written, the document should go under change control, like your other documents.
2. **Transition from problem statement:**  This document is intended to “answer” the problem statement you wrote. That is, it refers to the problem statement and describes how your high-level design will “solve” that problem. Indeed, if you go off and talk about some cool part of the design, it had better be traced back to “why it helps solve the problem” in a reasonable and cost-effective sounding way.
3. **Transition from requirements:** The architecture document also should sound like it elaborates at least a bit on the most critical parts of the requirements captured in use cases and in the supplementary spec. For example, if you have one major use case related to the key features, then it should be in here (Section 3). If you have “performance” as your top quality attribute requirement, and a key example of where that needs to be adhered to in the supplementary spec, then this architecture document should spell out how that performance will be achieved (Sections 4 & 5).
4. **Readability in general:** Most of this document should be readable by your client, all of it if your client is a technical person. The client should be convinced by this document that your design will solve his/her problem. It also should be readable by all the technical stakeholders. You should write this document with your readers in mind – what will they need to do with the document? Standard readability rules need to be followed – like avoid ambiguity and repetition, and explain what an acronym or jargon means the first time you use it, etc.
5. **Design information:** Ideally, this should explain to developers how the high-level design is supposed to work, in such a way that they could detail that from what they see here.
6. **Transition to testing:** Finally, it should guide (at a high level) testers as to what additional kinds of “white box” testing might be needed to do integration testing and system testing, beyond what they see in the requirements for “black box” testing of requirements.
7. **Overall length:** Target 10 pages, for the initial version of this document. The real intent is that it’s usable by your team, as the reliable reference for “what it’s supposed to look like” as a design.
8. **Appearance:** The heuristic for this is that an architecture document is “half pictures.” Each diagram or other figure should be fully annotated and explained. See section 6, below, for more info on that.
9. **Architectural style:** It should be clear from this document what the overall “style” of your system is, and you should show this in the images as well as talking about it. See [Garlan & Shaw’s famous article](http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf)[[2]](#footnote-3) for an introduction / refresher on this subject. (Of course, you may use more than one such style for different things the system does.)
10. **Interfaces:** An architecture document should provide high-level guidance to the key interfaces of the system, both to the outside world and also within the system itself. For example, if the system uses Corba to talk from its clients to servers, or PHP, or SOAP, there should be at least pointers from here to the other documents which detail those interfaces. Note that interfaces are a major part of the Element Catalog in Section 6, below. Section 7 describes the high-level Framework of your system. Theoretically, that should include a lot on the framework’s OO interfaces, and strategic ones of these should be discussed. Of course, there’s going to be real code to support this, as well.

# ----------------------------------------------------------------------------

# <Table of contents. You can use what’s built into Word.>

# 1. Introduction

<Describe the purpose of the document and give a brief overview. Include a guide for different types of stakeholders.>

# 2. Background

<Give a brief statement of the problem and the intended solution. Describe the main stakeholders and their concerns. Describe any existing systems, especially those that have interfaces with your system. Include a shortened version the very highest level points from the Problem Statement, and give a reference to that.>

# 3. Functional Requirements

<Describe the main functional requirements. You do not need to include all of the use cases, but a few examples might be helpful, as noted in the Guidelines, *above*. Refer the reader to the Requirements Document for more details. Pick at least one use case and detail it with design information, as a prime example of how the system will work doing its main function.>

# 4. Quality Attributes

<Use the scenario table format to describe all of the quality attributes that must be included in the final system. Rank these architectural drivers, in terms of their importance. As noted in the guidelines, above, the most important of these quality attributes, for the system to succeed, should be given more space.>

# 5. Patterns and Tactics

< Describe the tactics used to achieve each quality attribute. Describe any patterns that are used to package tactics. If you don't have any patterns, describe how different forces influence the choice of tactics. You should give the most attention to what is believed to be the most important of these quality attributes – how to achieve that.

This section especially should record your rationale for the design. It should say what alternatives you considered, and why you chose the one you did. Your instructors will be paying special attention to what you say here!

