Quantico Cloud Application Development
Project Manual

Charles C. Carter

July 30, 2020

Contents

Preface

The MSSA curriculum consists of four different technologies. They are:

- C# programming
- T-SQL Query Fundamentals
- ASP.NET MVC web app development
- Azure clould app development

These four technologies give students a real proficiency in cutting edge technology and prepare students to enter the workforce. However, they make no attempt to teach students *how* to develop applications. For example, a real ability to develop applications requires an awareness of the following, among other things:

- the software life cycle
- the software process
- structured programming
- program logic and design
- patterns and object oriented analysis
- version control
- unit testing
- code security
- communication skills

The purpose of this project is two fold. First, it gives the students experience in putting their skills to work creating an application from scratch. Second, it introduces the skills necessary to develop applications but which are not specifically covered by the curriculum. As a side effect, it allows students to build a portfolio demonstrating their skills to prospective employers. It in no way attempts comprehensive coverage of any aspect of application development or software engineering. Instead, it is very much a work in progress which is designed to introduce students to application development from an integrated perspective.

The project in outline consists of five parts. Part I on page 7 consists of researching and choosing a project and making a presentation summarizing their chosen project. Part II on page 15 develops a database supporting their project. Part III on page 22 includes a study of software design principles, especially the software process, which consists of these activities: (a) requirements, (b) analysis, (c) design, (d) implementation, and (e) testing. Part IV on page 39 consists of the construction of an application in three iterations, where the student will iteratively complete the design and implementation of their project. Part V on page 51, the final part, consists of the project presentation. The appendix on page 53 contains references to Markdown, `git`, and Github.

Chapter 0

Introduction

0.1 Overview

The MSSS Cloud Application Development project consists of a project summarized in table 1 and detailed in this manual. This consists of five phases: project inception, the database component, the software process, an application in three iterations, and a final presentation. Much of the material covered by the CAD project consists of *how* a developer develops an application. This material is beyond that covered in the texts and training materials. The two purposes of the projects are to (1) give students the opportunity to reflect on what they are learning and apply the material to a "real world" project, and (2) allow them to construct a portfolio that they can use for the purpose of gaining employment as an application developer.

0.2 Project Inception

The first three weeks of the MSSA program will consist of the selection of the project. Students are not expected to have the technical skills necessary to begin the project, but are expected to have a layman's grasp of software applications and an interest in some knowledge domain that they can leverage throughout their course. The focus of this phase is written and spoken communication skills.

0.2.1 Preliminary research

Students will begin thinking about the kind of project they would want to showcase for their project. They should list the specific functionalities of their chosen kind of project, and so some preliminary investigation toward implementing the project. All projects should include a database component (model), a programming component (controller), and an interface component (view). The deliverable should be a five paragraph document listing several specific projects they would be interested in doing, and a short description of the characteristics of each project. See chapter 1 for more information.

0.2.2 Project selection

Students will select one project from their list to focus on for their project They should make a formal investigation of the requirements of the project. If possible, they should identify similar applications that have been written. The deliverable is a written paper. The project proposal should contain a sufficient description of the project that would allow construction of the project to begin, including a discussion of the supporting database, the program code, and the user interface. See chapter 2 for more information.

	Project Inception	
Ch. 1	Project exploration	5 paragraph writeup
Ch. 2	Project selection	an oral and written presentation
Ch. 3	Project presentation I	an Information Flow Diagram and requirements list
	Database Development	
Ch. 4	Requirements & conceptual design	Entity Relationship Diagram
Ch. 5	Logical & physical design	Database Diagram
Ch. 6	Database implementation	SQL source listing and presentation II
	Software Development Process	
Ch. 7	Requirements gathering	UML use case
Ch. 8	Requirements analysis	Software requirement specification
Ch. 9	Program design	UML design diagrams
Ch. 10	Program inplementation & testing	Program source listing
Ch. 11	Presentation III	Use case, SRS, UML design drawings
	Application Development	
Ch. 12	Requirements, analysis, design Single Responsibility Principle	Use case, SRS, UML design drawings
Ch. 13	Iteration 1 Implementation Oper-closed Principle	Source listing
Ch. 14	Requirements, analysis, design Liskov Substitution Principle	Use case, SRS, UML design drawings
Ch. 15	Iteration 2 Implementation Interface Segregation Principle	Source listing
Ch. 16	Requirements, analysis, design Dependency Inversion Principle	Use case, SRS, UML design drawings
Ch. 17	Iteration 3 Implementation Code Security	Source listing
	Final Presentation	
Ch. 18		Final Presentation

Table 1: Project schedule

0.2.3 Project presentation I

The students will present their project selection to the class. The deliverable is a (revised) selection paper, and an optional slide deck.

0.3 Database Development

This phase concentrates on using persistent data using a relational database (SQL Server). The outcome of this phase will be a fully functional database.

0.3.1 Requirements and conceptual design

Every database stores data according to some requirements. In this step, students will collect as many of their data requirements as possible. The deliverable is a simple list of data requirements in a plain text or Markdown file. The requirements should be as complete and as detailed as possible. See chapter 3 for more information. Students will also discover the nature of conceptual database design. The deliverable will be an entity-relationship diagram (ERD) showing the conceptual design of their project database. See chapter 4 for more information.

0.3.2 Logical and physical design

Students will discover the nature of logical and physical database design. The deliverable will be an database diagram showing the logical design of their project database. See chapter 5 for more information.

0.3.3 Implementation and presentation II

Students will write a SQL source listing implementing their database. The deliverable will be the SQL source listing. Students shall also give an oral presentation with an optioal slide deck. See chapter 6 for more information.

0.4 Software development process

0.4.1 Requirements gathering

Students will create a list of application requirements. These generally consist of the *functional* requirements of their project. This list should be as exhaustiive and complete as possible. The requirements list provides the foundation of the application, in the sense that you cannot build an appication if you do know know what kid of appication you are building. See chapter 7 for more information.

0.4.2 Requirements analysis

Students will create a software requirements specification (SRS) of the *functional* requirements of their project. (The project does not include *non-functional* requirements.) The deliverable will be the project SRS. See chapter 8 for more information.

0.4.3 Program design

Students will create appropriate design artifacts. These will include a class diagram, a component diagram, a system sequence diagram, and other documents as necessary (e.g., data flow diagrams, state models, etc.) The deliverable will be a collection of design diagrams.See chapter 9 for more information.

0.4.4 Program implementation and testing

Students will implement the logic component of their project. The deliverable will be the source code listings of their implementation. These will not include auxilliary files, such as container folders, configuration files, etc. See chapter 10 for more information.

0.4.5 Presentation III

Presentation: students will optionally prepare a slide presentation and make an oral presentation of the database to the class. Deliverables include the slide presentation and their oral presentation. See chapter 11 for more information.

0.5 Application development

Students will complete an ASP.NET MVC application as described below.[1] The application includes the business rules for the application. This should be accessible from a console interface. It should be a fully functional version of their implemented project.

0.5.1 Iteration 1, Requirements, Analysis, and Design

The deliverables are as follows: (a) a written use case for *one* cohesive set of requirements; (b) an *amended* SRS updated to reflect the use case; and (c) a set of appropriate UML design diagrams.

0.5.2 Iteration 1, Implementation and Testing

The deliverables consist of the source listing of their application, including unit tests if appropriate.

0.5.3 Iteration 2, Requirements, Analysis, and Design

The deliverables are as follows: (a) a written use case for *one* cohesive set of requirements; (b) an *amended* SRS updated to reflect the use case; and (c) a set of appropriate UML design diagrams.

0.5.4 Iteration 2, Implementation and Testing

The deliverables consist of the source listing of their application, including unit tests if appropriate.

0.5.5 Iteration 3, Requirements, Analysis, and Design

The deliverables are as follows: (a) a written use case for *one* cohesive set of requirements; (b) an *amended* SRS updated to reflect the use case; and (c) a set of appropriate UML design diagrams.

0.5.6 Iteration 3, Implementation and Testing

The deliverables consist of the source listing of their application, including unit tests if appropriate.

0.6 Final project presentation

Presentation: students will prepare a written documentation of their project and optionally prepare a slide presentation, and make an oral presentation to the class. Deliverables include the written documentation, the optional slide presentation and their oral presentation.

[1]Students may obtain permission for use of an alternate technology such as UWP.

0.7 Wrap Up

The instructor's obligation throughout the project is to provide guidance for each phase. For example, during the Data phase, he should discuss the database design process including normal forms and integrity constraints. During the Logic phase, he should discuss iterative, incremental processes and UML. During the Interface, he should discuss principles of user interface design.

Instructors should use the project as an opportunity to add value to the curriculum by exposing the students to topics not expressly included in the curriculum, such as version control, software engineering, software quality control (testing), security, design patterns, and so on. This should be at the instructor's discretion.

The project is a *supplement* to the official curriculum. As such, it should enhance the student's experience, and not detract from it by overloading the student with an amount of work that is impossible to do. The emphasis is on understanding the topics presented and building a portfolio, not building a completely functional application.

Part I

Project Selection

Chapter 1

Project Inception

1.1 Preliminary research

Students will begin thinking about the kind of project they would want to showcase for their portfolio. They should think about the specific functionalities of their chosen kind of project, and some preliminary investigation toward implementing the project. All projects should include a data component and an interface component. The deliverable should be a document listing several specific projects they would be interested in doing, and a short description of the characteristics of each project.

You may choose your own project. Here is a list of sample projects for you to consider.

1.1.1 Recommended projects

Club or Organization This might include member registration, a member directory, an event calender, a newsletter, a photo album, a FAQ, or other functions suitable for organizations. Examples would include a social organization, a service organization, a band or orchestra, a church, an athletic team, etc.

Testing system This is an application for automatic testing, similar to the Microsoft certification tests. It generates tests, allows students to take the tests, grades the tests, and reports the scores.

School system This may consist of a student module, an instructor module, a curriculum module, and provides for the assignment of instructors to classes, students to classes, course management, student management, an d instructor management. As example would be the ERAU student information system.

1.1.2 Other projects, not suggested

Electronic learning system This may consist of a collection of courses. Each course may allow for tests, quizzes, exercises, research papers, discussions, etc. As example would be Cougarview.

An auction system. This my consist of a sellers module, a bidders module, a purchase module, and provide for different kinds of sales (i.e., open bidding, bidding with reserve, buy now, etc.). It would include user registrations and appropriate databases of auctions, transactions, etc. An example would be eBay.

Messaging system This would include user registrations, posting messages, sending messages, composing groups, and similar functionality. An example would be Twitter.

Encyclopedia This would include editor registrations, posting entries, editing entries, search functions, analytical functions, and similar functionality. An example would be Wikipedia.

Electronic voting system This would include voter registrations, candidate qualification, casting ballots with appropriate authentication, and calculation of results. An example would be the current voting system in Georgia.

Networking system This would include user registrations, posting employment data, school data, certification data, allow the posting of jobs, etc. An example would be LinkedIn.

Online store This would implement the display of merchandise, a shopping cart, a payment module, and inventory control mechanism, etc. An example would be Amazon.

Database GUI This would implement a graphical front end to a database. Modules would (1) create or drop databases, (2) create, drop, and alter tables, (3) insert, update, or delete data, and (4) run simple queries. An example would be TOAD.

Automated teller machine This might include customer authentication, checking account status, acceptance of deposits, and dispensing cash.

A strategy game At a minimum, this would consist of a user interface, a data source, and a logic engine implementing the game rules.

1.2 Deliverable

Your deliverable will be a five paragraph (or more) paper of three or four proposed projects. Your description of each proposed project should contain enough information so that readers can understand the nature of the project. You may also give examples of similar, existing implementations.

The format of the paper will eventually use *Markdown*. For now, please write your paper either in Markdown or in plain ASCII text with no formatting, perhaps using Notepad++, Microsoft Notepad or a similar application. In general, your papers should consist of a short introduction, a short conclusion, and a body containing the substance of your discussion. For this paper, structure it like this:

1. A first paragraph or two as the introduction introducing yourself and stating your personal objectives for your project.
2. The body of your paper as follows:
   (a) A paragraph or two describing your first reviewed project, and how that project will help you accomplish your personal objectives.
   (b) A paragraph or two describing your second reviewed project, and how that project will help you accomplish your personal objectives.
   (c) A paragraph or two describing your third reviewed project, and how that project will help you accomplish your personal objectives.
   (d) (optional) A paragraph or two describing your fourth reviewed project, and how that project will help you accomplish your personal objectives.
3. A final paragraph or two as the conclusion reflecting on how you will achieve your personal objectives by building the project.

Your paper should have a title, author, and date. The subdivisions of your paper should have appropriate section and subsection headings.

1.3 Markdown

Markdown is a very simple, human readable, formatting system. A Markdown document is a simple text document, with no binary codes or special formatting instructions. It contains only printable characters, such as alphabetical characters, punctuation, and digits. The file extension is `.md`. This is half of all the Markdown you will need to know. You can pick up the other half just as easily.

- Headings
- Paragraphs
- Source listings
- Itemized lists
- Enumerated lists
- Rules
- Emphasis
- Hyperlinks
- Images
- HTML elements

Chapter 2

Project Selection

2.1 Project selection

Students will select one project from their list to focus on for their project They should make a formal investigation of the requirements of the project. If possible, they should identify similar applications that have been written. The deliverable is a six to eight paragraph project proposal. The project proposal should contain a sufficient description of the project that would allow construction of the project to begin.

2.2 Deliverable

You have two deliverabes for this step, an oral presentation and a written paper. Your oral persentation should be no longer than five minutes, and should cover the same topics as the written paper. It should be short and succienct, but it should cover all topics adequately. You may use a slide presentation if desired.

Your written deliverable will be a six to eight paragraph discussion of your proposed project. Your proposal should begin with a short introduction and conclude with a short conclusion. It should include discussions of the purpose of the software, an overall description of the high-level functional requirements of the software (that is, what the software will actually do), a survey of the relevant literature available that will assist you in completing your project, a short discussion of similar software, and a brief discussion of your project plan. Your writeup is not limited to two pages, and can include other sections that you feel will help the reader in understanding your proposal.

The format of the paper will eventually use *Markdown*. For now, please write your paper either in Markdown or in plain ASCII text with no formatting, perhaps using Microsoft Notepad or a similar application. In general, your papers should consist of a short introduction, a short conclusion, and a body containing the substance of your discussion. For this paper, structure it like this:

1. A first paragraph or two briefly introducing yourself and stating how your chosen project will accomplish your personal objectives.
2. A paragraph or two describing the data phase of your project. This should include a general description of the kinds of information your database will contain.
3. A paragraph or two describing the programming phase of your project. This should include a general description of the processing necessary to accomplish your objectives.
4. A paragraph or two describing the interface your project will present to the users. Focus on the functionality, not things such as layout, color, font, and so on.
5. A final paragraph or two reflecting on how you will actually go about building your project. This does not need to be in detail, but a general plan of actiob.

2.3 Distributed Version Control

The topic for this week will be version control using the Github Distributed remote repository. The following will be covered.

- The rationale for version control in software development
- Installation of the Git SCM software
- The command `git config` for the user name, user email, and core editor
- The command `git init`
- The command `git add` with the appropriate arguments
- The command `git status`
- The command `git commit` with the appropriate arguments
- The command `git log`
- The rationale for distributed version control in software development
- Signing up for a Github account
- Creating your first remote repository
- The command `git remote add` with the appropriate arguments
- The command `git remote -v`
- The command `git push` with the appropriate arguments
- a `.gitignore` file
- a `README.md` file

Chapter 3

Project Presentation I

To Do: Add activity diagram. Chapter 5 has the lab to convert a decimal integer into an octal integer.

3.1 Deliverables

This project step consists of your first presentation. In technology, you will be often called on to deliver a presentation. It can be as simple as describing a piece of code to yur colleagues, or as stressful as demonstrating a project before the CEO and the Board of Directors. You should be comfortable speaking to an audience.

Revised paper You should review and revise your paper from Step 2. Read your paper with a critical eye and rewrite it as a improved version.

Oral presentation Your presentation should be between five and six minutes. I recommend *writing* your presentation out and *rehearsing* it several times. The objective is not to read your presentation or memorize it, but to know it so well that you can speak about your topic using only a brief outline.

