

CS2101 Effective Communication for Computing Professionals

Writing the Developer Guide

Why learn to write user and developer guides?



Software Developer

SmsDome Pte Ltd

Posted On :21 Sep 2017

✓ Job Summary

- Salary: \$ 3500 - 5250
- Location: Singapore - North
- Work Type: Permanent / Full Time
- Min. Education Level: Degree
- Field of Study: Computer Science
- Years of Experience: 5
- Skills: Communications skills, strong programming skills, Interpersonal skills

Job Descriptions

Job description:

- Responsible for the development of systems and projects through the entire application development life cycle
- Identify modifications needed and implement enhancements in existing applications to meet changing requirements
- Perform database administration
- Investigate and resolve technical issues
- **Produce technical documentation for new and existing applications**
- Provide technical support outside normal business hours when required
- Provide Level 3 technical support



Software Engineer

Nidec Singapore Pte Ltd

- Challenging on the Global Standards
- For Everything That Spins and Moves
- Research and Development for the Future

JOB DESCRIPTION

The position requires a highly motivated candidate to carry out creating automation tools based on our products with various commercial/non-commercial CAE software to enhance the strength of each package.

The main task for the software engineer will be:

- Assisting CAE related activities by developing automation codes for thermal & fluid flow, structural, and electromagnetic simulations on motors, cooling fans and other related products using various CFD/FEA tools
- Validation and verification of automation process
- **Preparation of technical documents and manuals based on the results obtained**
- Participation in technical discussion with CAE team members

Learning outcomes

By the end of the session, you will be able to

- understand the purpose of a developer guide, and its readers' needs
- consider what makes a developer guide user-friendly and reader-focused
- analyze a sample developer guide to identify its strengths and areas for improvement
 - a skill useful for self-editing and peer review
- consider the needs of the reader when writing/revising your own developer guide

Important note

What you learn about writing the developer guide are general technical writing principles.

- Your job is to apply these principles to writing your own CS2103T/CS2113T developer guide.
- You will need to apply these principles to the best of your ability, given the guidelines and constraints of the module and the media/platform used to display the developer guide.
- The developer guide will not be graded in CS2101.

This lesson is not meant to tell you exactly what you need to write for CS2103T/CS2113T.

Functions of Software Documentation

Tutorials

- learning-oriented, allows the newcomer to get started; the tutorial enables the learner to make sense of the rest of the documentation, and the software itself.

How-to guides

- goal-oriented, shows how to solve a specific problem, a series of steps.