If you used “patterns” in your system, in the sense of the Gang-of-Four patterns, and want to elaborate on those, that probably should go in Section 7, as a part of your Framework discussion.>

# 6. Views

<These are the pictures and their explanations. Provide an introductory paragraph listing the purpose of each view. What kinds of views do you need? See the Viewtypes discussion, below. Include each view as a lettered subsection of this section. That is, the first view should be 6.A, the second should be 6.B, etc. Each view should have all of the following subsections:

1. Primary presentation – this will often be a diagram
2. Element catalog
	1. Elements - brief description of each
	2. Relations - brief description of each
	3. Interfaces - for views that include them
		1. Interface identity - unique name
		2. Resources provided - syntax and semantics
		3. Locally defined data types - if used
		4. Exception definitions - including handling
		5. Variability provided - for product lines
		6. Quality attribute characteristics - what is provided?
		7. Element requirements - names of required items, assumptions
		8. Rationale and design issues - why these choices
		9. Usage guide - protocols
3. Context diagram -- how the system relates to its environment
4. Variability guide – how to exercise any variation points
5. Architecture background – why the design reflected in the view came to be
6. Glossary of terms used - alternatively, you may place these in a separate glossary for the whole document
7. Other information - if needed

**Viewtypes:**

There are typically 3 kinds of views you need to show, which translates to 3 or more views in this high-level document:

**Modules** – These are “design time entities” of your system. The structure of the OO design (or other software design methodology). This is probably the same thing as the “framework” described in the next section. So, Section 7 becomes effectively an extension of the picture and the basic explanation of it given here. Often, there are multiple versions of this viewtype – say, a [UML class diagram](http://www.agilemodeling.com/artifacts/classDiagram.htm) showing OO decomposition, generalization and uses; and then a view of the software “layers” in the system, perhaps showing how your software uses third-party software.

**Component-and-connector (C&C)** – This shows the “run time entities” of your system. How is it all supposed to work? For example, how does a “message” or “transaction” get through your system? Or, how does the main use case happen, in terms of pieces of the system interacting? The explanation should be “stream of execution.” Often, your system’s “style” comes out the most in this figure, and readers can see the pipes-and-filters, or client-server design. Interfaces to other systems would be at least generally described here. There are UML diagrams you can use for these, too – see [SEI Article](http://www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr008.pdf).

**Allocation** – This one always shows hardware as well as software, a larger picture of the system in its environment. Especially, it shows what software on what box talks to what other software on its box. The links between the boxes should be as informative as possibly in their legend, saying things like what protocols or interfaces are used, and what the capacity of these links will be. However, there are additional kinds of allocation views that you may also need, like one showing how it will be deployed or installed.

**Data** – These include things like entity-relationship diagrams for the database. Now, in this architecture document, you might only show the most critical parts of that design, since otherwise you could easily have half the document describing the data!>

**User interface** – Typically these interaction design images appear in a more detailed design spec. However, as with data, an exemplary one which gets across a major point about your design could be appropriate. The software framework supporting user interaction would be described in Section 7.

# 7. Framework

<It’s presumed that, as you write this, or shortly thereafter, you create the high level design for your system. This is its “framework.” Describe the components of the framework constructed by the development team. You may want to use a diagram or refer to one of the views in the previous section. Describe how the framework enables/enforces/facilitates the software architecture.

A framework is a general OO design that makes it easy to attach more detailed pieces. So, it has, for example, the most general “parent” classes of the system. And it shows how these connect to others. Typically a framework is full of generics, iterators, interfaces, methods designed to be overloaded, and all that good OO stuff. The description should be supported by a UML class diagram which shows the connections in the whole system (probably in Section 6, above).

This is the section where things like “[model view controller](http://en.wikipedia.org/wiki/Model-view-controller)” and [“Gang of Four” patterns](http://en.wikipedia.org/wiki/Design_Patterns) would be a part of the discussion. However, if you were going to fully flesh out the description of even your framework, that would likely be too long for this high-level intro-to-everything.>

# 8. Acknowledgements

<Acknowledge the help of the customers and developers. Also give thanks for any other sources of material you used, such as authors of patterns.>

# 9. References

<Include citations here, both internal to the team and external. Use a standard format, such as IEEE (http://standards.ieee.org/guides/style/section7.html) or ACM (http://moon.cse.yzu.edu.tw/acm/docs/ihhuang/citation/). Use a standard format, such as IEEE (<http://standards.ieee.org/guides/style/section7.html> ). >

# 10. Revision history

<Table containing the revision number, date, who made the change, what the change was>

# 11. Appendices

<If you have other material you want to include, place it here. For example, you might include a glossary of technical terms and abbreviations.>

1. If you write separate, additional specs, these often are on separate subjects. For example, an Interaction Design spec with all the user interface information, a database design spec, use cases detailed with design information, an OO design document, and an enlargement of the “supplementary spec” with all the other functional and non-functional design data. [↑](#footnote-ref-2)
2. If the regular link doesn’t work, see copy [here](Garlan%20%26%20Shaw%20intro_softarch.pdf). [↑](#footnote-ref-3)