Slide deck (optional) If desired, you can create a slide deck in aid of your presentation. Again, you shouldn't read your slides but use them to help your audience understand your topic. Avoid "death by PowerPoint" at all costs.

3.2 Software Life Cycle

A brief introduction to the software life cycle:

- Inception
- Elaboration
- Construction
- Maintenance
- End of life

3.3 Software Development Cycle

A brief introduction to software development workflows:

- Requirements
- Analysis
- Design
- Implementation
- Testing

3.4 Software Process Models

A brief introduction to software process models:

- Incremental development
- Iterative development
- Spiral models
- Waterfall
- Rational Unified Process
- Agile processes

Part II

Database Development

Chapter 4

Database Requirements and Conceptual Design

4.1 Database Conceptual Design

Typically, a database project consists of five or six phases: requirements engineering, conceptual design (analysis), logical design, physical design, implementation, and testing. For the CAD project, we will only cover requirements, conceptual design, logical design, physical design, and implementation. We will not cover requirements engineering generally nor testing. We do not cover the requirements engineerinng phase because, presumably, since you are your own client/customer, you will already know your requirements. We do cover testing, but not in the database phase of the project. During the later phases of the project, you will be able to test your database design and revise it as necessary.

Generally, the conceptual design phase of a database project requires developers to break down the problem domain into relevant objects, called entities, and identify the relationship between objects. For example, a school might have courses, teachers, and students: teachers *teach* courses, students *enroll in* courses, and teachers *grade* students. The entities are TEACHER, STUDENT, and COURSE. The relationships are TEACH, ENROLL IN, and GRADE.

For this week's assignment, you will prepare a conceptual design of your project database and create an *entity-relationship diagram*. This process requires you to identify the entities that your project deals with and the relationship between the entities. This process also requires you to identify important attributes of your entities, and (if necessary) of your relationships. Examples of important attributes for STUDENT would be ID and NAME, and for COURSE would be CRN and TITLE.

We will also discuss the Dia Diagram Editor, `https://sourceforge.net/projects/dia-installer/`.

4.2 Deliverable

Your deliverable is an Entity Relationship Diagram (ERD). Your ERD is an image, and you should prepare it in some graphical format. The preferred format is PDF, but other image formats are acceptable, such as SVG, JPEG, GIF, PNG, EPS, etc.

4.3 Entities, Relationships, and Attributes

The following topics will be discussed:

- What is *conceptual database design*
- What is an *entity*
- What is a *weak entity*

- What is an *entity attribute*
- What is a *relationship*
- What is a *relationship type*
- What is a *relationship degree*
- What is a *relationship attribute*
- What is a *candidate key*
- What is a *super key*

Chapter 5

Logical and Physical Design

5.1 Database Logical Design

Typically, a database project consists of five or six phases: requirements engineering, conceptual design (analysis), logical design, physical design, implementation, and testing. For the CAD project, we will only cover requirements, conceptual design, logical design, physical design, and implementation. We will not cover requirements engineering nor testing. We do not cover the requirements phase because, presumably, since you are your own client/customer, you will already know your requirements. We do cover testing, but not in the database phase of the project. During the later phases of the project, you will be able to test your database design and revise it as necessary.

Generally, the logical design phase of a database project requires developers to decompose the entities identified by the conceptual design and create a logical schema. This means that the entities become tables (relations), and the attributes are expanded and identified as fields. During this process, the database is normalized and integirity constraints are realized. Logical design is much more of an art than a science, and is notoriously difficult.

For this week's assignment, you will prepare a logical design of your project database and create a *database diagram*. This process requires you to identify the relations (tables), that you normalize your database, and that you specify all fields (attributes) of your tables.

5.2 Deliverable

Your deliverable is an Database Diagram. Your database diagram is an image, and you should prepare it in some graphical format. The preferred format is PDF, but other image formats are acceptable, such as SVG, JPEG, GIF, PNG, EPS, etc.

5.3 Normalization and integrity constraints

The following topics will be discussed:

- What is *first normal form*
- What is *second normal form*
- What is *third normal form*
- What is *entity integrity*
- What is *domain integrity*
- What is *referential integrity*
- What is *logical (business) integrity*

5.4 Thought Exercise

You are given an assignment to build a database that models student enrollments in courses. In your conceptual design phase, you identify two entities, students and courses, and one relationship, `[Students] Enroll in [Courses]`. In your first attempt, you see immediately that your database is not in first normal form in that multiple students enroll in the same course and multiple students enroll in the same course. It's not in second normal form because student attributes do not depend on the course primary key, and course attributes depend do not on the student primary key. You see that it also (probably) is not in third normal form because some non-key attributes uniquely identify other non-key attributes.

Complete a logical design of this problem. We will discuss this in class. In my solution to this exercise, I designed five different tables. You may or may not have the same number of tables in your solution.

Chapter 6

Database Implementation, Presentation II

6.1 Database Physical Design

Typically, a database project consists of five or six phases: requirements engineering, conceptual design (analysis), logical design, physical design, implementation, and testing. For the CAD project, we will only cover conceptual design, logical design, physical design, and implementation. We will not cover requirements engineering nor testing. We do not cover the requirements phase because, presumably, since you are your own client/customer, you will already know your requirements. We do cover testing, but not in the database phase of the project. During the later phases of the project, you will be able to test your database design and revise it as necessary.

Physical design targets a particular relational database management system, such as SQL Server or SQLite. Logical designs target an abstract database, but physical designs must be specific to the particular database system. All elements must be specified, such as the constraints we will discuss. You should be able to hand your physical design to a SQL programmer, and he should be able to write the implementation code without reference to anything other than your physical design.

For this week's assignment, you will prepare a physical design of your project database and create a *database diagram*. This process requires you to create tables, create columns, specify

6.2 Deliverable

Your deliverable is a Database Diagram. Your database diagram is an image, and you should prepare it in some graphical format. The preferred format is PDF, but other image formats are acceptable, such as SVG, JPEG, GIF, PNG, EPS, etc.

6.3 Database objects and constraints

The following topics will be discussed:

- What is are *column level constraints* and *table level constraints*
- What is a *primary key*
- What is a *foreigh key*
- What are *insert, delete, and update anomolies*
- What is a *nullability constraint*

- What is a *uniqueness constraint*
- What is a *datatype*
- What is an *index*
- What is an *enumeration*
- What is a *check constraint*

Part III

Software Development Process

Chapter 7

Requirements Gathering

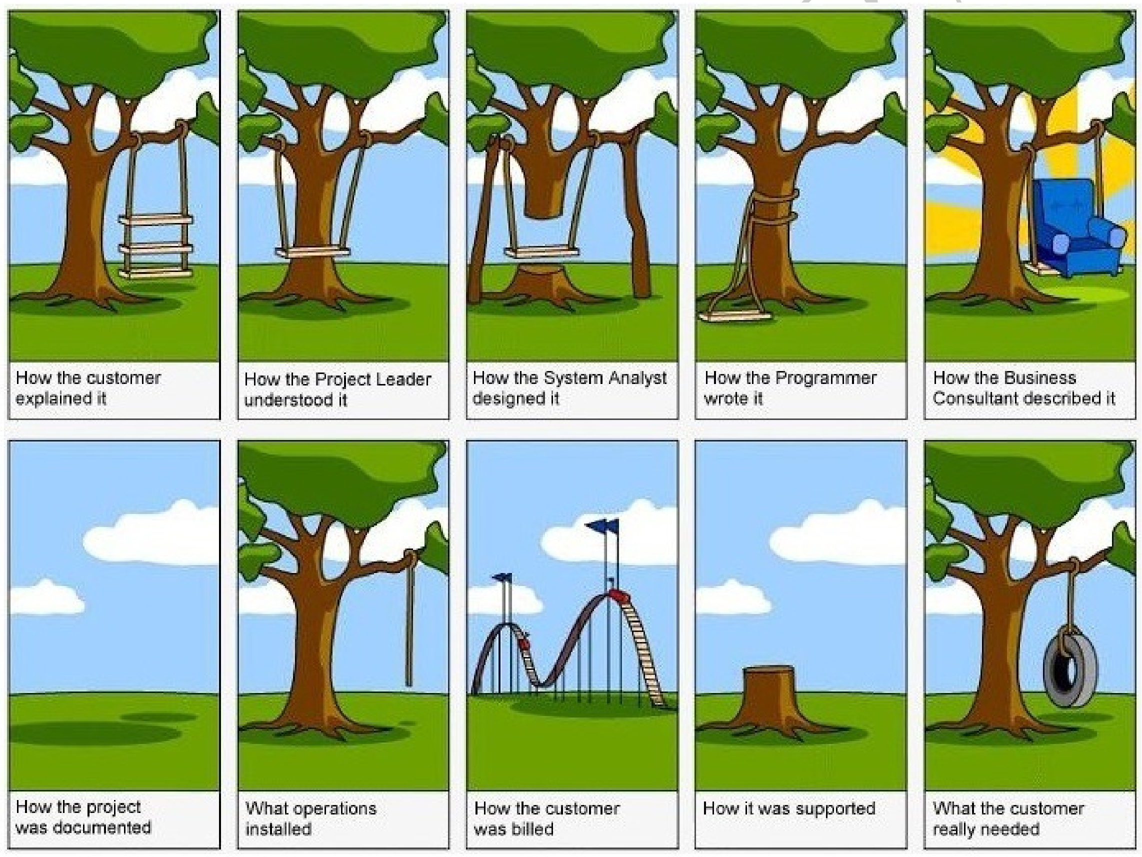

How the customer explained it
How the Project Leader understood it
How the System Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Figure 7.1: Different views of requirements

With over 70% of project failures being attributed to requirements gathering, why are we still using the same techniques and expecting different results? Requirements need to be discovered before they can be gathered and this requires a robust approach to analyzing the business needs.[1]

[1]Stieglitz, C. (2012). Beginning at the endrequirements gathering lessons from a flowchart junkie. Paper presented at PMI Global Congress 2012North America, Vancouver, British Columbia, Canada. Newtown Square, PA: Project Management Institute.

7.1 Requirements Phase

The Cloud Application Development Project consists of four parts, an inception part, the database part, the prgram development (logic) part, and the interface part. All these parts can be considered integral to software development, but they are approached differently. Last week, we concluded the database portion of the project. This week, we begin five week looking specifically at programming development. We can call this *software emngineering* in a narrow sense.

The discipline of software engineering includes various workflows in building software applications. These workflows are the same as engineering workflows in other engineering disclipines, such as civil and automotive engineering. These workflows consists of the following:

- Requirements Gathering
- Requirements Analysis
- Program Design
- Program Implementation
- Application Testing

Software engineers use many different processes, among them waterfall, the Rational Unified Process, and various agile processes, including Scrum and Kanban. Different processes order and emphasize the workflows differently, but despite their differences, all processes use the same workflows. The process we will use in this project is a modified version of Extreme Programming (XP). If you are interested in XP, you can read about it offline.

7.2 Requirements Gathering

Requirements gathering is probably the most important activity to be performed in delivering an information solution. There is no one perfect means for identifying and gathering requirements. The most appropriate methods will vary from project to project. Some commonly used methods include:[2]

- Interviews
- Storyboarding
- Use cases
- Questionaires
- Brainstorming
- Prototyping

Use cases can be especially valuable since they provide specific scenarios of how the solution is intended to be used and by whom. These use cases can then provide a direct basis for testing the delivered solution. Following are some things to keep in mind when gathering requirements:

- Identify and involve a representative set of stakeholders (don't lose sight of all of the players)
- Seek breadth before depth (get the big picture before deep diving)
- Iterate and clarify (as more requirements surface they will evolve)
- Prioritize (separate the must-haves from the nice-to-haves)
- Use the stakeholder's terminology (you're doing an information solution not a technical solution)

[2]https://its.unl.edu/bestpractices/requirements-gathering

- Employ KISS (keep it simple but be thorough)
- Realize that you will never get a complete set of requirements up-front; some won't surface until the stakeholders have pieces of the solution that they can see and touch (be careful of scope creep)
- Remember goals and objectives are not requirements (they are just as important though)
- Requirements and constraints should drive the solution not the other way around

7.3 What is Requirements Elicitation?

Requirements elicitation (also known as Requirements Gathering or Capture) is the process of generating a list of requirements (functional, system, technical, etc.) from the various stakeholders (customers, users, vendors, IT staff, etc.) that will be used as the basis for the formal Requirements Definition.

The process is not as straightforward as just asking the stakeholders what they want they system to do, as in many cases, they are not aware of all the possibilities that exist, and may be limited by their immersion in the current state. For example asking people in the 19th Century for their requirements for a self-propelled vehicle, would have just resulted in the specification for a faster horse-drawn carriage rather than an automobile. Beware the old adage, "it's everything I asked for, but not what I need"![3]

7.3.1 What Techniques Can Be Used?

Interviews - These are an invaluable tool at the beginning of the process for getting background information on the business problems and understanding a current-world perspective of what the system being proposed needs to do. You need to make sure that your interviews cover a diverse cross-section of different stakeholders, so that the requirements are not skewed towards one particular function or area.

Questionnaires - One of the challenges with interviews is that you will only get the information that the person is consciously aware of. Sometimes there are latent requirements and features that are better obtained through questionnaires. By using carefully chosen, probing questions (based on the information captured in prior interviews), you can drill-down on specific areas that the stakeholders don't know are important, but can be critical to the eventual design of the system.

User Observation - One of the best ways to determine the features of a system, that does not result in "paving the cowpath" (i.e. building a slightly improved version of the current state) is to observe users actually performing their daily tasks, and ideally recording the actions and activities that take place. By understanding the holistic context of how they perform the tasks, you can write requirements that will reinvent the processes rather than just automating them, and will ensure that usability is paramount.

Workshops - Once you have the broad set of potential requirements defined, you will need to reconcile divergent opinions and contrasting views to ensure that the system will meet the needs of all users and not just the most vocal group. Workshops are a crucial tool that can be used to validate the initial requirements, generate additional detail, gain consensus and capture the constraining assumptions.

Brainstorming - This is a powerful activity, which can be performed either in the context of a workshow or on its own. By considering different parts of the system and considering "what-if" scenarios, or "blue-sky" ideas, you can break out of the context of the current-state and consider visionary ideas for the future. Tools such as whiteboards or mind-mapping software can be very helpful in this phase.

Role Playing - In situations where the requirements depend heavily on different types of user, formal role-playing (where different people take on the roles of different users in the system/process) can be a good way of understanding how the different parts of the system need to work to support the integrated processes (e.g in an ERP system).

[3]https://www.inflectra.com/ideas/topic/requirements-gathering.aspx

Use Cases and Scenarios - Once you have the high-level functional requirements defined, it is useful to develop different use-cases and scenarios that can be used to validate the functionality in different situations, and to discover any special exception or boundary cases that need to be considered.

Prototyping - There is truth to the saying "I don't know what I want, but I know that I don't want that!". Often stakeholders won't have a clear idea about what the requirements are, but if you put together several different prototypes of what the future could be, they will know which parts they like. You can then synthesize the different favored parts of the prototypes to reverse-engineer the requirements.

7.3.2 How Should the Information Be Captured?

There are many different ways to capture the information, from a simple Word document, spreadsheet or presentation to sophisticated modelling diagrams. We recommend that the initial high-level brainstorming and requirements discovery be done on a whiteboard to foster collaboration. Once the initial ideas have crystallized, we recommend using a formal Requirements Management System to record the information from the whiteboard and drill-down the functional requirements in smaller focus-groups to arrive at the use-cases and system requirements.

7.3.3 What Pitfalls Exists?

The biggest risk is that by asking existing users or stakeholders to help define the requirements, you will get a requirements specification that is unduly influenced by the current ways of doing business. Therefore we recommend that you ensure that sufficient third-party research into industry-wide trends and usability research (e.g. observation) is done to ensure that the requirements take into account future opportunities as well as current problems.

7.4 Deliverable

Your deliverable will be a document detailing several use case *stories* for your project. Your document will be written using markdown and posted to your Github account. You may also submit use case *diagrams* but these are optional.