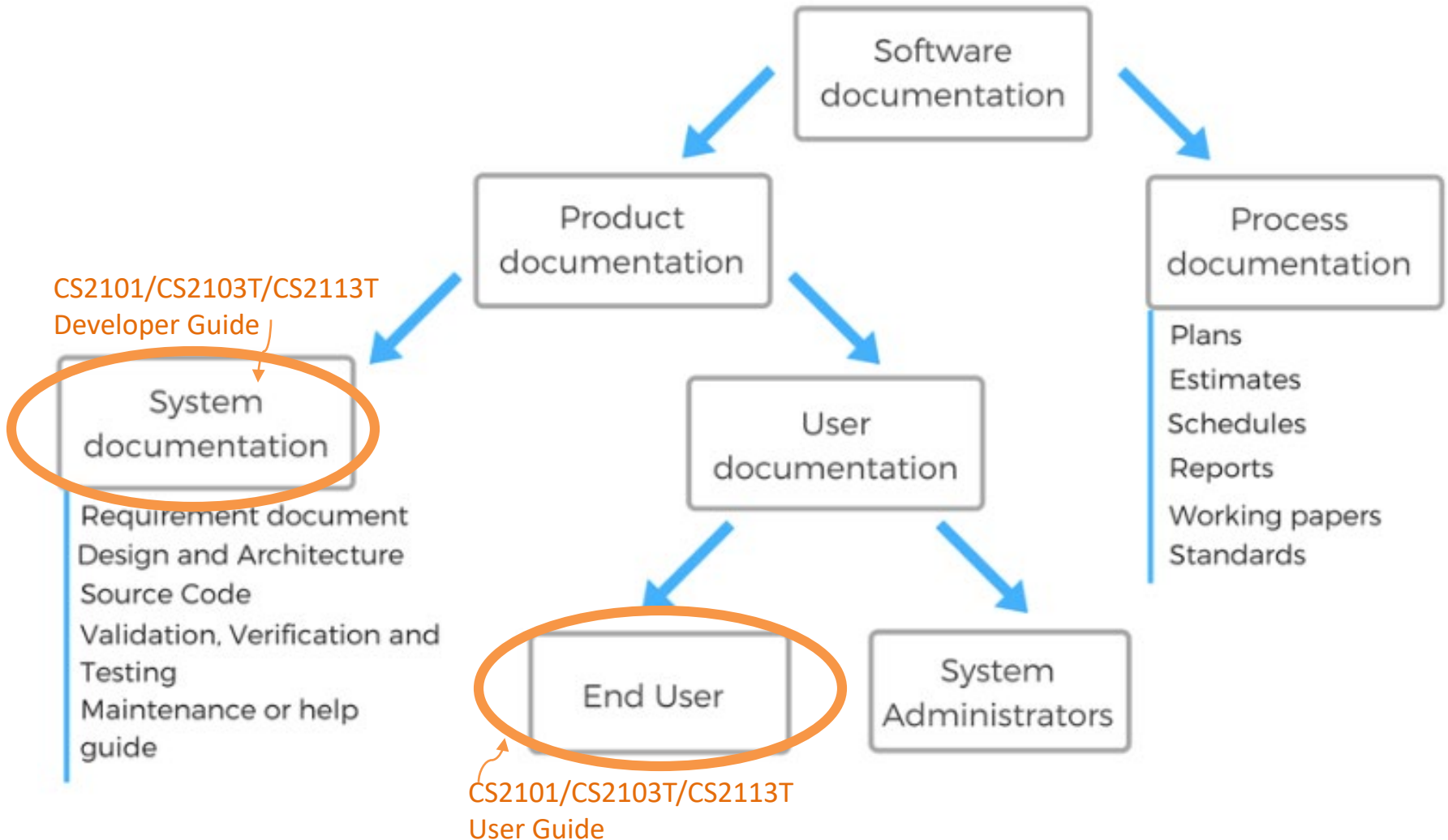
Explanation

- understanding-oriented, explains, provides background context

Reference guides

- information-oriented, describes the machinery and how to operate it.

Documentation types



System documentation

- e.g. Developer Guides
- describe the system itself and its parts
- includes requirements documents, design decisions, architecture descriptions, program source code, and help guides

User documentation

- e.g. User Guides
- mainly prepared for end-users of the product and system administrators
- includes tutorials, user guides, troubleshooting manuals, installation, and reference manuals

Discuss

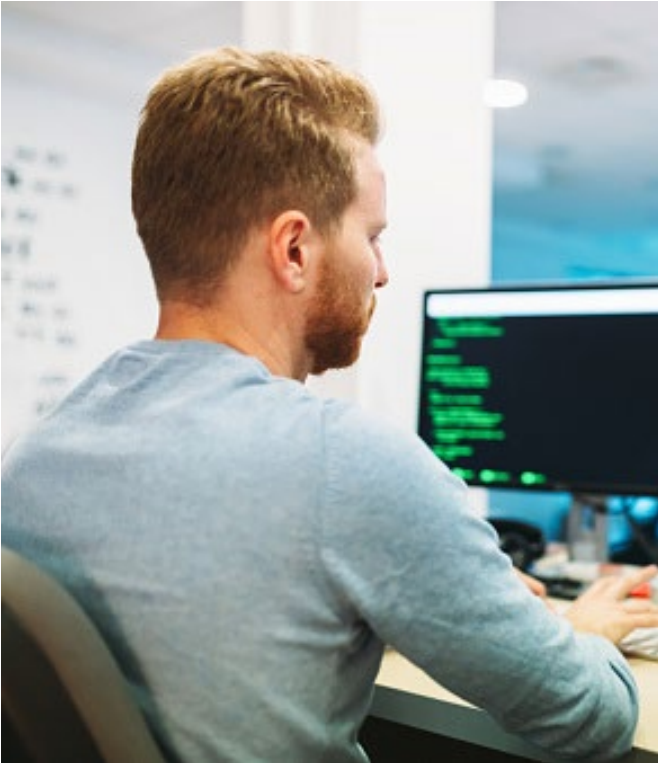
1. What is a developer guide?
2. Who reads developer guides?
What do they use it for?
 - Would they have all the required technical knowledge to understand everything in the developer guide?
3. What contents are there in developer guides?
4. How do readers use the developer guide?
What are their expectations?
5. What are the implications of all these on the DG?



1. What is a Developer Guide?

- Documents how a system or a component is **designed, implemented and tested**
- Contains description of **architecture** of software, **detailed specifications** on smaller pieces of the design, and **outline** of all parts of the software and **how they will work**
- Used to maintain, upgrade, evolve, and if necessary, restore the system
- Needs to be up-to-date and reflect changes to the system.

2. Who reads developer guides?



Program managers

- to evaluate that the system architecture and design support the requirements

Development engineers

- to understand the architecture and follow the design to build the system

Administrators

- to understand the internal workings of the system in order to administer the system effectively

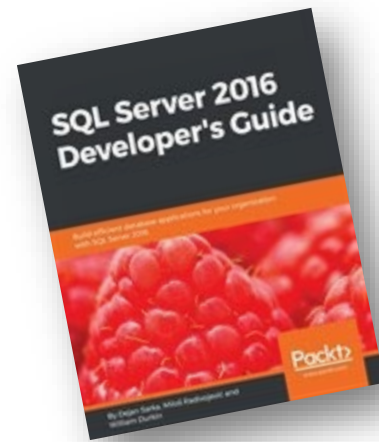
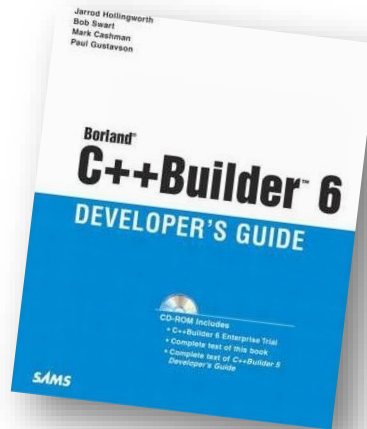
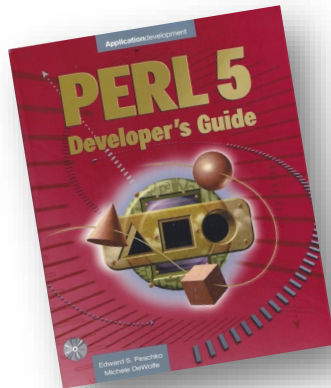
Operators

- to improve productivity while using the system on daily basis

Maintenance engineers

- to understand how the system was built in order to be able to perform any enhancement or reengineering work.

