Principles of Software Engineering

Ramaswamy Krishnan Chittur

Scope

- This course deals with how to design and develop a reasonably big software – big enough that it warrants multiple phases and multiple developers
  - Software development lifecycle.
  - Spec the software requirements.
  - Model the software using UML diagrams.
  - Architecting big software. This involves analyzing the following:
    - The various components that are required.
    - The dependencies between the components, and how they would communicate with each other.
    - Make well thought-out design choices (cost-benefit ratio, risks).
  - Design and design patterns.
  - Predict the performance.
  - Implement.
  - Validate.
  - Track the progress of the project throughout the development cycle.

Fundamentals of good software

- ❑ Good software has:

1. A good design document.
2. Well designed components.
3. Safe, clean and well structured code.
4. Fitting data structures and good algorithms.
5. Good test matrix.

Software Development Life Cycle (SDLC)

- Systematic process to develop a large software system:
  - Gathering requirements
  - Writing specification document
  - Architecture and Design
  - Development
  - Integration
  - Testing and Validation
  - Deployment
  - Maintenance
- Common SDLC Models:
  - Waterfall
  - Agile
  - More models some of which are hybrid of these two fundamental SDLC models

Evolution of programming models

- Goal:
  - Improving management of large software systems.

- Programming models:
  - SP - Structured Programming:

    Decomposes large monolithic programs into modules using functional decomposition.
  - OOD - Object Oriented Design:

    Associates data with functions allowed to act on that data, e.g., an object, proven to be an effective structuring tool.
  - COM - Component Object Model:

    Reduces interdependencies between objects, using interfaces and object factories for isolation.
  - AOP - Aspect Oriented Programming:

    Separates program’s primary functionality from needed support infrastructure to simplify a program’s logic.

Object Oriented Design (OOD)

- The four basic principles of object-oriented programming are:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Reference:
  - Object-Oriented Programming (C#) | Microsoft Docs
- Exercises:
  - Try the exercises at Object-Oriented Programming (C#) | Microsoft Docs
  - Consider a class that manages persistence:
    - Provides the following functions:
      - void Store(string id, string value)
      - string Retrieve(string id)
    - Implement three different classes that will persist the data using: XML, JSON and Plain Text.

Software Contracts

- Interfaces and contracts:
  - Car <-> **Axle** <-> Wheel
  - Door <-> **Hinge** <-> Wall
- Why do we need a contract?
  - Componentization to develop independently and concurrently.
- How do we specify a software contract?
  - Through an interface
- Example:
  - *User interface* <-> **Communication Interface** <-> *Networking*.
  - In the above example, the contract is simple. ***Send, Receive***.
    - Send takes in a message and address from the UI and passes it to the networking component.
    - Receive dynamically shares with the UI, the message received by the networking component.
    - Client is agnostic to implementation. Communication channel can use sockets, named pipes, tcp, http etc. UI (client) does not care.

Unified Modelling Language (UML)

- Visualizes the architecture and design of a software system. After all, a picture speaks a thousand words as they say.
- Various types of UML diagrams:
  - Module diagram
    - No cyclic dependency
    - Lower level modules may not depend on upper level modules
  - Class diagram
    - Relationship between classes: Composition, Aggregation, Inheritance and Using.
  - Activity diagram
- Reference:
  - UML 2.5 Diagrams Overview (uml-diagrams.org)
  - Practical UML□A Hands-On Introduction for Developers (embarcadero.com)
- Exercises:
  - Draw a class diagram that shows the relationship between the following types:

    Interface IAutomobile, Class Engine, Class Tyre, Class SnowChain, Class Passenger, and Class Car.

Specification Document (Spec)

- Blueprint of the software system being developed.
- Typical spec contains:
  - Requirements
  - Goals
  - Architecture and Design, use UMLs to represent ideas
  - Interfaces, Prototype code
  - Analysis of performance, security
  - Validation techniques
- Examples:
  - GitHub - chittur/parallel-programming-language: A new language for parallel programming, and its compiler and runtime
  - From the course Moodle page here.

Specs and design docs

> “Writing is a rigorous test of simplicity: It is just not possible to write convincingly about ideas that cannot be understood.” –
>
> Dr. Per Brinch Hansen.