For further information on use cases, see the following:

- `https://www.usability.gov/how-to-and-tools/methods/use-cases.html`
- `http://www.gatherspace.com/static/use_case_example.html`
- `https://www.uml-diagrams.org/use-case-diagrams-examples.html`
- `http://tynerblain.com/blog/2007/04/09/sample-use-case-example/`

Chapter 8

Requirements Analysis and Specification

8.1 Software Development Cycle

A brief review to software development workflows:

- Requirements
  - requirements engineering
  - collaboration with stakeholders
  - use cases (user stories)
- Analysis
  - functional requirements
  - non-functional requirements
  - Software Requirements Specification (SRS)
- Design
  - Unified Modeling Language (UML)
  - static design diagrams
  - dynamic design diagrams
- Implementation
- Testing

8.2 Requirements Analysis

The discipline of software engineering includes various workflows in building software applications. These workflows are the same as engineering workflows in other engineering disclipines, such as civil and automotive engineering. These workflows consists of the following:

- Requirements Gathering
- Requirements Analysis
- Program Design

- Program Implementation
- Application Testing

Software engineers use many different processes, among them waterfall, the Rational Unified Process, and various agile processes, including Scrum and Kanban. Different processes order and emphasize the workflows differently, but despite their differences, all processes use the same workflows. Ideally, in this course we would use an iterative, incremental process, but unfortunately, 18 weeks is too short to do that. The process we will use in this project is a waterfall process. That is, we start ot the "top" of the waterfall with requirements , and "fall" down through analysis, design, implementation, and testing.

In the last step, we looked at the *requirements* workflow. In gathering requirements, developers try to understand the tasks that the application is to model. Developers engage in many different kinds of activities, such as reviewing paper files and other business documents, interviews with employees and other stakeholders, observation, and so on. The object of the requirements phase is to discover what it is that the software should actually do. The requirements workflow is incredibly important. After all, how can you build an application if you do not know what it will do? In fact, the largest number of software defects are failures in requirements. We build error-prone software because we fail to understand what it's supposed to do.

In ths step, we will do the *analysis* workflow. The object of analysis is to develop a *software requirements specification* (SRS) from the requirements. A SRS is a formal written document that derives directly from the requirements and addressed to software designers. Generally, requirements can be seen as *functional* and *non-functional* requirements. Functional requirements consist of discrete objectives stating exactly what the software will do. Non-functional requirements consist of other requirements, such as machine capability, availability and uptime, GUI requirements, and other requirements that are necessary for the software to perform but do not describe the functions that the software should perform. In this project, we will only consider fumctional requirements.

We will base our discussion on IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, June 25, 1998. This document has been superseded, but purchase is required for the current version. You can find copies of Std 830-1998 freely available. From the introduction of Std 830-1998:

> *This recommended practice describes recommended approaches for the specification of software requirements. It is based on a model in which the result of the software requirements specication process is an unambiguous and complete specification document. It should help*
>
> 1. *Software customers to accurately describe what they wish to obtain;*
> 2. *Software suppliers to understand exactly what the customer wants;*
> 3. *Individuals to accomplish the following goals:*
>   (a) *Develop a standard software requirements specification (SRS) outline for their own organizations;*
>   (b) *Define the format and content of their specific software requirements specifications;*
>   (c) *Develop additional local supporting items such as an SRS quality checklist, or an SRS writer's handbook.*

We will focus our study on sections 4.3, *Characteristics of a good SRS*, and 5.3, *Specific requirements (Section 3 of the SRS)*. Section 4.3 sets forth the charictics of a good SRS. An SRS should be

a) Correct;

b) Unambiguous;

c) Complete;

d) Consistent;

e) Ranked for importance and/or stability;

f) Veriable;

g) Modiable;

h) Traceable.

Section 5.3 sets forth the specific requirements of an SRS, covering:

a) Specific requirements should be stated in conformance with all the characteristics described in 4.3.

b) Specific requirements should be cross-referenced to earlier documents that relate.

c) All requirements should be uniquely identiable.

d) Careful attention should be given to organizing the requirements to maximize readability

Specific requirements include external interfaces, functions, performance requirements, logical database requirements, design constraints, standards compliance, software system attributes, reliability, availability, security, maintainability, portability, organization, modes, user classes, objects, features, stimuli, response, and hierarchy. For this class, we will focus primarily on functions, user classes, and objects.

8.3 Deliverable

You deliverable is a formal SRS, written using Markdown formatting. You should upload your SRS to your project account in Github.

8.4 Object Oriented Development Principles

A brief review to software development principles for OOAD:

- Single responsibility principle (SRP): This principle states that software component (function, class or module) should focus on one unique tasks (have only one responsibility).
- Open/closed principle (OCP): This principle states that software entities should be designed with the application growth (new code) in mind (be open to extension), but the application growth should require the smaller amount of changes to the existing code as possible (be closed for modification).
- Liskov substitution principle (LSP): This principle states that we should be able to replace a class in a program with another class as long as both classes implement the same interface. After replacing the class no other changes should be required and the program should continue to work as it did originally.
- Interface segregation principle (ISP): This principle states that we should split interfaces which are very large (general-purpose interfaces) into smaller and more specific ones (many client-specific interfaces) so that clients will only have to know about the methods that are of interest to them.
- Dependency inversion principle (DIP): This principle states that entities should depend on abstractions (interfaces) as opposed to depend on concretion (classes).

Chapter 9

Application Design

9.1 Program Design

The discipline of software engineering includes various workflows in building software applications. These workflows are the same as engineering workflows in other engineering disclipines, such as civil and automotive engineering. These workflows consists of the following:

- Requirements Gathering
- Requirements Analysis
- Program Design
- Program Implementation
- Application Testing

For this deliverable, we will focus on software design. The design of software is *by far* the most difficult phase of software development. It's equal parts both science and art. This can be seen in the development of various programming paradigms since the advent of stored program computers. Programming paradigms include procedural (imperative) programming, functional programming, object oriented programming, event driven programming, declarative programming, and many others. In this class, we will focus only on object oriented programming.

The object of software design is to develop a series of design documents, software design descriptions (Software Design Descriptions) from the software requirements specification (SRS). Design documents can take various forms — for our purposes we will use graphics as design documents. We will will focus on the Unified Modeling Language (UML) documents, with one exception. We will base our discussion on IEEE Std 1016-2009, *IEEE Standard for Information Technology — Systems Design — Software Design Descriptions*, July 20, 2009. This document has been superseded, but purchase is required for the current version. You can find copies of Std 1016-2009 freely available. From the introduction:

> *SDDs play a pivotal role in the development and maintenance of software systems. During its lifetime, an SDD is used by acquirers, project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers. Each of these stakeholders has unique needs, both in terms of required design information and optimal organization of that information. Hence, a design description contains the design information needed by those stakeholders. . . . The standard specifies that an SDD be organized into a number of design views. Each view addresses a specific set of design concerns of the stakeholders. Each design view is prescribed by a design viewpoint. A viewpoint identifies the design concerns to be focused upon within its view and selects the design languages used to record that design view. The standard establishes a common set of viewpoints for design views, as a starting point for the preparation of an SDD, and a generic capability for defining new design viewpoints thereby expanding the expressiveness of an SDD for its stakeholders.*

We will focus our study on section 5, *Design Viewpoints.*

1. Introduction
2. Context viewpoint
3. Composition viewpoint
4. Logical viewpoint
5. Dependency viewpoint
6. Information viewpoint
7. Patterns use viewpoint
8. Interface viewpoint
9. Structure viewpoint
10. Interaction viewpoint
11. State dynamics viewpoint
12. Algorithm viewpoint
13. Resource viewpoint

9.2 Project design

In this project, we have several predetermined design constraints. The most important is that of *object oriented anaysis and design.* We are using a object-oriented language, C#, and this fact alone dictates that we use object-oriented techniques and principles in building the application. Almost as important is that of the *model-view-controller* design pattern. The use of design patterns is essential for object oriented technology.[1]

In your design, you will have three kinds of classes: Model classes, which represent the data used by ytour application, View classes, which represent user iteraction, and Controller classes, which represent the program logic and business rules. To simplify building the project, we will start by building a very simple console application. As we progress, you will use ASP.NET to build an application with a graphical usr interface. For an example of a simple console application implementing MVC, please see appendix **??**.

9.3 Discussion Topics

Theoretically, one team composes the SRS and hands it off to the design team. The design team (in theory) works *only* from the SRS and creates a series of design documents. The design team hands the design documents off to the implementation, which implements the software working *only* from the design documents. The design team thus has a double responsibility: it must decompose the software requirements into a set of drawings (and other documents) that completely express all the functional requirements, and it must prepare the SDD so that it may be implemented completely and unambiguously. Needless to say, this is a very difficult task.

We will discuss three of the most used programming paradigms in use today, procedural (imperative) programming, functional programming, and object oriented programming. We will look at the different ways they express delivering instructions to the computer, their advantages, and their differences.

We will discuss four kinds of UML diagrams: (1) class diagrams, (2) state diagrams, (3) system sequence diagrams, and (4) component diagrams. We will also discuss data flow diagrams (DFD), which are not part of UML.

[1]The "bible" of design patterns is *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, John Vlissides, Ralph Johnson. At some point in your career as a developer, you will read this book.

9.4 Deliverables

Your deliverable is a set of images in PDF format, including a class diagram, a state diagram, and a component diagram.

Chapter 10

Implementation and Testing

10.1 Requirements Analysis

The discipline of software engineering includes various workflows in building software applications. These workflows are the same as engineering workflows in other engineering disclipines, such as civil and automotive engineering. These workflows consists of the following:

- Requirements Gathering
- Requirements Analysis
- Program Design
- Program Implementation
- Application Testing

Software engineers use many different processes, among them waterfall, the Rational Unified Process, and various agile processes, including Scrum and Kanban. Different processes order and emphasize the workflows differently, but despite their differences, all processes use the same workflows. Ideally, in this course we would use an iterative, incremental process, but unfortunately, 18 weeks is too short to do that. The process we will use in this project is a waterfall process. That is, we start ot the "top" of the waterfall with requirements , and "fall" down through analysis, design, implementation, and testing.

In the last step, we looked at the *requirements* workflow. In gathering requirements, developers try to understand the tasks that the application is to model. Developers engage in many different kinds of activities, such as reviewing paper files and other business documents, interviews with employees and other stakeholders, observation, and so on. The object of the requirements phase is to discover what it is that the software should actually do. The requirements workflow is incredibly important. After all, how can you build an application if you do not know what it will do? In fact, the largest number of software defects are failures in requirements. We build error-prone software because we fail to understand what it's supposed to do.

In ths step, we will do the *analysis* workflow. The object of analysis is to develop a *software requirements specification* (SRS) from the requirements. A SRS is a formal written document that derives directly from the requirements and addressed to software designers. Generally, requirements can be seen as *functional* and *non-functional* requirements. Functional requirements consist of discrete objectives stating exactly what the software will do. Non-functional requirements consist of other requirements, such as machine capability, availability and uptime, GUI requirements, and other requirements that are necessary for the software to perform but do not describe the functions that the software should perform. In this project, we will only consider fumctional requirements.

We will base our discussion on IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, June 25, 1998. This document has been superseded, but purchase is required for the current version. You can find copies of Std 830-1998 freely available. From the introduction of Std 830-1998:

> *This recommended practice describes recommended approaches for the specification of software requirements. It is based on a model in which the result of the software requirements specication process is an unambiguous and complete specification document. It should help*
>
> 1. *Software customers to accurately describe what they wish to obtain;*
> 2. *Software suppliers to understand exactly what the customer wants;*
> 3. *Individuals to accomplish the following goals:*
>   (a) *Develop a standard software requirements specification (SRS) outline for their own organizations;*
>   (b) *Define the format and content of their specific software requirements specifications;*
>   (c) *Develop additional local supporting items such as an SRS quality checklist, or an SRS writer's handbook.*

We will focus our study on sections 4.3, *Characteristics of a good SRS*, and 5.3, *Specific requirements (Section 3 of the SRS)*. Section 4.3 sets forth the charictics of a good SRS. An SRS should be

a) Correct;

b) Unambiguous;

c) Complete;

d) Consistent;

e) Ranked for importance and/or stability;

f) Veriable;

g) Modiable;

h) Traceable.

Section 5.3 sets forth the specific requirements of an SRS, covering:

a) Specific requirements should be stated in conformance with all the characteristics described in 4.3.

b) Specific requirements should be cross-referenced to earlier documents that relate.

c) All requirements should be uniquely identiable.

d) Careful attention should be given to organizing the requirements to maximize readability

Specific requirements include external interfaces, functions, performance requirements, logical database requirements, design constraints, standards compliance, software system attributes, reliability, availability, security, maintainability, portability, organization, modes, user classes, objects, features, stimuli, response, and hierarchy. For this class, we will focus primarily on functions, user classes, and objects.

10.2 Deliverable

You deliverable is a formal SRS, written using Markdown formatting. You should upload your SRS to your project account in Github.

Chapter 11

Project Presentation III

11.1 Program Design

The discipline of software engineering includes various workflows in building software applications. These workflows are the same as engineering workflows in other engineering disclipines, such as civil and automotive engineering. These workflows consists of the following:

- Requirements Gathering
- Requirements Analysis
- Program Design
- Program Implementation
- Application Testing

For this deliverable, we will focus on software design. The design of software is *by far* the most difficult phase of software development. It's equal parts both science and art. This can be seen in the development of various programming paradigms since the advent of stored program computers. Programming paradigms include procedural (imperative) programming, functional programming, object oriented programming, event driven programming, declarative programming, and many others. In this class, we will focus only on object oriented programming.

The object of software design is to develop a series of design documents, software design descriptions (Software Design Descriptions) from the software requirements specification (SRS). Design documents can take various forms — for our purposes we will use graphics as design documents. We will will focus on the Unified Modeling Language (UML) documents, with one exception. We will base our discussion on IEEE Std 1016-2009, *IEEE Standard for Information Technology — Systems Design — Software Design Descriptions*, July 20, 2009. This document has been superseded, but purchase is required for the current version. You can find copies of Std 1016-2009 freely available. From the introduction:

> *SDDs play a pivotal role in the development and maintenance of software systems. During its lifetime, an SDD is used by acquirers, project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers. Each of these stakeholders has unique needs, both in terms of required design information and optimal organization of that information. Hence, a design description contains the design information needed by those stakeholders. ... The standard specifies that an SDD be organized into a number of design views. Each view addresses a specific set of design concerns of the stakeholders. Each design view is prescribed by a design viewpoint. A viewpoint identifies the design concerns to be focused upon within its view and selects the design languages used to record that design view. The standard establishes a common set of viewpoints for design views, as a starting point for the preparation of an SDD, and a generic capability for defining new design viewpoints thereby expanding the expressiveness of an SDD for its stakeholders.*

We will focus our study on section 5, *Design Viewpoints.*

1. Introduction
2. Context viewpoint
3. Composition viewpoint
4. Logical viewpoint
5. Dependency viewpoint
6. Information viewpoint
7. Patterns use viewpoint
8. Interface viewpoint
9. Structure viewpoint
10. Interaction viewpoint
11. State dynamics viewpoint
12. Algorithm viewpoint
13. Resource viewpoint

11.2 Project design

In this project, we have several predetermined design constraints. The most important is that of *object oriented anaysis and design.* We are using a object-oriented language, C#, and this fact alone dictates that we use object-oriented techniques and principles in building the application. Almost as important is that of the *model-view-controller* design pattern. The use of design patterns is essential for object oriented technology.[1]

In your design, you will have three kinds of classes: Model classes, which represent the data used by ytour application, View classes, which represent user iteraction, and Controller classes, which represent the program logic and business rules. To simplify building the project, we will start by building a very simple console application. As we progress, you will use ASP.NET to build an application with a graphical usr interface. For an example of a simple console application implementing MVC, please see appendix **??**.

11.3 Discussion Topics

Theoretically, one team composes the SRS and hands it off to the design team. The design team (in theory) works *only* from the SRS and creates a series of design documents. The design team hands the design documents off to the implementation, which implements the software working *only* from the design documents. The design team thus has a double responsibility: it must decompose the software requirements into a set of drawings (and other documents) that completely express all the functional requirements, and it must prepare the SDD so that it may be implemented completely and unambiguously. Needless to say, this is a very difficult task.