3. What contents are there in a Developer Guide?



Contents of a Developer Guide


- **Introduction:** Software overview, Purpose of the document and scope
- **Design descriptions:** Description of the system and subsystem inputs and outputs relative to the user/operator; i.e. how user commands are implemented by the system. Presents usage scenarios to illustrate the mechanisms using class diagrams, sequence diagrams, activity diagrams, notable algorithms, etc.
- **Architecture:** Describes the major components of the software and their roles, how they are organized, and how they interact with each other
- **Design considerations:** Explains reason for selecting a specific design over others
- **Code examples**, if relevant
- Instructions for **setting up** and **testing**
- **Known issues** and future work



**What contents are there in the
AB3 Developer guide?**

What contents are there in the AB3 Developer guide?

AB-3

[User Guide](#) [Developer Guide](#) [About Us](#) 

Developer Guide

- [Acknowledgements](#)
- [Setting up, getting started](#)
- [Design](#)
 - [Architecture](#)
 - [UI component](#)
 - [Logic component](#)
 - [Model component](#)
 - [Storage component](#)
 - [Common classes](#)
- [Implementation](#)
 - [\[Proposed\] Undo/redo feature](#)
 - [Proposed Implementation](#)
 - [Design considerations:](#)
 - [\[Proposed\] Data archiving](#)
- [Documentation, logging, testing, configuration, dev-ops](#)
- [Appendix: Requirements](#)
 - [Product scope](#)
 - [User stories](#)
 - [Use cases](#)
 - [Non-Functional Requirements](#)
 - [Glossary](#)
- [Appendix: Instructions for manual testing](#)
 - [Launch and shutdown](#)
 - [Deleting a person](#)
 - [Saving data](#)

What is said on the other side

Individual Expectations on Documentation

- **Objective:** showcase your ability to write both *user-facing documentation* and *developer-facing documentation*.
- **Expectation** **Update the User Guide (UG) and the Developer Guide (DG) parts** that are related to the enhancements you added. The minimum requirement is given below. (Reason: Evaluators will not be able to give you marks unless there is sufficient evidence of your documentation skills.)
 - UG: 1 or more pages
 - DG: 3 or more pages
- **Tip: If the UG/DG updates for your enhancements are not enough to reach the above requirements,** you can make up the shortfall by documenting 'proposed' features and alternative designs/implementations.
- **Expectation** **Use at least 2 types of UML diagrams in your DG updates** i.e., diagrams you added yourself or those you modified significantly.

<https://nus-cs2103-ay2021s2.github.io/website/admin/tp-expectations.html>

<https://nus-cs2113-ay2021s2.github.io/website/admin/tp-expectations.html>

TP: Week 10

⊖ tP week 9: v1.0

tP week 11: v2.0 ⊕

tP week 10: mid-v2.0 ★★

- 
1. 👤 Start the next iteration
 2. 👤 Update the DG with design details **🕒 IMPORTANT**
 3. 👤 Smoke-test CATcher **🕒 IMPORTANT** **P**
 4. 👤 Ensure the code RepoSense-compatible

<https://nus-cs2113-ay2021s2.github.io/website/admin/tp-w10.html>

2 Update the DG with design details IMPORTANT

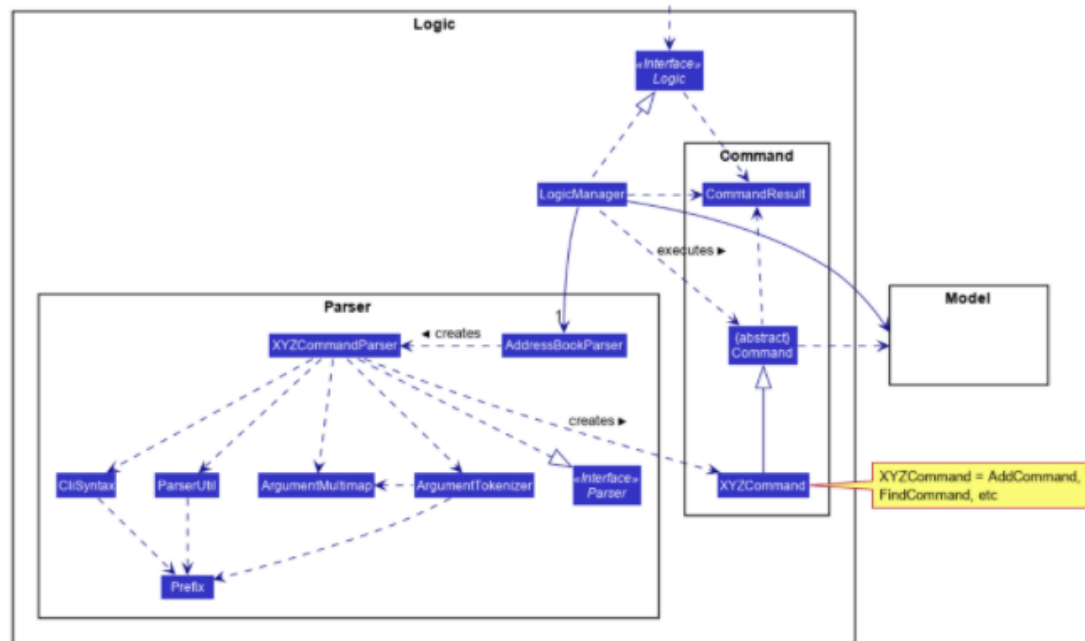
- **Update the Developer Guide** as follows:
 - Each member should describe the implementation of at least one enhancement she has added (or planning to add).
Expected length: 1+ page per person
 - Describing the design at a multiple-levels (e.g., first, describe at *architecture-level*, then describe at *component-level*) is optional. It is also acceptable to have one *Design & Implementation* section in which you describe the entire thing at the class- and object-level.
 - The description can contain things such as,
 - How the feature is implemented (or is going to be implemented).
 - Why it is implemented that way.
 - Alternatives considered.