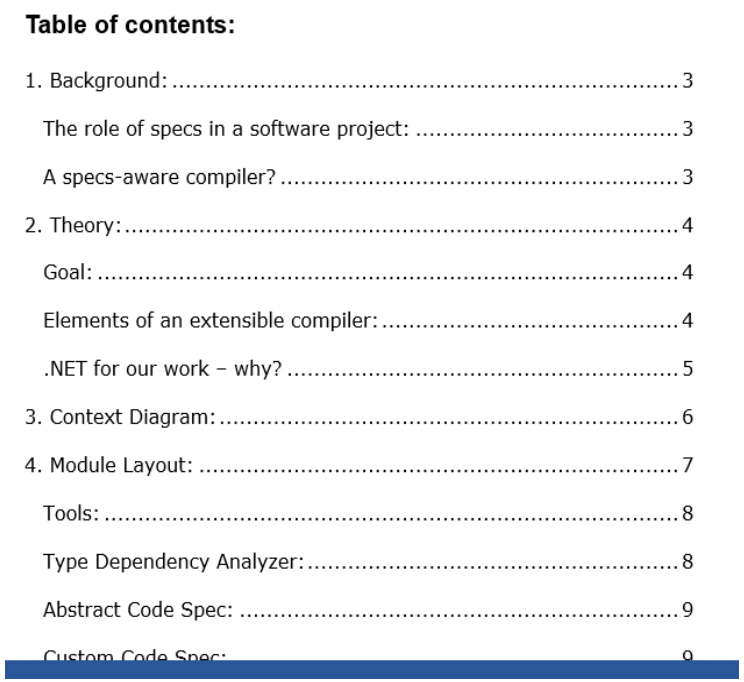
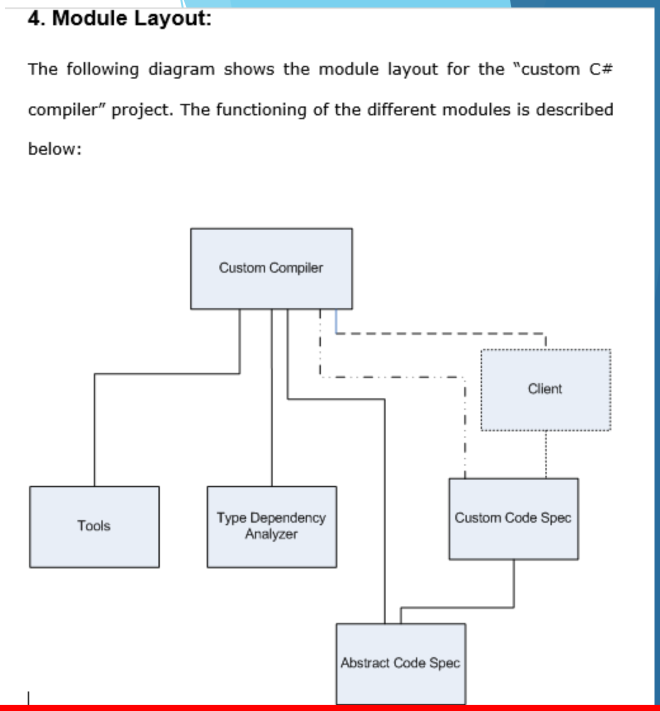
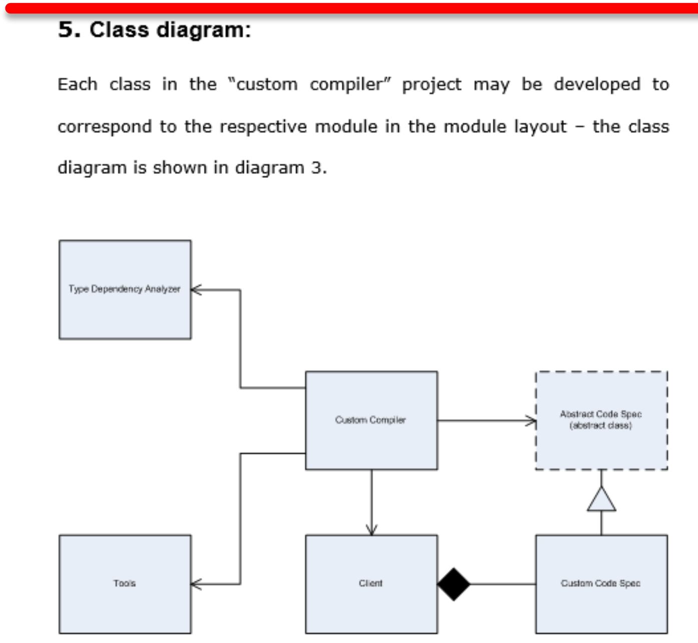