We will discuss three of the most used programming paradigms in use today, procedural (imperative) programming, functional programming, and object oriented programming. We will look at the different ways they express delivering instructions to the computer, their advantages, and their differences.

We will discuss four kinds of UML diagrams: (1) class diagrams, (2) state diagrams, (3) system sequence diagrams, and (4) component diagrams. We will also discuss data flow diagrams (DFD), which are not part of UML.

[1]The "bible" of design patterns is *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, John Vlissides, Ralph Johnson. At some point in your career as a developer, you will read this book.

11.4 Deliverables

You deliverable is a set of images in PDF format, including a class diagram, a state diagram, and a component diagram.

Part IV

Application Development

Chapter 12

Iteration 1 RAD, Single Responsibility

12.1 Program Implementation

Working from your design, you will implement a significant subset of your project. Your implementation should focus primarily on the data component and the processing component. We will focus on user interfaces during the last phase of the project.

In order to simplify the project, we will not address a graphical user interface in this phase. You application must be built as a **console** application. In the next phase, we will adapt your project to a graphical user interface using the MVC design pattern.

It is commonly said that a graphical user interface contributes 80 percent of the complexity to an application, but only 20 percent of the functionality. You will experience this as you begin to develop graphical applications. However, this course is not a user interface design course, but an application development course. As such, it's essential to focus obn functionality. Graphical user interfaces are important, but functionality is much more important.

12.2 Deliverables

Your assignment is to implement a subset of the requirements you have previously identified. You are not expected to implement the entire project. You deliverable is a set of C# class files.

Chapter 13

Iteration 1 Implementation and Testing, Open-Closed

13.1 Testing

For this deliverable, we will focus on application testing. We will survey the following topics

1. unit tests
2. integration tests
3. regression tests
4. acceptance tests

Perhaps the key concept is this: *Testing is not complete until it is automated.* We will consider primarily usit tests, how you should conduct unit tests, and how you should automate unit tests.

13.2 Principles of testing

Discussion text to be added.

13.3 Test driven development

Discussion text to be added.

13.4 Deliverables

You deliverable is a set of C# class files implementing a serieso of automated tests.

Chapter 14

Iteration 2 RAD, Liskov Substitution

14.1 User Interface Design Process

The design process for user interfaces is somewhat similar to the design process for applications. In both cases, developers must identify requirements, define a requirements specification, design the interface, implement the interface, and test the interface. However, the work of a design engineer depends a great deal on graphical design skills, and most developers of user interfaces have much more in common with graphical artists than software developers. This week, we will consider the process of user interface design. Next week we will con sider the principles of interface design.

The user interface design process may be broken down into the following phases. As with software development processes, developers have different development models. However, all models include these phases:

- requirements engineering
- requirements analysis
- information architecture design
- construction of wireframes
- construction of prototypes
- implmentation
- testing

For this project, we will combine the wireframe, a prototype, and mockup steps. Next week, we will implement the user interface. Presumably, since this is your project, you have a pretty concrete idea about the requirements of the interface and the information architecture.

A *graphical user interface* denotes a human-computer interface (HCI), whereby humans give commands to computers by means of a point-and-click mechanism rather than by typing commands into a command prompt. The interface works by translating the actions of the human into commands that the application recognizes. This is an important point that cannot be missed: *a graphical interface is only another means of issueing commands to a computer.* As such, it cannot do more that the API of the software provides, and typically provides far less.

What is a wireframe? A wireframe is a model containing the essential elements of a system, with all external characteristics as to style and color removed. The wireframe represents the *functionality* of a system. For example, a wireframe of a new model of an automobile is just, literally, a "wire frame." We can see the engine compartment, the passenger compartment, the cargo compartment, the engine, transmission, and steering components, and the shape and size of the car. We can't see the exterior features, design elements, color, etc. In the same way, a wireframe of a user interface contains just the basics, with nothing to distract the developers. The function of the wireframe is simply to nail down the kinds of controls the interface contains. Many times, wireframes of user interfaces are realized by images: drawings, photographs, or other pictorial representations.

What is a prototype? A prototype can be though of as a wireframe with actions. The controls "work" in some sense. Users can type text into textboxes, select items from a drop down list, select files from a file explorer, push radio buttons, click submit buttons, etc. The purpose of a prototype is the exploration of the kinds of actions a user makes with the interface. Prototypes are usually realized by code that compiles and runs, but does not interact with any system. As example might be an HTML file that can be opened in a browser, that the user can explore, but that does not actually do anything.

What is a mockup? A mockup can be thought of as a prototype with style and design. What are the colors used? What fonts wil be used? What font sizes will the textual elements be? How will the interface be laid out? What are the accessibility issues? Mockups are generally realized by image files.

14.2 User Interface Design Activities

Users, UX and context

- identification of ways of the products application
- identification of the general target audiences (TA) attributes
- identification of usability goals of the TA
- identification of users roles vs. goals, ranking of goals importance for users
- identification of functionality options necessary for meeting TA goals and objectives; ranking of functionality attributes dependant on how well they help to reach goals
- comparative analysis of functionality and content vs. competitors products
- consideration of business and functionality-related limitations
- choice of the optimal products interfaces enabling to reach the key business goals of the project

The final document: usability analysis outlining potential user capabilities of the product vs. The initial business and functional requirements and limitations

Navigation and structure

- designing scenarios outlining the user-product interaction in order to reach the goals (applicable to the chosen interface options and user roles)
- ranking of scenarios by importance, relying on usage frequency and users roles
- development of the information architecture and structure and navigation interface design providing optimal functionality, content and user interaction scenarios

The final document: the UI-structure outlining the pattern of products interface and the path the user follows while browsing the website (in accordance with the user scenarios and roles)

Layout design

- layout design of the structures pages, which are to feature on a screen. Such design meets requirements towards navigation, functional, graphic and text elements of the screen forms of the pages. The requirements in question, in their turn, meet the standards of usability checklists

The final document: the UI-design featuring a catalogue of the key screen interface forms and requirements to location, priority, form and content of information, graphical and functional elements

Visual interface design

- designing of creative visual elements of the interface to meet the brand-book standards and the corporate identity. This includes: style, color, fonts, graphic solutions (read more...)
- design of association icons and graphic symbols
- general design of the key screen forms of the key screen forms of the compositional design

The final document: the GUI-design outlining visual standards of information, graphic and functional interface elements

Preparing of prototype

- creation of interactive model of the product enabling to make a full-scale investigation and evaluate the products usability, since the prototype fully imitates the future product (read more...)

The final document: the products prototype, which fully reflects features and usability of the future product (in terms of the user interface)

User testing

- recruitment of respondents
- the research set-up (defining of the research hypothesis, design of user scenarios etc.)
- conduction of testing and recording of the results
- the results analysis
- provision of report with recommendations on the way to eliminate weak points and problem zones

  The final document: report on user testing

UI Specification

- preparation of the document User Interface Specification outlining standards of the structural, compositional and visual design of the product taking into account recommendations based on the usability testing results
- revision of the prototype relying on recommendations based on usability testing

The final document: Specification of the products UI

14.3 Deliverable – Prototype

You deliverable is a set of source files representing views of each functionality your application will provide to the user. These files may be C# files, `.cshtml` files, or plain HTML files.

Chapter 15

Iteration 2 Implementation and Testing, Interface Segregation

15.1 User Interface Design Process

The design process for user interfaces is somewhat similar to the design process for applications. In both cases, developers must identify requirements, define a requirements specification, design the interface, implement the interface, and test the interface. However, the work of a design engineer depends a great deal on graphical design skills, and most developers of user interfaces have much more in common with graphical artists than software developers. This week, we will consider the process of user interface design. Next week we will con sider the principles of interface design.

The user interface design process may be broken down into the following phases. As with software development processes, developers have different development models. However, all models include these phases:

- requirements engineering
- requirements analysis
- information architecture design
- construction of wireframes
- construction of prototypes‘
- implmentation
- testing

For this project, we will only work through a wireframe, a prototype, a mockup, and the implementation phase. Presumably, since this is your project, you have a pretty concrete idea about the requirements of the interface and the information architecture.

A *graphical user interface* denotes a human-computer interface (HCI), whereby humans give commands to computers by means of a point-and-click mechanism rather than by typing commands into a command prompt. The interface works by translating the actions of the human into commands that the application recognizes. This is an important point that cannot be missed: *a graphical interface is only another means of issueing commands to a computer.* As such, it cannot do more that the API of the software provides, and typically provides far less.

What is a wireframe? A wireframe is a model containing the essential elements of a system, with all external characteristics as to style and color removed. The wireframe represents the *functionality* of a system. For example, a wireframe of a new model of an automobile is just, literally, a "wire frame." We can see the engine compartment, the passenger compartment, the cargo compartment, the engine, transmission, and steering components, and the shape and size of the car. We can't see the exterior features, design elements, color, etc. In the same way, a wireframe of a user interface contains just the basics, with nothing to distract the developers. The function of the wireframe is simply to nail down the kinds of controls the interface contains. Many times, wireframes of user interfaces are realized by images: drawings, photographs, or other pictorial representations.

What is a prototype? A prototype can be though of as a wireframe with actions. The controls "workj" in some sense. Users can type text into textboxes, select items from a drop down list, select files from a file explorer, push radio buttons, click submit buttons, etc. The purpose of a prototype is the exploration of the kinds of actions a user makes with the interface. Prototypes are usually realized by code that compiles and runs, but does not interact with any system. As example might be an HTML file that can be opened in a browser, that the user can explore, but that does not actually do anything.

What is a mockup? A mockup can be thought of as a prototype with style and design. What are the colors used? What fonts wil be used? What font sizes will the textual elements be? How will the interface be laid out? What are the accessibility issues? Mockups are generally realized by image files.

15.2 Deliverable – Wireframe

You deliverable is a set of image files representing a wireframe of each functionality your application will provide to the user. For example, if your application provides secure access, you might have (1) a splash page), (2) a new user registration page, (3) a user login page, and (4) a home page for logged in users. You may also include pages for recovery of usernames, the resetting of passwords, etc. If your application is an e-commerce application, your wireframe might depict a product listing (without product information, of course).

Chapter 16

Iteration 3 RAD, Dependency Inversion

16.1 User Interface Design Activities

Users, UX and context

- identification of ways of the products application
- identification of the general target audiences (TA) attributes
- identification of usability goals of the TA
- identification of users roles vs. goals, ranking of goals importance for users
- identification of functionality options necessary for meeting TA goals and objectives; ranking of functionality attributes dependant on how well they help to reach goals
- comparative analysis of functionality and content vs. competitors products
- consideration of business and functionality-related limitations
- choice of the optimal products interfaces enabling to reach the key business goals of the project

The final document: usability analysis outlining potential user capabilities of the product vs. The initial business and functional requirements and limitations

Navigation and structure

- designing scenarios outlining the user-product interaction in order to reach the goals (applicable to the chosen interface options and user roles)
- ranking of scenarios by importance, relying on usage frequency and users roles
- development of the information architecture and structure and navigation interface design providing optimal functionality, content and user interaction scenarios

The final document: the UI-structure outlining the pattern of products interface and the path the user follows while browsing the website (in accordance with the user scenarios and roles)

Layout design

- layout design of the structures pages, which are to feature on a screen. Such design meets requirements towards navigation, functional, graphic and text elements of the screen forms of the pages. The requirements in question, in their turn, meet the standards of usability checklists

The final document: the UI-design featuring a catalogue of the key screen interface forms and requirements to location, priority, form and content of information, graphical and functional elements

Visual interface design

- designing of creative visual elements of the interface to meet the brand-book standards and the corporate identity. This includes: style, color, fonts, graphic solutions (read more...)
- design of association icons and graphic symbols
- general design of the key screen forms of the key screen forms of the compositional design

The final document: the GUI-design outlining visual standards of information, graphic and functional interface elements

Preparing of prototype

- creation of interactive model of the product enabling to make a full-scale investigation and evaluate the products usability, since the prototype fully imitates the future product (read more...)

The final document: the products prototype, which fully reflects features and usability of the future product (in terms of the user interface)

User testing

- recruitment of respondents
- the research set-up (defining of the research hypothesis, design of user scenarios etc.)
- conduction of testing and recording of the results
- the results analysis
- provision of report with recommendations on the way to eliminate weak points and problem zones

  The final document: report on user testing

UI Specification

- preparation of the document User Interface Specification outlining standards of the structural, compositional and visual design of the product taking into account recommendations based on the usability testing results
- revision of the prototype relying on recommendations based on usability testing

The final document: Specification of the products UI

16.2 Deliverable – Prototype

You deliverable is a set of source files representing a prototype of each functionality your application will provide to the user. These files may be C# files, `.cshtml` files, XAML files, or plain HTML files.

Chapter 17

Iteration 3 Implementation and Testing, Code Security

17.1 Interface elements

Choose the controls according to the use intended. Generally, uses may be classified as (1) input, (2) navigation, (3) informational, and (4) containing.

Input Controls buttons, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, date field

Navigational Components breadcrumb, slider, search field, pagination, tags, icons

Informational Components tooltips, icons, progress bar, notifications, message boxes, modal windows

Containers accordion

17.2 User Interface Design Principles

Keep the interface simple The best interfaces are almost invisible to the user. They avoid unnecessary elements and are clear in the language they use on labels and in messaging.

Create consistency and use common UI elements By using common elements in your UI, users feel more comfortable and are able to get things done more quickly. It is also important to create patterns in language, layout and design throughout the site to help facilitate efficiency. Once a user learns how to do something, they should be able to transfer that skill to other parts of the site.

Be purposeful in page layout Consider the spatial relationships between items on the page and structure the page based on importance. Careful placement of items can help draw attention to the most important pieces of information and can aid scanning and readability.

Strategically use color and texture You can direct attention toward or redirect attention away from items using color, light, contrast, and texture to your advantage.

Use typography to create hierarchy and clarity Carefully consider how you use typeface. Different sizes, fonts, and arrangement of the text to help increase scanability, legibility and readability.

Make sure that the system communicates whats happening Always inform your users of location, actions, changes in state, or errors. The use of various UI elements to communicate status and, if necessary, next steps can reduce frustration for your user.

Think about the defaults By carefully thinking about and anticipating the goals people bring to your site, you can create defaults that reduce the burden on the user. This becomes particularly important when it comes to form design where you might have an opportunity to have some fields pre-chosen or filled out.

17.3 Deliverable – Mockup

You deliverable is a set of image files representing a mockup of each functionality your application will provide to the user.

Part V

Project Completion and Evaluation

Chapter 18

Final Presentation

18.1 Implementation Introduction

You have built a data component to your project. You have build a component containing the business rules and logic of your project. You have designed a user interface for your project. At this stage, you should be ready to put everything together. If this were a "real world" project, this would represent the first iteration at the implementation phase — you would still have to test your application, and move to the second iteration. Lather. Rinse. Repeat.

The purpose of this project is to introduce you to the *how* a project is developed. You will almost never be involved in every part of the development of an application, except very small applications. But someone will be involved in every stage, whether designing the information architecture and building the database, designing and building the business logic, and designing and building the user interface.

Your implementation will not be complete. However, it should demonstrate what you have learned in this course, and should have introduced you to some of the important concepts of software development. Implement what you can, and if you have the interest and the time to continue to work on your project, it would be time well spent.