DG Tips

- **Aim to showcase your documentation skills.** The stated objective of the DG is to explain the implementation to a future developer, but a secondary objective is to serve as evidence of your ability to document deeply-technical content using prose, examples, diagrams, code snippets, etc. appropriately. To that end, you may also describe features that you plan to implement in the future, even beyond v2.1 (hypothetically).

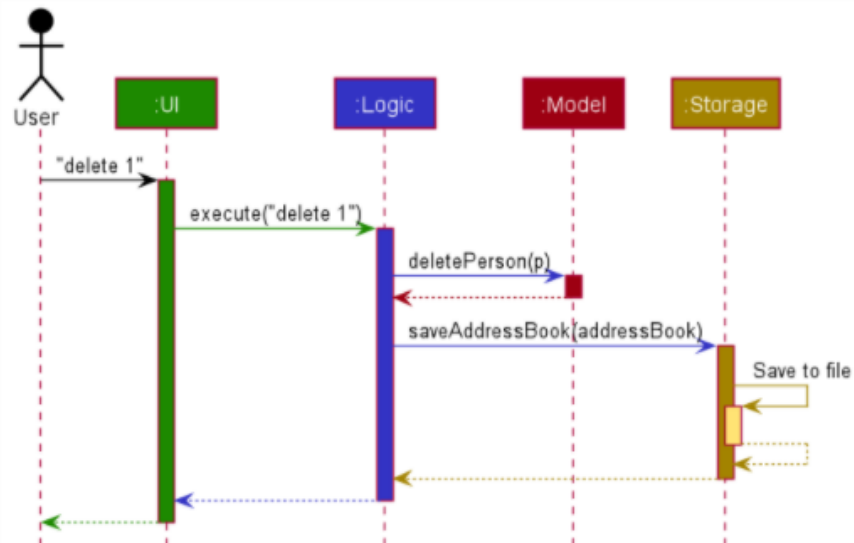
For an example, see [the description of the undo/redo feature implementation in the AddressBook-Level3 developer guide](#).

- **Use multiple UML diagram types.** Following from the point above, try to include UML diagrams of multiple types to showcase your ability to use different UML diagrams.
- **Diagramming tools:**
 - AB3 uses PlantUML (see the guide [Using PlantUML @SE-EDU/guides](#) for more info).
 - You may use any other tool too (e.g., PowerPoint). But if you do, note the following:
 - Choose a diagramming tool that has some 'source' format that can be version-controlled using git and updated incrementally (reason: because diagrams need to evolve with the code that is already being version controlled using git). For example, if you use PowerPoint to draw diagrams, also commit the source PowerPoint files so that they can be reused when updating diagrams later.
 - Use the same diagramming tool for the whole project, except in cases for which there is a *strong* need to use a different tool due to a shortcoming in the primary diagramming tool. Do not use a mix of different tools simply based on personal preferences.
 - ⓘ Can IDE-generated UML diagrams be used in project submissions? Not a good idea. Given below are three reasons each of which can be reported by evaluators as 'bugs' in your diagrams, costing you marks:
 - They often don't follow the standard UML notation (e.g., they add extra icons).
 - They tend to include *every* little detail whereas we want to limit UML diagrams to important details only, to improve readability.
 - Diagrams reverse-engineered by an IDE might not represent the actual design as some design concepts cannot be deterministically identified from the code. e.g., differentiating between multiplicities `0..1` vs `1`, composition vs aggregation



How the architecture components interact with each other

The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `delete 1`.











- **Keep diagrams simple.** The aim is to make diagrams *comprehensible*, not necessarily *comprehensive*.

Ways to simplify diagrams:

- **Omit less important details.** Examples:
 - a class diagram can omit minor utility classes, private/unimportant members; some less-important associations can be shown as attributes instead.
 - a sequence diagram can omit less important interactions, self-calls.
- **Omit repetitive details** e.g., a class diagram can show only a few representative ones in place of many similar classes (note how the *AB3 Logic class diagram* shows concrete **Command* classes using a placeholder *XYZCommand*).
- **Limit the scope of a diagram.** Decide the purpose of the diagram (i.e., what does it help to explain?) and omit details not related to it.
- **Break diagrams into smaller fragments** when possible.
 - If a component has a lot of classes, consider further dividing into sub-components (e.g., a Parser sub-component inside the Logic component). After that, sub-components can be shown as black-boxes in the main diagram and their details can be shown as separate diagrams.
 - You can use *ref* frames to break sequence diagrams to multiple diagrams.
- **Use visual representations** as much as possible. E.g., show associations and navigabilities using lines and arrows connecting classes, rather than adding a variable in one of the classes.
- For some more examples of what NOT to do, see [here](#).
- **Integrate diagrams into the description.** Place the diagram close to where it is being described.
- **Use code snippets sparingly.** The more you use code snippets in the DG, and longer the code snippet, the higher the risk of it getting outdated quickly. Instead, use code snippets only when necessary and cite only the strictly relevant parts only. You can also use pseudo code instead of actual programming code.
- **Resize diagrams** so that the text size in the diagram matches the the text size of the main text of the diagram. See [example](#).