Table of contents:



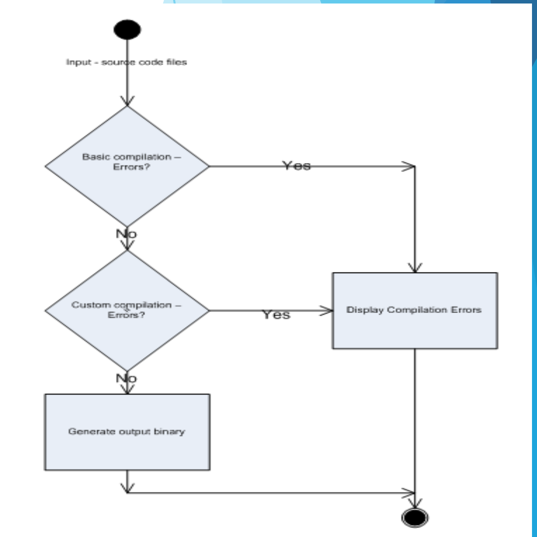
4. Module Layout:

The following diagram shows the module layout for the "custom C# compiler" project. The functioning of the different modules is described below:



5. Class diagram:

Each class in the "custom compiler" project may be developed to correspond to the respective module in the module layout – the class diagram is shown in diagram 3.



Input - source code files
Basic compilation – Errors?
Yes
No
Custom compilation – Errors?
Yes
No
Display Compilation Errors
Generate output binary

Figure 4: Activity Diagram

Software Design Patterns

- Best practices and solutions to common problems in software development
- Categories:
  - Creational
    - Examples: Singleton, Factory
  - Structural
    - Examples: Adapter, Composite
  - Behavioural
    - Examples: Publisher-Subscriber, Chain of responsibility
- References:
  - Design Patterns: Elements of Reusable Object-Oriented Software - *Ralph Johnson · Erich Gamma · Richard Helm · John Vlissides*.
- Exercises:
  - Try out the design patterns sample from our Moodle here. What are the design patterns you can find in that code sample?

User Interface (UI) programming

- Various UI frameworks:
  - MFC, WPF, WinUI, Swift UI etc.
- We will be focusing on WPF and XAML:
  - WPF: a UI framework for creating desktop client applications.
  - XAML: a declarative language that's based on XML, used extensively to build UX.
- References:
  - WPF: What is WPF? - Visual Studio (Windows) | Microsoft Docs
  - XAML: XAML overview - Visual Studio (Windows) | Microsoft Docs

UI Programming Design Patterns

- Various UX design patterns:
  - Model-View-Controller (MVC)
  - Model-View-Presenter (MVP)
  - Model-View-ViewModel (MVVM)

- References:
  - MVC: Overview of ASP.NET Core MVC | Microsoft Docs
  - MVP: Design Patterns: Model View Presenter | Microsoft Docs
  - MVVM: The Model-View-ViewModel Pattern - Xamarin | Microsoft Docs

Model-View-ViewModel

- Advantages:
  - Business logic separated from the UI layout. Connected via the ViewModel adapter.
  - The ViewModel and the Model can be unit tested independent of the View.
  - The View can be remodelled independently.
  - Design and development can go in parallel, and mostly independent of each other.
- Disadvantages:
  - Overkill for smaller applications.
  - Complex data bindings can be hard to debug.
- Exercises:
  - Refer to the MVVM code sample on our Moodle page here.
- References:
  - MVVM - Writing a Testable Presentation Layer with MVVM | Microsoft Docs

UI programming fundamentals

- Core principles:
  - Keep UI thread responsive. Delegate to worker threads to unblock the UI thread.
  - Access UI elements only on the UI thread (on most UI frameworks, including WPF).

- References:
  - Keep the UI thread responsive - UWP applications | Microsoft Docs
  - Calling Synchronous Methods Asynchronously | Microsoft Docs

Q & A