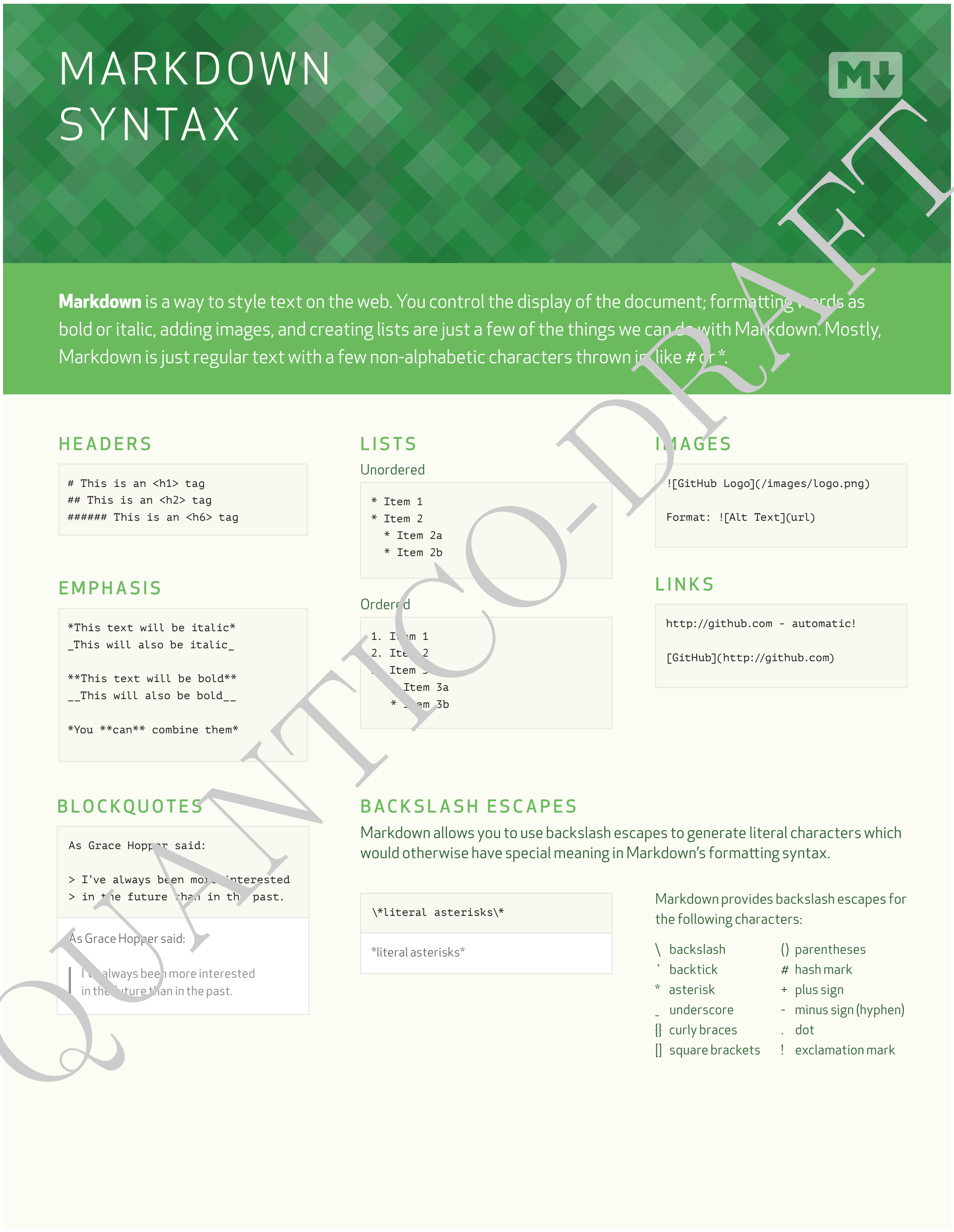
18.2 Deliverable – Implementation

You deliverable is a set of source files representing an implementation of your project, the interface, and *including the data and logic components.* These files must be C# files.

Appendices

Appendix A

Markdown Cheatsheet



MARKDOWN SYNTAX

Markdown is a way to style text on the web. You control the display of the document; formatting words as bold or italic, adding images, and creating lists are just a few of the things we can do with Markdown. Mostly, Markdown is just regular text with a few non-alphabetic characters thrown in, like # or *.

HEADERS

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

EMPHASIS

```
*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

LISTS

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

IMAGES

```
![GitHub Logo](/images/logo.png)

Format: ![Alt Text](url)
```

LINKS

```
http://github.com - automatic!

[GitHub](http://github.com)
```

BLOCKQUOTES

```
As Grace Hopper said:

> I've always been more interested
> in the future than in the past.
```

As Grace Hopper said:

> I've always been more interested in the future than in the past.

BACKSLASH ESCAPES

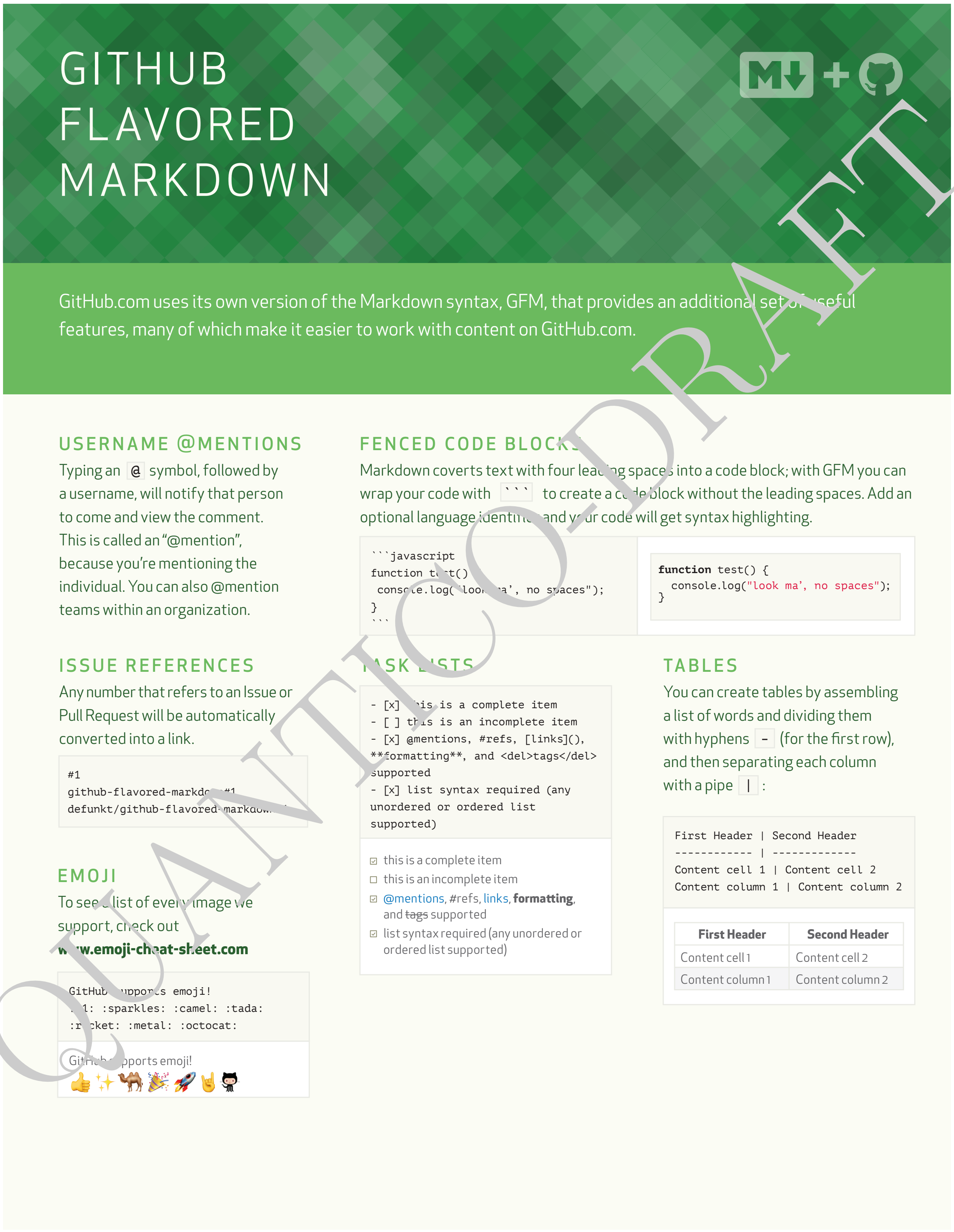
Markdown allows you to use backslash escapes to generate literal characters which would otherwise have special meaning in Markdown's formatting syntax.

```
\*literal asterisks\*
```

literal asterisks

Markdown provides backslash escapes for the following characters:

\	backslash	()	parentheses
`	backtick	#	hash mark
*	asterisk	+	plus sign
_	underscore	-	minus sign (hyphen)
{}	curly braces	.	dot
[]	square brackets	!	exclamation mark



GITHUB FLAVORED MARKDOWN

GitHub.com uses its own version of the Markdown syntax, GFM, that provides an additional set of useful features, many of which make it easier to work with content on GitHub.com.

USERNAME @MENTIONS

Typing an `@` symbol, followed by a username, will notify that person to come and view the comment. This is called an "@mention", because you're mentioning the individual. You can also @mention teams within an organization.

ISSUE REFERENCES

Any number that refers to an Issue or Pull Request will be automatically converted into a link.

```
#1
github-flavored-markdown#1
defunkt/github-flavored-markdown#1
```

EMOJI

To see a list of every image we support, check out **www.emoji-cheat-sheet.com**

```
GitHub supports emoji!
:+1: :sparkles: :camel: :tada:
:rocket: :metal: :octocat:
```

GitHub supports emoji!
👍 ✨ 🐫 🎉 🚀 🤘

FENCED CODE BLOCKS

Markdown coverts text with four leading spaces into a code block; with GFM you can wrap your code with ` ``` ` to create a code block without the leading spaces. Add an optional language identifier and your code will get syntax highlighting.

~~~
```javascript
function test() {
 console.log('look ma', no spaces");
}
```
~~~