Week 13 [Fri, Nov 5th] - Project

tP:  v2.1

- 
1.  Submit deliverables  Mon, Nov 8th 2359
 2.  Submit the demo video  Wed, Nov 10th 2359
 3.  Prepare for the practical exam
 4.  Attend the practical exam  during the lecture on Fri, Nov 12th


- The main content you add should be in the `docs/DeveloperGuide.md` file (for ease of tracking by grading scripts).
If you use PlantUML diagrams, commit the diagrams as `.puml` files in the `docs/diagrams` folder.
- **Should match the v2.1 implementation.**
- **Follow the AddressBook-Level3 (AB3) DG structure.** Sections to include in your DG:
 - Design: similar to AB3 DG except,
 - you may omit the *Architecture* section (no penalty)
 - if you have not organized the code into clearly divided components (no penalty if you didn't), you can use a single class diagram (if it is not too complicated) or use several class diagrams each describing a different area of the system.
 - Implementation: similar to AB3 DG
 - Appendix A: Product Scope
 - Appendix B: User Stories
 - Appendix C: Non Functional Requirements
 - Appendix D: Glossary
 - Appendix E: Instructions for Manual Testing (more details below)
- **OPTIONAL You can include proposed implementations of future features.**
- **! Include an appendix named *Instructions for Manual Testing*, to give some guidance to the tester to chart a path through the features, and provide some important test inputs the tester can copy-paste into the app.**
 - **Cover all user-testable features.**
 - **No need to give a long list of test cases** including all possible variations. It is upto the tester to come up with those variations.
 - Information in this appendix should *complement* the UG. **Minimize repeating information that are already mentioned in the UG.**
 - **Inaccurate instructions will be considered bugs.**

- The main content you add should be in the `docs/DeveloperGuide.md` file (for ease of tracking by grading scripts).

If you use PlantUML diagrams, commit the diagrams as `.puml` files in the `docs/diagrams`

• **Follow the AddressBook-Level3 (AB3) DG structure.** Sections to include in your DG:

- Design: similar to AB3 DG except,
 - you may omit the *Architecture* section (no penalty)
 - if you have not organized the code into clearly divided components (no penalty if you didn't), you can use a single class diagram (if it is not too complicated) or use several class diagrams each describing a different area of the system.
- Implementation: similar to AB3 DG
- Appendix A: Product Scope
- Appendix B: User Stories
- Appendix C: Non Functional Requirements
- Appendix D: Glossary
- Appendix E: Instructions for Manual Testing (more details below)

-  **Include an appendix named *Instructions for Manual Testing***, to give some guidance to the tester to chart a path through the features, and provide some important test inputs the tester can copy-paste into the app.
 - **Cover all user-testable features.**
 - **No need to give a long list of test cases** including all possible variations. It is upto the tester to come up with those variations.
 - Information in this appendix should *complement* the UG. **Minimize repeating information that are already mentioned in the UG.**
 - **Inaccurate instructions will be considered bugs.**



Developer Guide

- **Setting up, getting started**
 - **Design**
 - Architecture
 - UI component
 - Logic component
 - Model component
 - Storage component
 - Common classes
 - **Implementation**
 - [Proposed] Undo/redo feature
 - Proposed Implementation
 - Design consideration:
 - Aspect: How undo & redo executes
 - [Proposed] Data archiving
 - **Documentation, logging, testing, configuration, dev-ops**
 - **Appendix: Requirements**
 - Product scope
 - User stories
 - Use cases
 - Non-Functional Requirements
 - Glossary
 - **Appendix: Instructions for manual testing**
 - Launch and shutdown
 - Deleting a person
 - Saving data
-

Activity

In your teams, complete the **Developer Guide Journey of Discovery Worksheet** available on LumiNUS.

Be ready to share your answers with the rest of the class.

The following slides are notes for this lesson.
You tutor may not go through every single slide.
The contents of these slides will be covered in
your discussion activity.

Instructions:

Discuss, and get ready to present your answers.

Practice semi-impromptu speaking.

Practice good audience awareness!

Bear in mind that your audience may not have done the same analysis you have, so they may need a little more context and processing time...



Tip!

Talk TO your audience, not AT them.

Introduction

1. Describes the software
2. Describes the purpose of the developer guide, who its intended audience is and perhaps how to use it
3. Provides a preview of the contents of the developer guide
4. It may even include some information about the team of developers, and how to contact them.



The intro sets the tone of the document, and gives the user an overall impression of the brand. So it's important to set the tone right – by sounding inviting and friendly, yet professional.

Welcome to Tastypie!

Tastypie is a webservice API framework for Django. It provides a convenient, yet powerful and highly customizable, abstraction for creating REST-style interfaces.

- Getting Started with Tastypie
- Interacting With The API
- Tastypie Settings
- Using Tastypie With Non-ORM Data Sources
- Tools

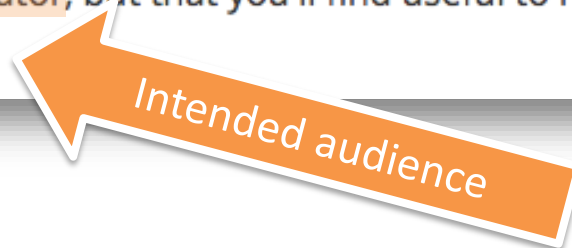


Use of adjectives to
“brand” software

Example:
Brief description of *Tastypie*

Developer guide

There are lots of ways to contribute to the Mozilla project: coding, testing, improving the build process and tools, or contributing to the documentation. This guide provides information that will not only help you get started as a Mozilla contributor, but that you'll find useful to refer to even if you are already an experienced contributor.



Describes what this DG can do for the reader/what the reader can do with this guide.

Notice that it TALKS to and addresses the reader directly – use of YOU language.

Example:

A brief description of the purpose of the Mozilla guide, and a reference to the target readers.