```
function test() {
  console.log("look ma', no spaces");
}
```

TASK LISTS

```
- [x] this is a complete item
- [ ] this is an incomplete item
- [x] @mentions, #refs, [links](),
**formatting**, and <del>tags</del>
supported
- [x] list syntax required (any
unordered or ordered list
supported)
```

- ☑ this is a complete item
- ☐ this is an incomplete item
- ☑ @mentions, #refs, links, **formatting**, and ~~tags~~ supported
- ☑ list syntax required (any unordered or ordered list supported)

TABLES

You can create tables by assembling a list of words and dividing them with hyphens `-` (for the first row), and then separating each column with a pipe `|`:

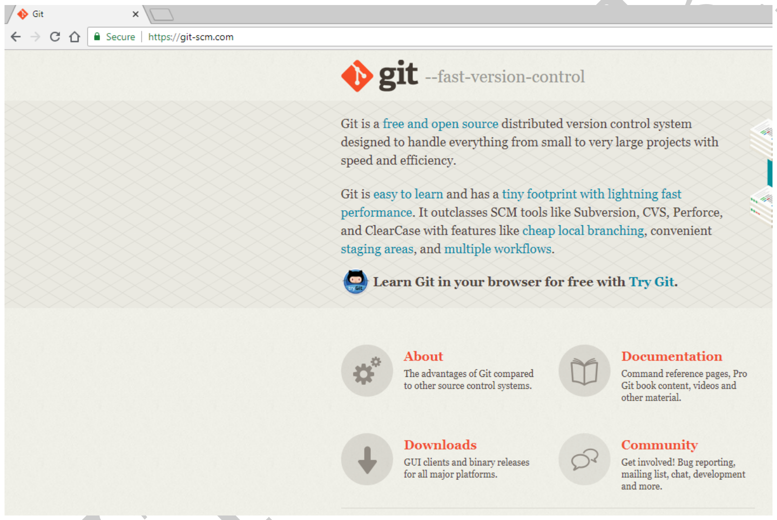
```
First Header | Second Header
------------ | -------------
Content cell 1 | Content cell 2
Content column 1 | Content column 2
```

First Header	Second Header
Content cell 1	Content cell 2
Content column 1	Content column 2

Appendix B

Installing git

This will contain instructions to install the git version control system.



Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Learn Git in your browser for free with Try Git.

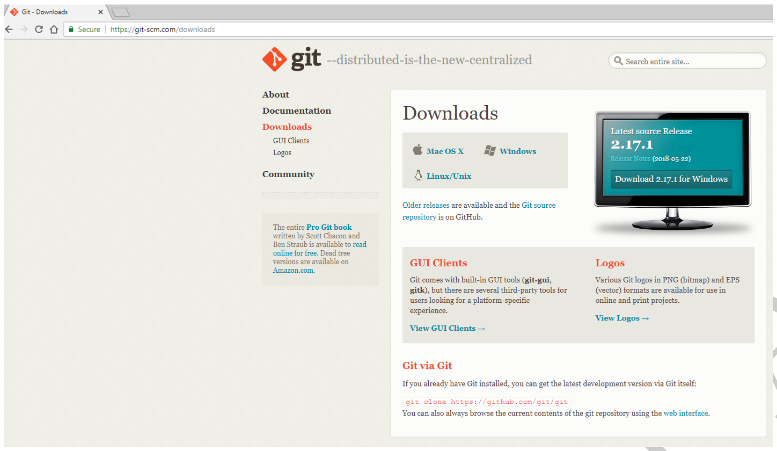
About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Figure B.1:



Git - Downloads

Secure | https://git-scm.com/downloads

git --distributed-is-the-new-centralized

Search entire site...

About
Documentation
Downloads
GUI Clients
Logos
Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloads

Mac OS X **Windows**
Linux/Unix

Older releases are available and the Git source repository is on GitHub.

Latest source Release
2.17.1
Release Notes (2018-05-22)
Download 2.17.1 for Windows

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

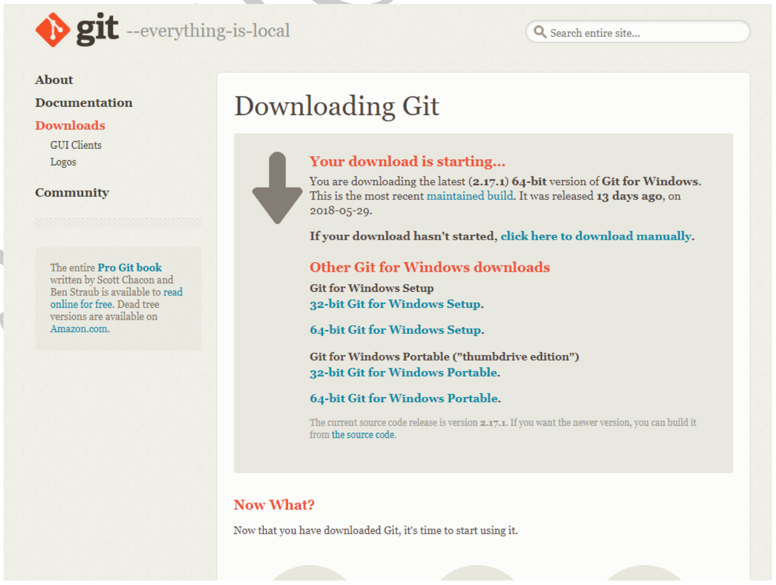
Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the web interface.

Figure B.2:



git --everything-is-local

Search entire site...

About
Documentation
Downloads
GUI Clients
Logos
Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloading Git

Your download is starting...

You are downloading the latest (**2.17.1**) **64-bit** version of **Git for Windows**. This is the most recent maintained build. It was released **13 days ago**, on 2018-05-29.

If your download hasn't started, click here to download manually.

Other Git for Windows downloads

Git for Windows Setup
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Git for Windows Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

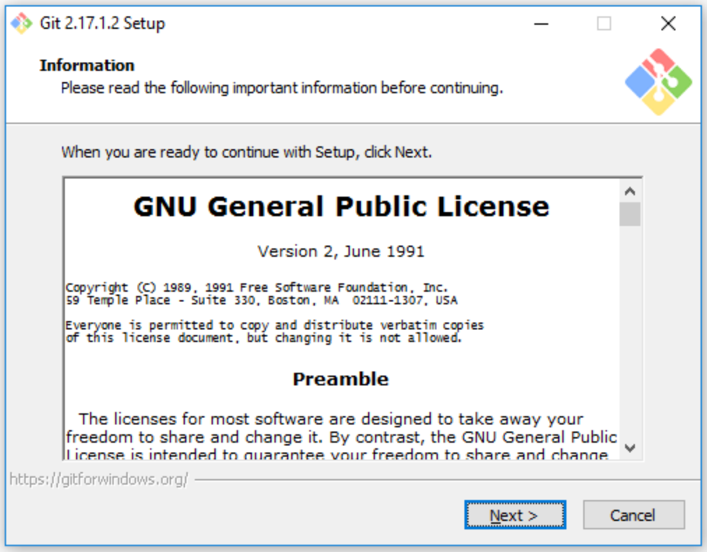
64-bit Git for Windows Portable.

The current source code release is version **2.17.1**. If you want the newer version, you can build it from the source code.

Now What?

Now that you have downloaded Git, it's time to start using it.

Figure B.3:



Git 2.17.1.2 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

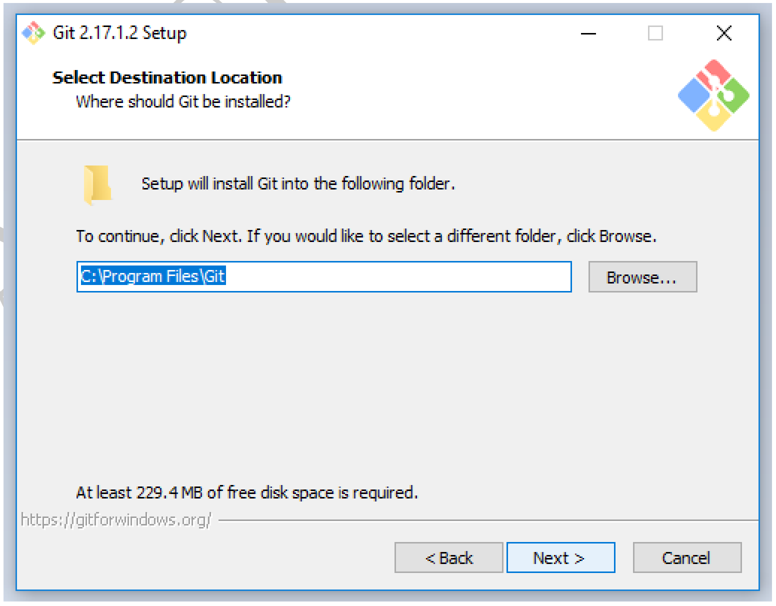
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Figure B.4:



Git 2.17.1.2 Setup

Select Destination Location

Where should Git be installed?

Setup will install Git into the following folder.

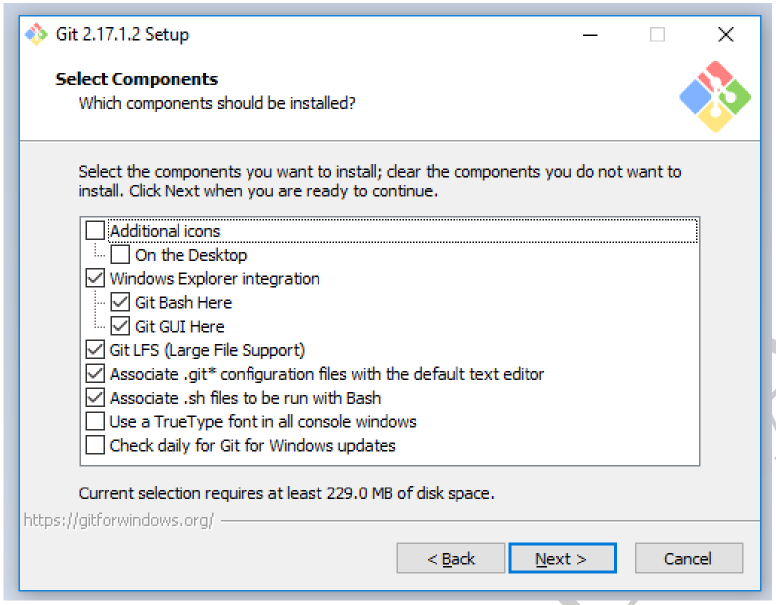
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Git

At least 229.4 MB of free disk space is required.

https://gitforwindows.org/

Figure B.5:


Git 2.17.1.2 Setup

Select Components

Which components should be installed?

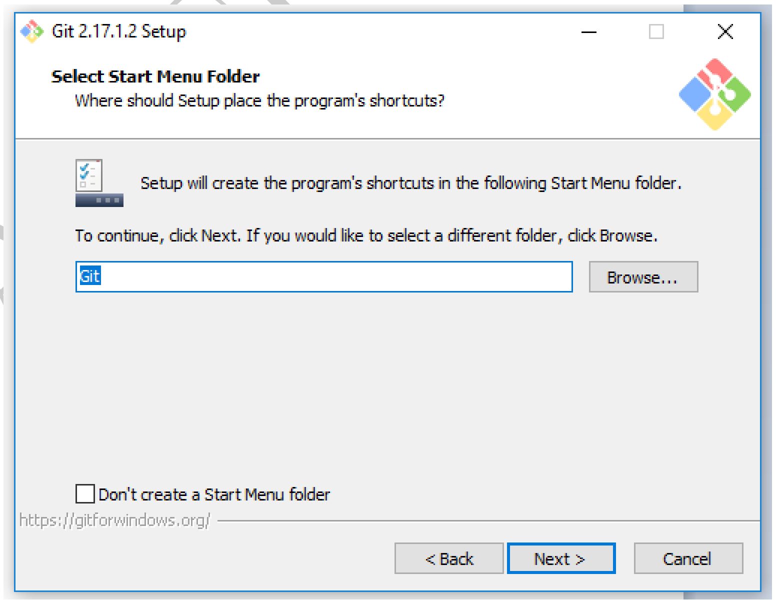
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 229.0 MB of disk space.

https://gitforwindows.org/

Figure B.6:


Git 2.17.1.2 Setup

Select Start Menu Folder

Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

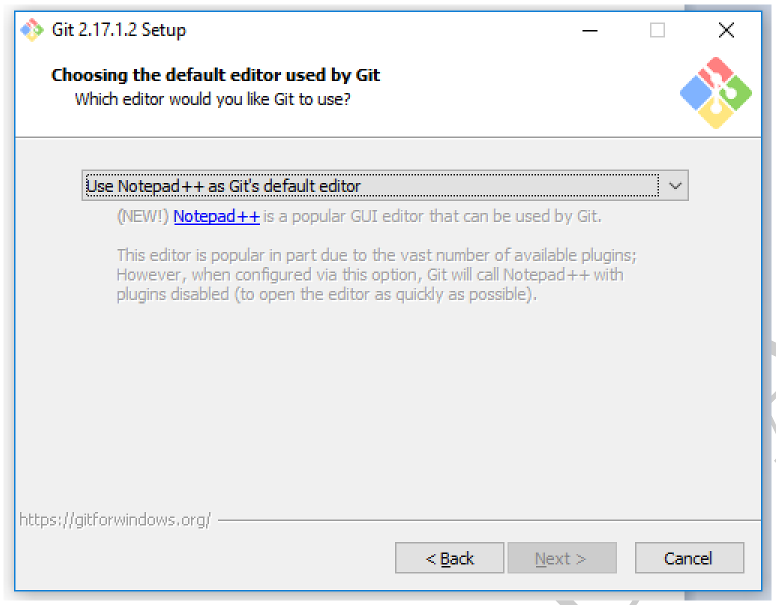
To continue, click Next. If you would like to select a different folder, click Browse.

Git

- [ ] Don't create a Start Menu folder

https://gitforwindows.org/

Figure B.7:



Git 2.17.1.2 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

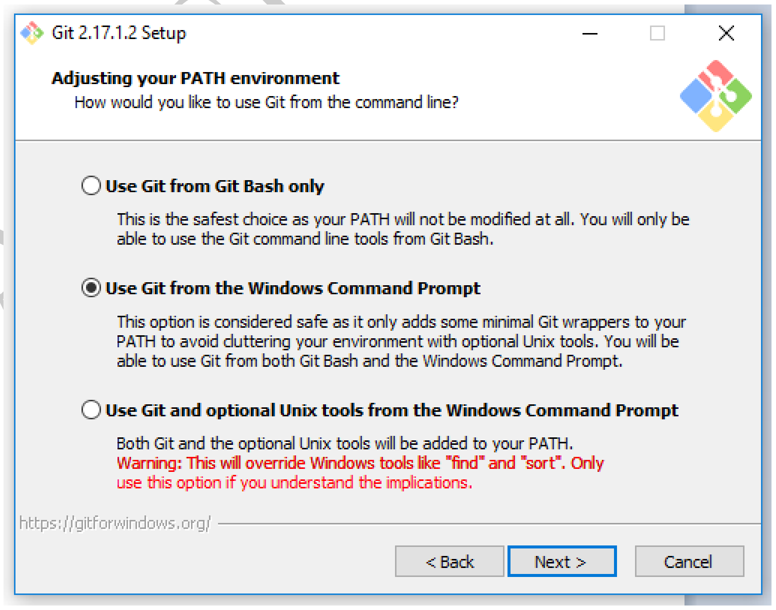
Use Notepad++ as Git's default editor

(NEW!) Notepad++ is a popular GUI editor that can be used by Git.

This editor is popular in part due to the vast number of available plugins; However, when configured via this option, Git will call Notepad++ with plugins disabled (to open the editor as quickly as possible).

https://gitforwindows.org/

Figure B.8:



Git 2.17.1.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

Use Git from Git Bash only

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

[✓ **Use Git from the Windows Command Prompt**]

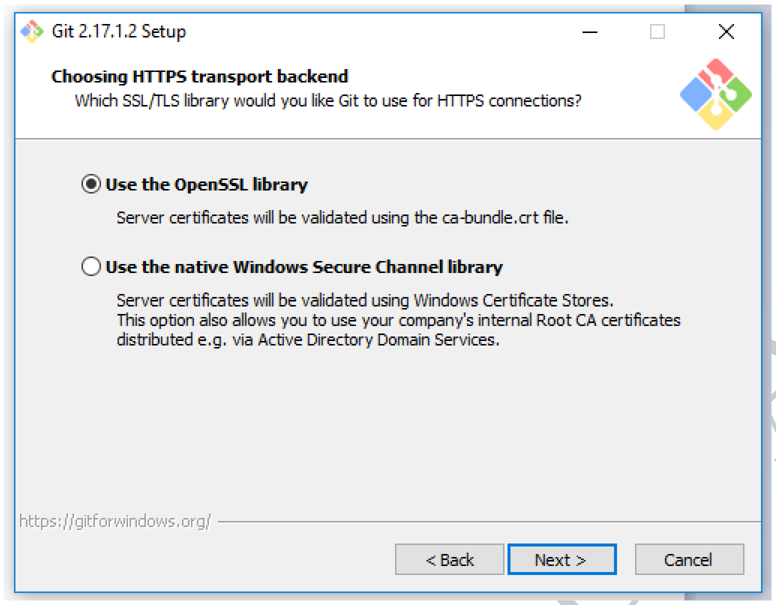
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://gitforwindows.org/

Figure B.9:



Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?

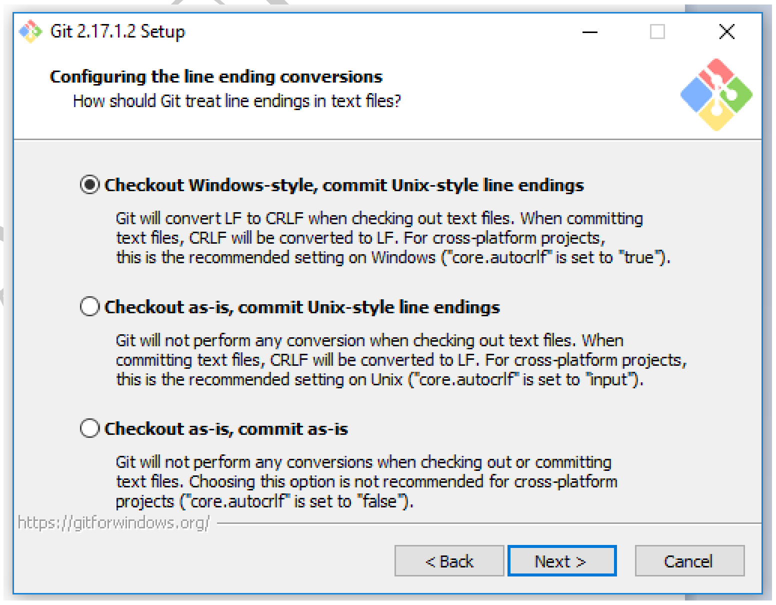
[✓ Use the OpenSSL library]

Server certificates will be validated using the ca-bundle.crt file.

Use the native Windows Secure Channel library

Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.

Figure B.10:



Configuring the line ending conversions
How should Git treat line endings in text files?

[✓ Checkout Windows-style, commit Unix-style line endings]

Git will convert LF to CRLF when checking out text files. When committing
text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Windows ("core.autocrlf" is set to "true").

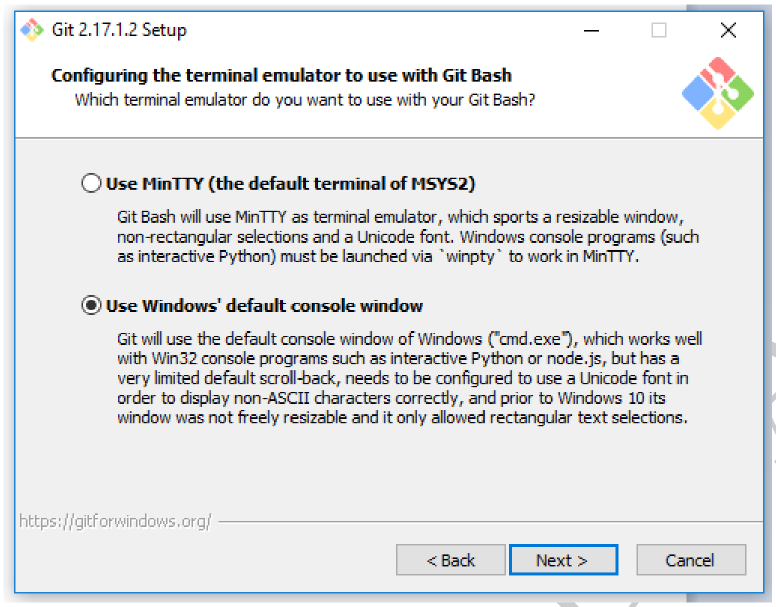
Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files. When
committing text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Unix ("core.autocrlf" is set to "input").

Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing
text files. Choosing this option is not recommended for cross-platform
projects ("core.autocrlf" is set to "false").

Figure B.11:



Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

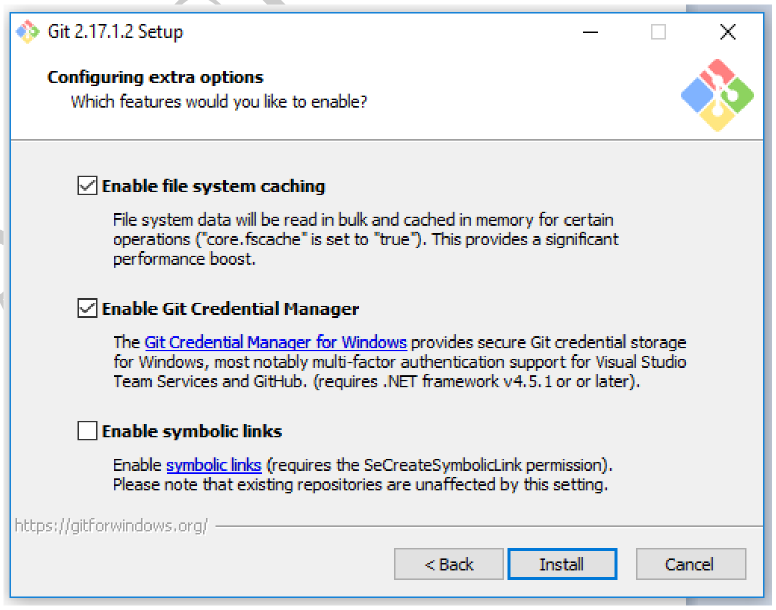
( ) **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

(●) **Use Windows' default console window**

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

Figure B.12:



Configuring extra options

Which features would you like to enable?

[✓] **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

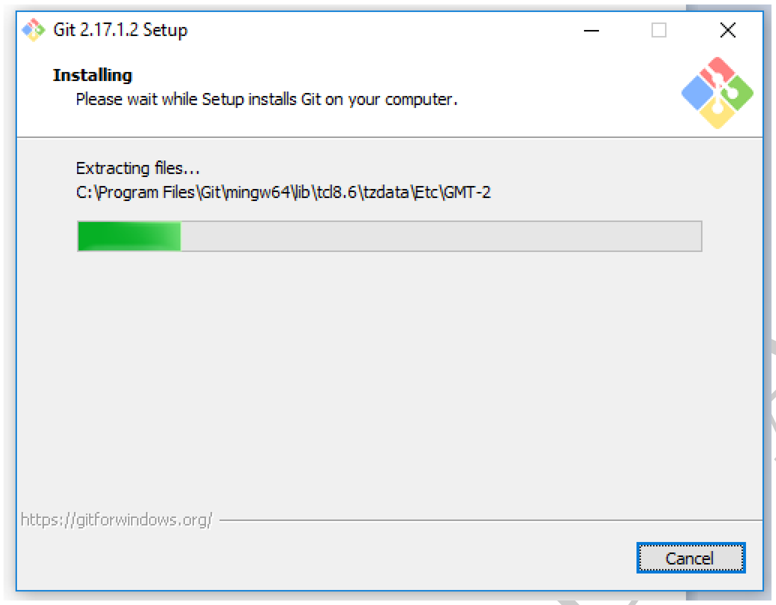
[✓] **Enable Git Credential Manager**

The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or or later).

[ ] **Enable symbolic links**

Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

Figure B.13:



Git 2.17.1.2 Setup

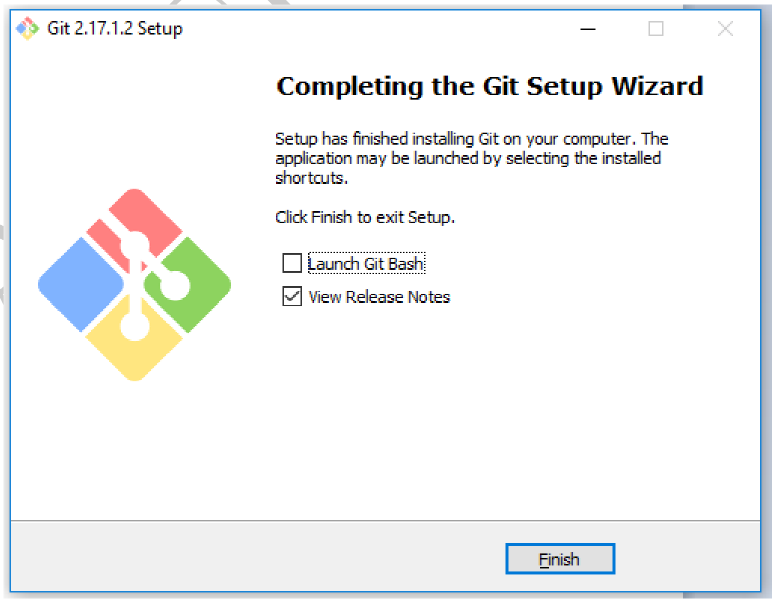
Installing

Please wait while Setup installs Git on your computer.

Extracting files...
C:\Program Files\Git\mingw64\lib\tcl8.6\tzdata\Etc\GMT-2

https://gitforwindows.org/

Figure B.14:



Git 2.17.1.2 Setup

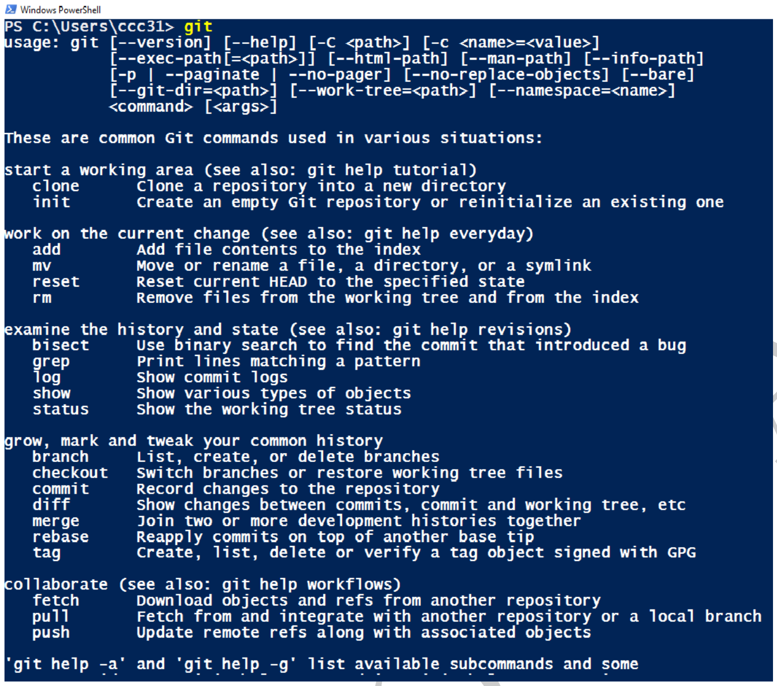
Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

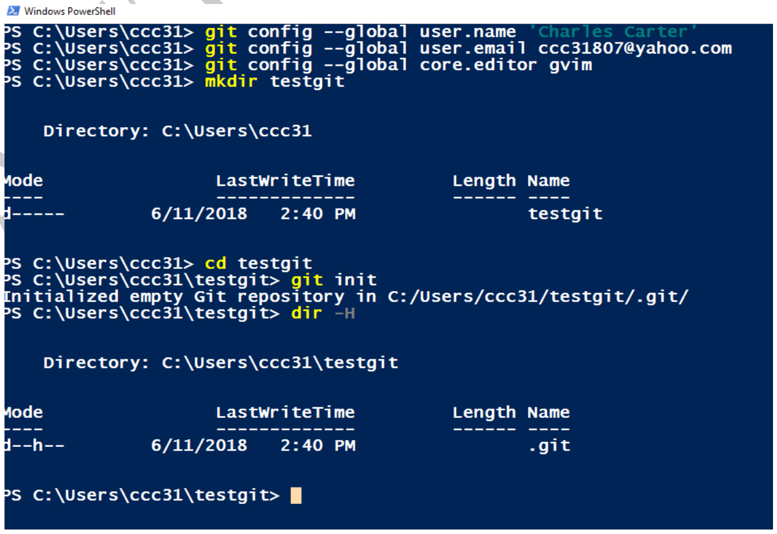
☐ Launch Git Bash
☑ View Release Notes

Figure B.15:



```
PS C:\Users\ccc31> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
```

Figure B.16:

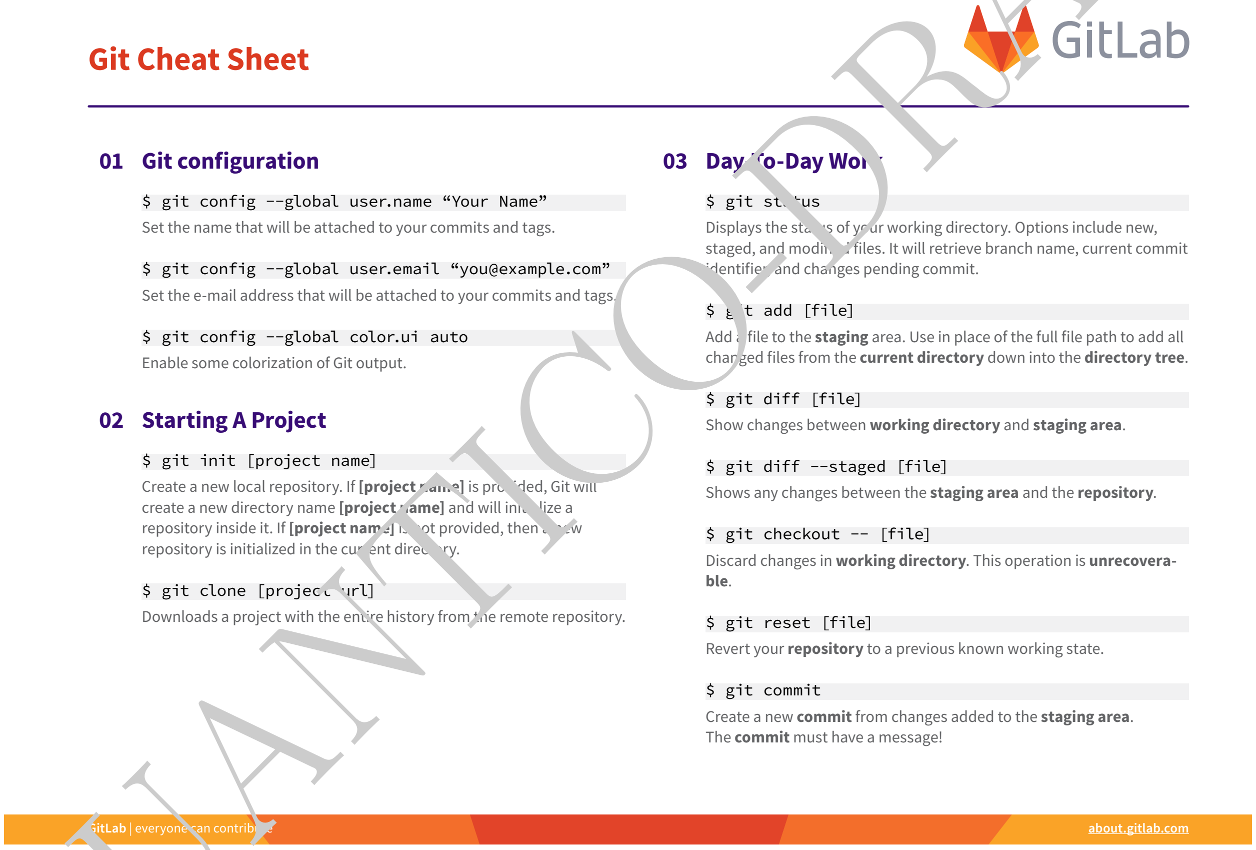


```
PS C:\Users\ccc31> git config --global user.name 'Charles Carter'
PS C:\Users\ccc31> git config --global user.email ccc31807@yahoo.com
PS C:\Users\ccc31> git config --global core.editor gvim
PS C:\Users\ccc31> mkdir testgit


    Directory: C:\Users\ccc31


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        6/11/2018   2:40 PM                testgit


PS C:\Users\ccc31> cd testgit
PS C:\Users\ccc31\testgit> git init
Initialized empty Git repository in C:/Users/ccc31/testgit/.git/
PS C:\Users\ccc31\testgit> dir -H


    Directory: C:\Users\ccc31\testgit


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d--h--        6/11/2018   2:40 PM                .git


PS C:\Users\ccc31\testgit>
```

Figure B.17:

Appendix C

Git Cheatsheet



Git Cheat Sheet

GitLab

01 Git configuration

```
$ git config --global user.name "Your Name"
```

Set the name that will be attached to your commits and tags.

```
$ git config --global user.email "you@example.com"
```

Set the e-mail address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Enable some colorization of Git output.

02 Starting A Project

```
$ git init [project name]
```

Create a new local repository. If **[project name]** is provided, Git will create a new directory name **[project name]** and will initialize a repository inside it. If **[project name]** is not provided, then a new repository is initialized in the current directory.

```
$ git clone [project url]
```

Downloads a project with the entire history from the remote repository.

03 Day-To-Day Work

```
$ git status
```

Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

```
$ git add [file]
```

Add a file to the **staging** area. Use in place of the full file path to add all changed files from the **current directory** down into the **directory tree**.

```
$ git diff [file]
```

Show changes between **working directory** and **staging area**.

```
$ git diff --staged [file]
```

Shows any changes between the **staging area** and the **repository**.

```
$ git checkout -- [file]
```

Discard changes in **working directory**. This operation is **unrecoverable**.

```
$ git reset [file]
```

Revert your **repository** to a previous known working state.

```
$ git commit
```

Create a new **commit** from changes added to the **staging area**. The **commit** must have a message!

GitLab | everyone can contribute

about.gitlab.com

```
$ git rm [file]
```
Remove file from **working directory** and **staging area**.

```
$ git stash
```
Put current changes in your **working directory** into **stash** for later use.

```
$ git stash pop
```
Apply stored **stash** content into **working directory**, and clear **stash**.

```
$ git stash drop
```
Delete a specific **stash** from all your previous **stashes**.

04 Git branching model

```
$ git branch [-a]
```
List all local branches in repository. With **-a**: show all branches (with remote).

```
$ git branch [branch_name]
```
Create new branch, referencing the current **HEAD**.

```
$ git checkout [-b][branch_name]
```
Switch **working directory** to the specified branch. With **-b**: Git will create the specified branch if it does not exist.

```
$ git merge [from name]
```
Join specified **[from name]** branch into your current branch (the one you are on currently).

```
$ git branch -d [name]
```
Remove selected branch, if it is already merged into any other. **-D** instead of **-d** forces deletion.

05 Review your work

```
$ git log [-n count]
```
List commit history of current branch. **-n count** limits list to last **n** commits.

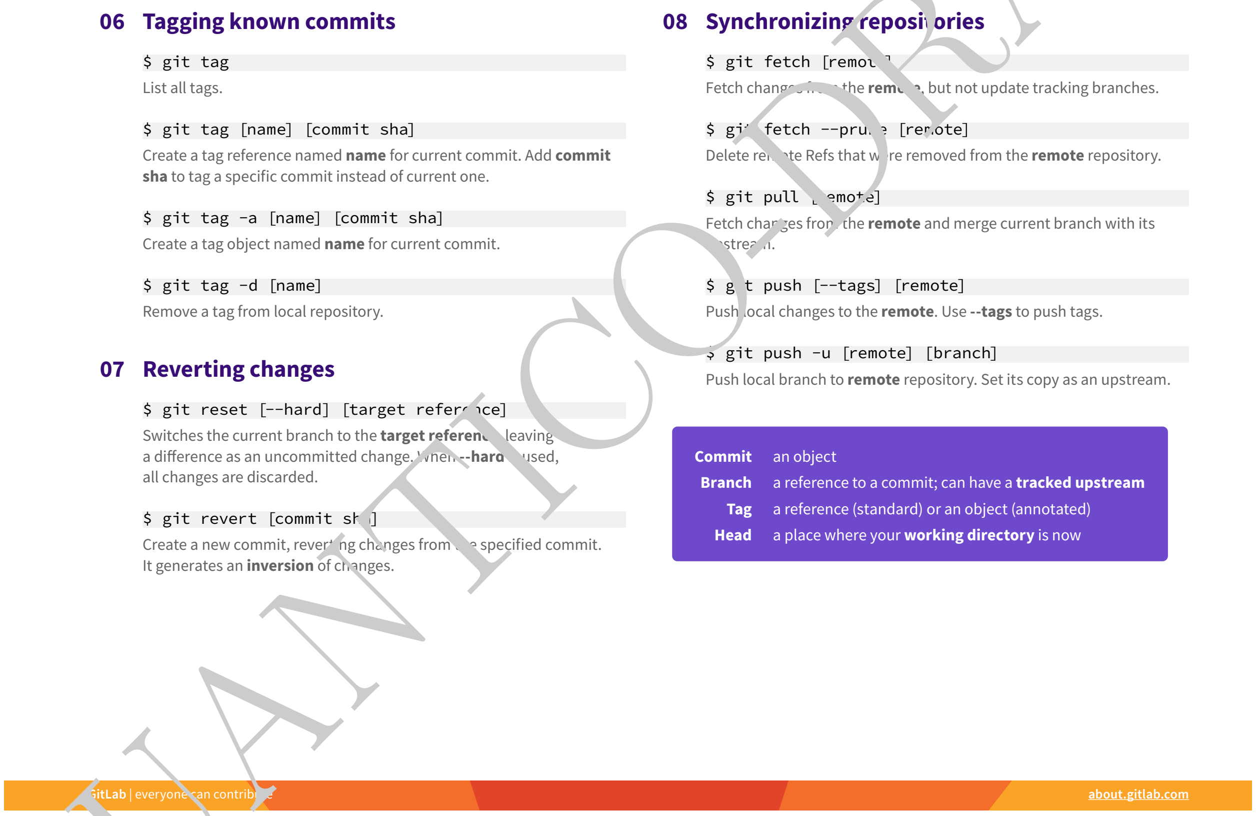
```
$ git log --oneline --graph --decorate
```
An overview with reference labels and history graph. One commit per line.