Developer's Guide

The Blogger Data API allows client applications to view and update Blogger content in the form of Google Data API feeds.

Your client application can use the Blogger Data API to create new blog posts, edit or delete existing blog posts, and query for blog posts that match particular criteria.

This developer's guide consists of the following sections. Note that some client libraries have been upgraded to work with version 2.0 of the Data API, while other client libraries use version 1.0.

- [Protocol](#) (2.0)
- [.NET](#) (2.0)
- [Java](#) (2.0)
- [JavaScript](#) (1.0)
- [PHP](#) (1.0)
- [Python](#) (1.0)



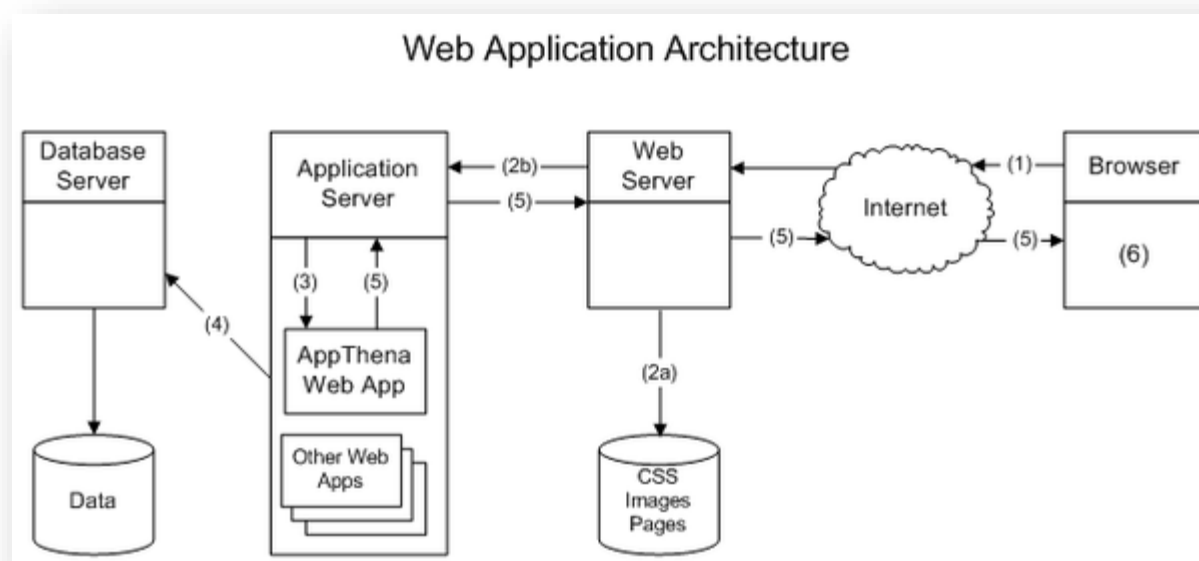
About this DG

Example:
A brief explanation of the purpose of the *Blogger Developer Guide*, and a preview of contents.

Again, notice use of YOU language.

Design

Explains the design of the app: The architecture and its components



Architecture

- To explain the organization of the software system
 - To make it easier for new programmers to become familiar with the structure of the software system
- Identify major system components and describe their static attributes and dynamic patterns of interaction
- Use a mixture of diagrams (class, sequence, dataflow diagrams) and prose.

1.0 Introduction

The Campus Connect Team B (CCB) Architecture Document is designed to illustrate and identify the high level architecture systems used to design and implement the Campus Connect application. The document contains an overall view of the system hierarchy, logical views of the system components, and a process view of the system's communication.

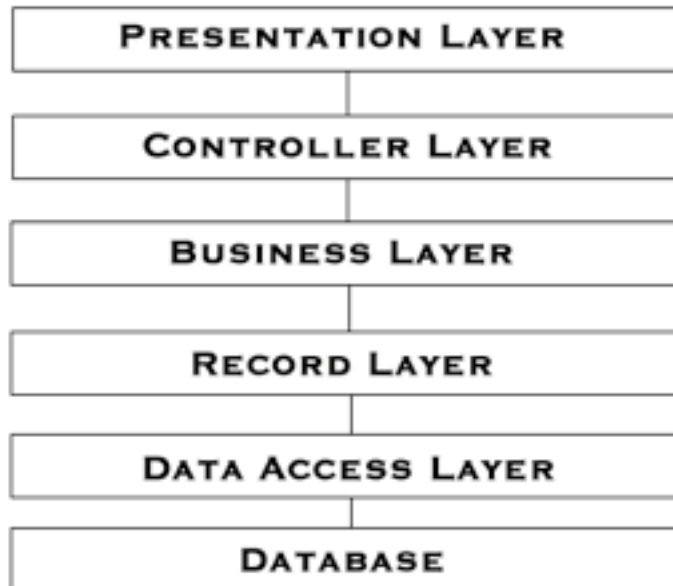
Example:

A brief intro to the Architecture section, and contents

2.0 High Level Hierarchy

2.1 Hierarchy Diagram

ARCHITECTURE HIERARCHY



2.2 Hierarchy Description

The architecture system for the CCB application is an n-tier architecture. This architecture system is designed to allow for proper information hiding, modular components, and single system dependencies. The abstraction of the presentation layer, and consequently the User Interface (UI), allow for a flexible pipeline for the optimization of the UI to meet customer needs and expectations. There are multiple layers between the Presentation Layer and the lowest level, due to the significant challenges present in the optimization and control of the Processes design. The Database layer is the lowest level in the hierarchy, and is an abstraction used to represent both text data (in the form of XML files), and serial device data.