```
$ git log ref..
```
List commits that are present on the current branch and not merged into **ref**. A **ref** can be a branch name or a tag name.

```
$ git log ..ref
```
List commit that are present on **ref** and not merged into current branch.

```
$ git reflog
```
List operations (e.g. checkouts or commits) made on local repository.

GitLab | everyone can contribute about.gitlab.com



06 Tagging known commits

```
$ git tag
```
List all tags.

```
$ git tag [name] [commit sha]
```
Create a tag reference named **name** for current commit. Add **commit sha** to tag a specific commit instead of current one.

```
$ git tag -a [name] [commit sha]
```
Create a tag object named **name** for current commit.

```
$ git tag -d [name]
```
Remove a tag from local repository.

07 Reverting changes

```
$ git reset [--hard] [target reference]
```
Switches the current branch to the **target reference**, leaving a difference as an uncommitted change. When **--hard** is used, all changes are discarded.

```
$ git revert [commit sha]
```
Create a new commit, reverting changes from the specified commit. It generates an **inversion** of changes.

08 Synchronizing repositories

```
$ git fetch [remote]
```
Fetch changes from the **remote**, but not update tracking branches.

```
$ git fetch --prune [remote]
```
Delete remote Refs that were removed from the **remote** repository.

```
$ git pull [remote]
```
Fetch changes from the **remote** and merge current branch with its upstream.

```
$ git push [--tags] [remote]
```
Push local changes to the **remote**. Use **--tags** to push tags.

```
$ git push -u [remote] [branch]
```
Push local branch to **remote** repository. Set its copy as an upstream.

Commit	an object
Branch	a reference to a commit; can have a **tracked upstream**
Tag	a reference (standard) or an object (annotated)
Head	a place where your **working directory** is now

GitLab | everyone can contribute   about.gitlab.com

A Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/Ubuntu please type in the **terminal**:

```
$ sudo apt-get install git
```

If you need to install Git from source, you can get it from git-scm.com/downloads.

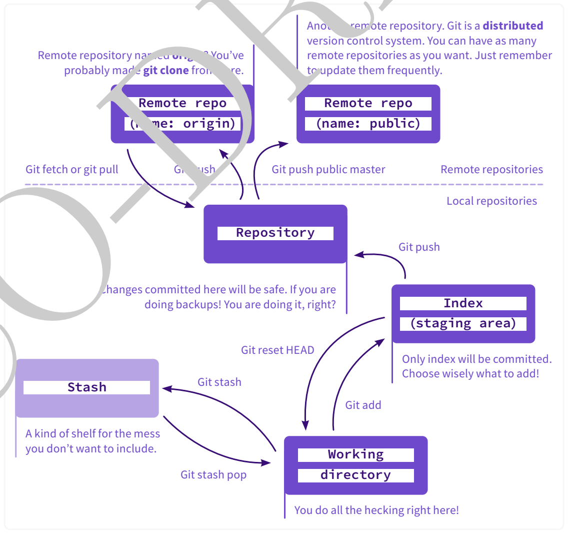
An excellent Git course can be found in the great **Pro Git** book by Scott Chacon and Ben Straub. The book is available online for free at git-scm.com/book.

B Ignoring Files

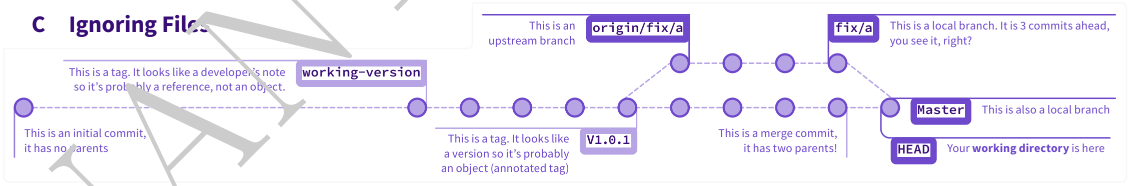
```
$ cat .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

Verify the .gitignore file exists in your project and ignore certain type of files, such as all files in **logs** directory (excluding the **.gitkeep** file), whole **tmp** directory and all files ***.swp**. File ignoring will work for the directory (and children directories) where **.gitignore** file is placed.

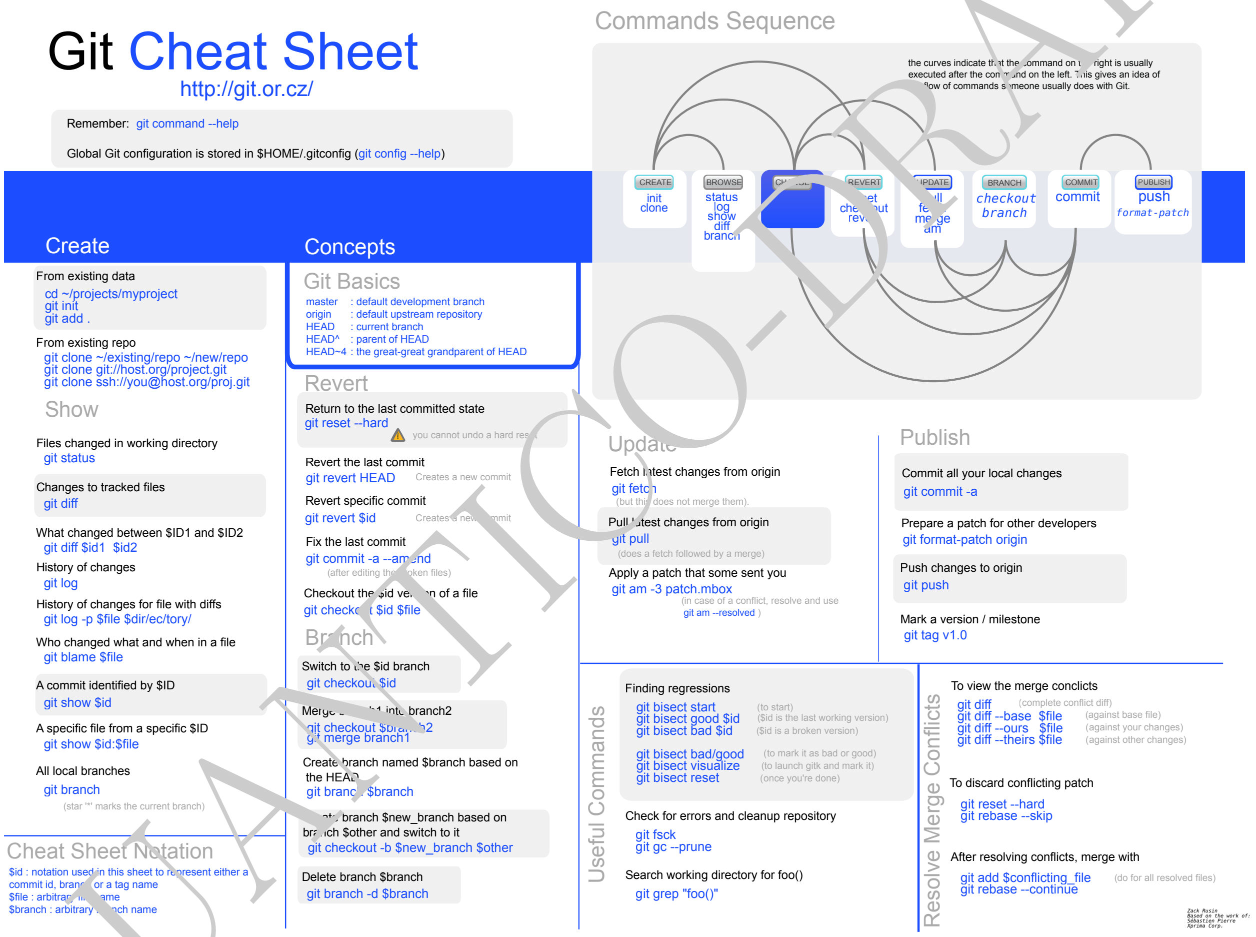
D The zoo of working areas


Remote repository named origin? You've probably made git clone from here.
Another remote repository. Git is a distributed version control system. You can have as many remote repositories as you want. Just remember to update them frequently.
Remote repo (name: origin)
Remote repo (name: public)
Git fetch or git pull
Git push
Git push public master
Remote repositories
Local repositories
Repository
Git push
Changes committed here will be safe. If you are doing backups! You are doing it, right?
Index (staging area)
Only index will be committed. Choose wisely what to add!
Git reset HEAD
Git stash
Git add
Stash
A kind of shelf for the mess you don't want to include.
Git stash pop
Working directory
You do all the hecking right here!

C Ignoring Files


This is an upstream branch
origin/fix/a
fix/a
This is a local branch. It is 3 commits ahead, you see it, right?
This is a tag. It looks like a developer's note so it's probably a reference, not an object.
working-version
This is an initial commit, it has no parents
This is a tag. It looks like a version so it's probably an object (annotated tag)
V1.0.1
This is a merge commit, it has two parents!
Master
This is also a local branch
HEAD
Your working directory is here

GitLab | everyone can contribute about.gitlab.com


CREATE
init
clone
BROWSE
status
log
show
diff
branch
CHANGE
REVERT
reset
checkout
revert
UPDATE
pull
fetch
merge
am
BRANCH
checkout
branch
COMMIT
commit
PUBLISH
push
format-patch

Git Cheat Sheet

http://git.or.cz/

Remember: git command --help

Global Git configuration is stored in $HOME/.gitconfig (git config --help)

Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between $ID1 and $ID2

```
git diff $id1  $id2
```

History of changes

```
git log
```

History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by $ID

```
git show $id
```

A specific file from a specific $ID

```
git show $id:$file
```

All local branches

```
git branch
```

(star '*' marks the current branch)

Cheat Sheet Notation

$id : notation used in this sheet to represent either a commit id, branch or a tag name
$file : arbitrary file name
$branch : arbitrary branch name

Concepts

Git Basics

master	: default development branch
origin	: default upstream repository
HEAD	: current branch
HEAD^	: parent of HEAD
HEAD~4	: the great-great grandparent of HEAD

Revert

Return to the last committed state

```
git reset --hard
```

⚠ you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files)

Checkout the $id version of a file

```
git checkout $id $file
```

Branch

Switch to the $id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named $branch based on the HEAD

```
git branch $branch
```

Create branch $new_branch based on branch $other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch $branch

```
git branch -d $branch
```

Commands Sequence

the curves indicate that the command on the right is usually executed after the command on the left. This gives an idea of the flow of commands someone usually does with Git.

Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use git am --resolved)

Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

Useful Commands

Finding regressions

```
git bisect start         (to start)
git bisect good $id      ($id is the last working version)
git bisect bad $id       ($id is a broken version)

git bisect bad/good      (to mark it as bad or good)
git bisect visualize     (to launch gitk and mark it)
git bisect reset         (once you're done)
```

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

Resolve Merge Conflicts

To view the merge conclicts

```
git diff                (complete conflict diff)
git diff --base  $file     (against base file)
git diff --ours   $file    (against your changes)
git diff --theirs $file    (against other changes)
```

To discard conflicting patch

```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with

```
git add $conflicting_file      (do for all resolved files)
git rebase --continue
```

Zack Rusin
Based on the work of:
Sébastien Pierre
Xprima Corp.

Appendix D

C# Console MVC

by Steve Conger,
Monday, January 14, 2013,
Source:
`http://congeritc.blogspot.com/2013/01/mvc-example-with-c-console-program.html`

Program.cs This class contains the main method. All it does is instantiate an object of class `TipCalculatorController` and initialize it by calling the constructor.

```
/*******************************************
* Program writte by: Steve Conger
* Accessed June 28, 2020
* http://congeritc.blogspot.com/2013/01/mvc-example-with-c-console-program.html
******************************************/

namespace MVCSample
{
    class Program
    {
        static void Main(string[] args)
        {
            TipCalculatorController t = new TipCalculatorController();
        }
    }
}
```

TipCalculatorController.cs This class is the Controller. It knows about `Tip` and`Display`. It is the *only* class in the application that knows about the Model and the View.

```
namespace MVCSample
{
    class TipCalculatorController
    {
        ///
        /// The TipCalculatorController class brings together
        /// the display and the tip or model classes
        /// I use the constructor to instantiate the Display.
        /// Instantiating the Display calls its constructor
        /// which calls the Get input method
        /// Once the input is entered I can instantiate
        /// the Tip class and pass the values from the
        /// Display class. Notice the dot notation and observe
        /// how the two classes interact
        ///
        // private fields
        private Tip tip;
        private Display display;
        // constructor
        public TipCalculatorController()
        {
            display = new Display();
            tip = new Tip(display.Amt, display.Percentage);
            display.TipAmount = tip.CalculateTip();
            display.Total = tip.CalculateTotal();
            display.ShowTipandTotal();
        }
    }
}
```

Tip.cs This class is the Model. It knows nothing about `TipCalculatorController` and`Display`.

```
namespace MVCSample
{
    class Tip
    {
        ///
        /// This class does a very simple tip
        /// calculation. It has two fields amount and
        /// tip percent. We are ignoring tax and whether
        /// we tip before or after tax. The point is the
        /// MVC model. This is the model part of MVC.
        /// It does the calculations and handles the data
        /// it is totally unaware of the Display class
        /// or the controller
        ///
        //private fields
        private double amount;
        private double tipPercent;
        //default constructor
        public Tip()
        {
            Amount = 0;
            TipPercent = 0;
        }
        //overloaded constructor
        public Tip(double amt, double percent)
        {
            Amount = amt;
            TipPercent = percent;
        }
        //public properties
        public double Amount
        {
            get { return amount; }
            set { amount = value; }
        }
        public double TipPercent
        {
            get { return tipPercent; }
            set
            {
                //here we check to see if
                //they entered the percent
                //as a decimal or a whole number
                //if it is a whole number
                //larger than 1 we divide it by
                //100, so the highest possible tip
                //is 100%
                if (value > 1)
                {
                    value /= 100;
                }
                tipPercent = value;
            }
        }
        public double CalculateTip()
        {
            //very simplistic tip calculation
            return Amount * TipPercent;
        }
        public double CalculateTotal()
        {
            //simple total calculation
            return CalculateTip() + Amount;
        }
    }
}
```

Display.cs This class is the View. It knows nothing about `TipCalculatorController` and`Display`.

```
using System;

namespace MVCSample
{
    class Display
    {
        ///
        /// this is the Display class. Its purpose is
        /// to gather the input and display the output
        /// Unlike our usual display I have made the
        /// variables Class level fields. This is so
        /// the Controller can have access to the fields
        /// through the properties. The Display class
        /// is totally unaware of the Tip class (the model)
        /// or the controller
        ///
        //private fields
        private double perc;
        private double amt;
        private double total;
        private double tipAmount;
        //constructor
        public Display()
        {
            Percentage = 0;
            TipAmount = 0;
            Amt = 0;
            Total = 0;
            GetValues();
        }
        //public properties
        public double TipAmount
        {
            get { return tipAmount; }
            set { tipAmount = value; }
        }
        public double Total
        {
            get { return total; }
            set { total = value; }
        }
        public double Percentage
        {
            get { return perc; }
            set { perc = value; }
        }
        public double Amt
        {
            get { return amt; }
            set { amt = value; }
        }
        //private method for getting input
        //it is called in the constructor
        private void GetValues()
        {
            Console.WriteLine("Enter␣the␣Amount␣of␣the␣meal");
            Amt = double.Parse(Console.ReadLine());

            Console.WriteLine("Enter␣the␣percent␣you␣want␣to␣tip");
            Percentage = double.Parse(Console.ReadLine());
        }
        //public method to show output
        //public so I can access it from the controller
        public void ShowTipandTotal()
        {
            Console.WriteLine("Your␣tip␣is␣{0:C}", TipAmount);
            Console.WriteLine("The␣total␣will␣be␣{0:C}", Total);
            Console.ReadKey();
        }
    }
}
```

Last note

The last word.