Example:

A diagram showing the architecture hierarchy, and a brief explanation of the components.

Not only describes the design, but also explains the concepts/reasons behind the design

4.1 High-Level Design (Architecture)

The high-level view or architecture consists of 5 major components:

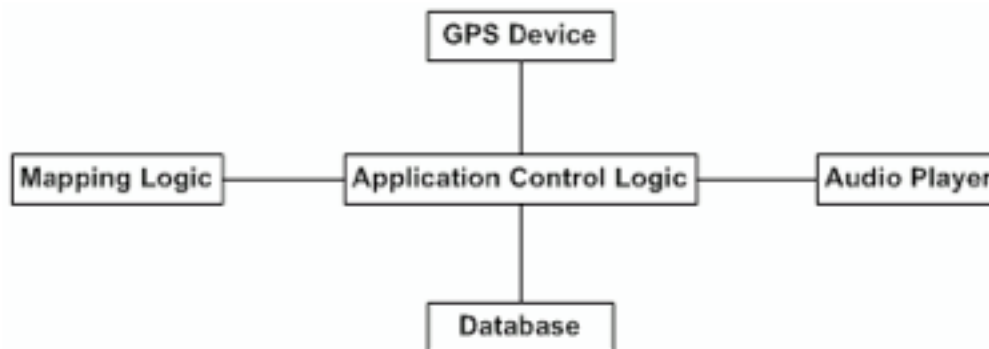


Figure 2 System Architecture

Example:
Labeled architecture
diagram and a brief
description of each
component

- The **GPS device** provides the user's location on campus (longitude and latitude coordinates). In basic mode, the user's position is used to decide which buildings to announce.
- The **Database** is a central repository for data on buildings, their locations and associated audio segments.
- The **Audio Player** controls playback of audio files.
- Given a position on earth, the **Mapping Logic** will calculate nearby buildings.
- The **Application Control Logic** is the main driver of the application. It presents information to the user and reacts to user inputs.

2.1. Architecture

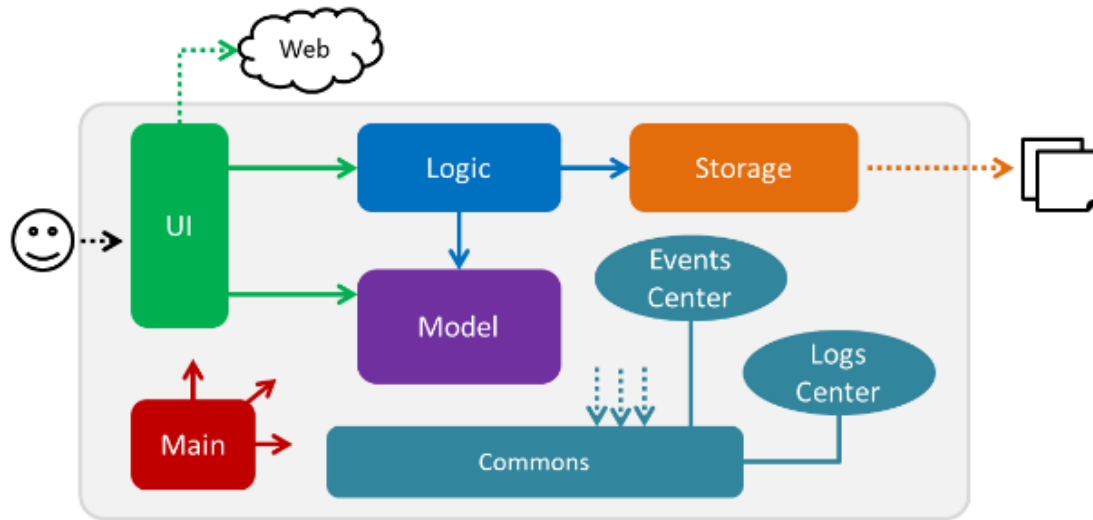


Figure 2.1.1 : Architecture Diagram

The *Architecture Diagram* given above explains the high-level design of the App. Given below is a quick overview of each component.

Use the top-down approach

- start with a general overview (the architecture of the software), before describing specific components (link each mechanism to the relevant part of the architecture)
- eg which part(s) of the architecture the mechanism is located in/related to...)



Implementation

- Explains the mechanism of various functions
- This is where you insert documentation for the features you've added to the software

CAUTION:

- Check that your formatting/layout, language and complexity of content is consistent with the rest of the document (and other group members').

Discover...

Look at the Implementation section in your sample Developer's Guide.

What information is inserted into this section?



FOR WRITING YOUR DEVELOPER GUIDE

General Tips

- Write your document **as you code**.
- As a **team**, decide on a **common and consistent style** and **format**.
- Write your sections **individually**.
- **Test each other's understanding** of the text, and ensure **coherence** throughout the document.
- Appoint an editor to ensure **language accuracy** and **consistent formatting**.
- Make your document **A**ccurate, **C**omplete and **E**asy to read – **ACE** it!



General Tips

Use **labeled diagrams** to support explanations. Remember to make reference to these diagrams in the text; eg *Figure 1 shows..., In Figure 1..., ... (Figure 1).*

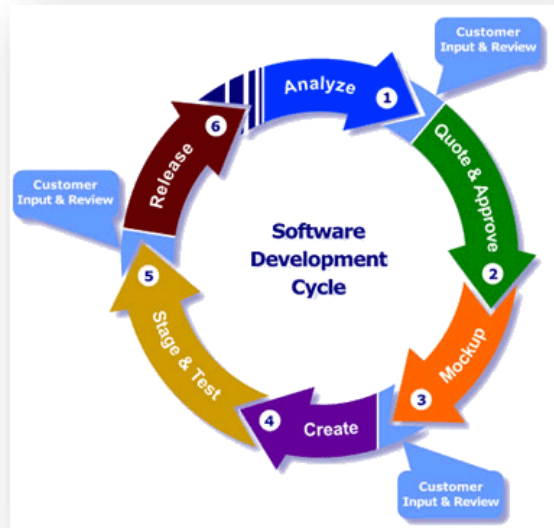


Figure 1: Software Development Cycle

General Tips

- Write concisely but include **important** details
- Be relevant but emphasize **unique** aspects
- Avoid duplicating information
 - describe **similarity** in one place
 - highlight **differences** in another place



Formatting and Layout

- Prepare an **outline** with different tiers of information.
- Decide on the **template** – font type, size, style for each level, *markup*.
- Ensure **parallelism** in text for the headings, sub-headings and lists.
- Think about the **graphics** – formal and informal graphics.



Proofreading and Editing

- Spell check
- Grammar check
- Editorial check
- Graphics/ layout check
- Content check – relevance and conciseness
- Consistent use of **markup**



Audience-Centredness

- Style and tone – some allowance for **informality**
- Be consistent with the use of pronouns “**we**” and “**you**”.
- *Use **direct, active verbs** rather than passive ones, eg
 - *a run configuration can also be created* (passive) vs *you can also create a run configuration* (active)

Audience-Centredness

- Use inline **markup** liberally and **consistently**
“Similarly, use **markup** for things like
`code samples`, **entry** versus *output*.”
- Write in **short paragraphs**.
- Use a variety of **structural elements**
 - tables, lists, code blocks, **callouts (notes)**

Audience-Centredness (example)

4.1.1 MainApp

MainApp contains root, which is a BorderPane with display as the center node and input as the bottom node. An event filter is added to root which allows it to detect key presses. For now, it only recognises CTRL+M as a shortcut to toggle the sidebar. You can further extend it to accept other key presses.

MainApp interacts directly with Logic to minimise coupling between the individual GUI elements and application logic.

root, BorderPane



CTRL+M



- ✓ Simple present tense: direct and active verbs
- ✓ Clear in-line markups
- ✓ Address reader (YOU language) where applicable

In Summary...

A developer guide is excellent if it can completely **eradicate the need for a new developer to consult the original team**; i.e. students should aim towards letting their guide replace them.



Remember:

Four key elements for effective guides



1. Consideration for audience's needs
2. Consistent style and formatting
3. Labelled and referenced visuals
4. Careful proof reading



Resources

- Documenting APIs. A guide for technical writers. (2017). Retrieved from https://idratherbewriting.com/learnapidoc/docapis_codesamples_bestpractices.html
- Kaplan-Moss, J. (2009). What to write. Retrieved from <http://jacobian.org/writing/what-to-write/>
- Kaplan-Moss, J. (2009). Writing great documentation: Technical style. Retrieved from <http://jacobian.org/writing/technical-style/>
- Fox, P. (2011). The six pillars of complete developer documentation. ProgrammableWeb. Retrieved from <http://www.programmableweb.com/news/six-pillars-complete-developer-documentation/2011/09/12>
- Martinez, D., Peterson, T., Wells, C., Hannigan, C. & Stevenson, C. (2011). *Kaplan technical writing: A comprehensive resource for technical writers at all levels*. New York: Kaplan Publishing.
- Rajapakse, D. C. (2010), online book Practical Tips for Software-Intensive Student Projects V3.0. Retrieved from: http://www.comp.nus.edu.sg/~damithch/guide3e/pdf/Practical_Tips_for_Software_Intensive_Student_Projects%20%5B3e%5D-review%20copy.pdf

Samples of authentic developer guides

- Apple Developer Documentation: Contacts
<https://developer.apple.com/documentation/contacts>
- Mozilla Developer Guide
https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide
- U.S. Government Printing Office Federal Digital System System Design Document 
https://www.govinfo.gov/media/FDsys_Architecture.pdf
- Highwater Design Specification Document 
http://www.cci.drexel.edu/seniordesign/2016_2017/HighWater/HighWaterDesignDocument.pdf

Activity

Discuss your key takeaways from today's session.

Deliverables for UGDG

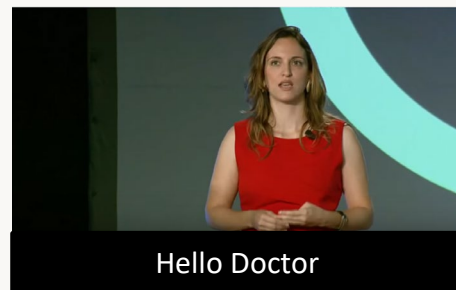
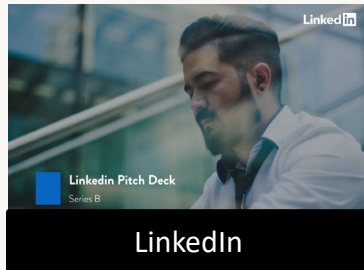
- Week 10: UGDG in-class peer review
- After peer review
 - Submit peer review reports to LumiNUS
 - Make the necessary revisions to UGDG for your testers
- On Week 13 Friday
 - Submit UGDG final copy to CS2101.
 - UG is graded in CS2101 – 20%.
 - DG is not graded in CS2101.

****Deadlines for UGDG drafts may be different for CS2103T/CS2113T**

Prepare for the next lesson

Look at these 5 samples.

Which one would you categorise as a **product demo**, and which one a **pitch**? Why?



Be ready to share you answers in the next lesson.