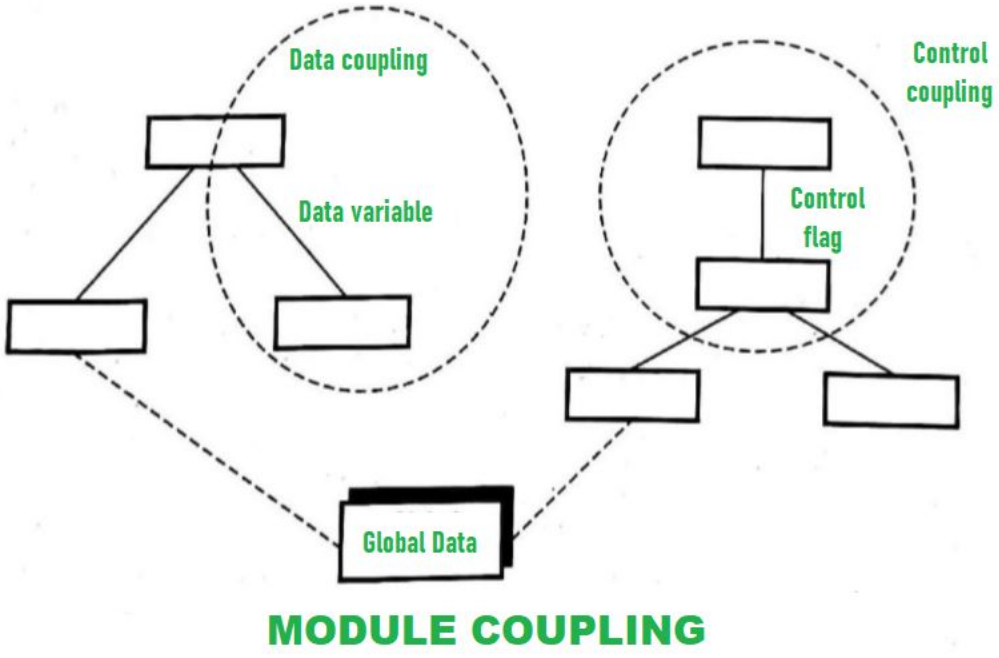
Modular programming



Modular programming

Program Control Module: In this category, a program is controlled by an independent module uniquely designed for this purpose only. The other programs may use the identical module with a similar name, but depending on the program, the module's content is designed variably.

Specific Task Module: In this category, a module is produced to achieve a particular task that is prevalent in several programs. Specific task modules are previously coded and examined, so it is easy to trust them to compose an extensive program efficiently. However, noticing this functionality of specific task modules, we also refer to them as foundational elements

Modules carries the functionality of the program and have a set of predefined data transmission options:

- no communication in with no communication out
- no communication in with some communication out
- some communication in with some communication out
- some communication in with no communication out

The Postit method

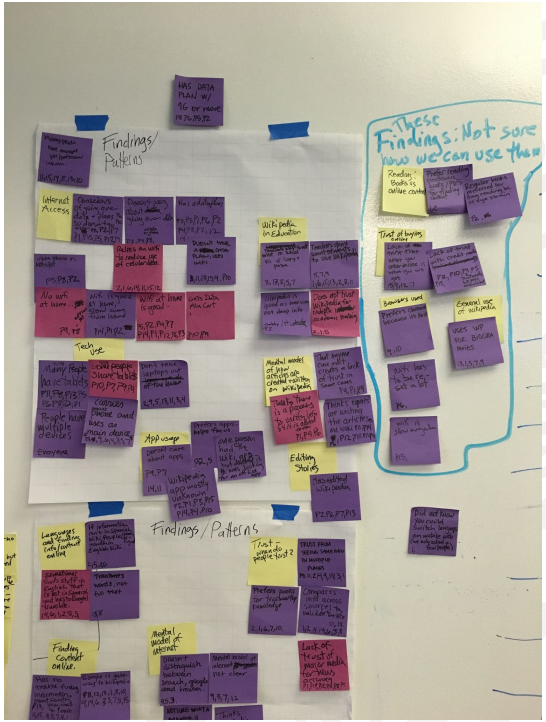
A way to discover packages

Write your Behaviours/functional requirements on a note, group similar behaviour in clusters, then once all your behaviours are sorted you can draw circles on on the white board delimiting your packages. This process can be done iteratively at any stage of the process.

If you are under FDA review remember to hide your sticky notes :-)

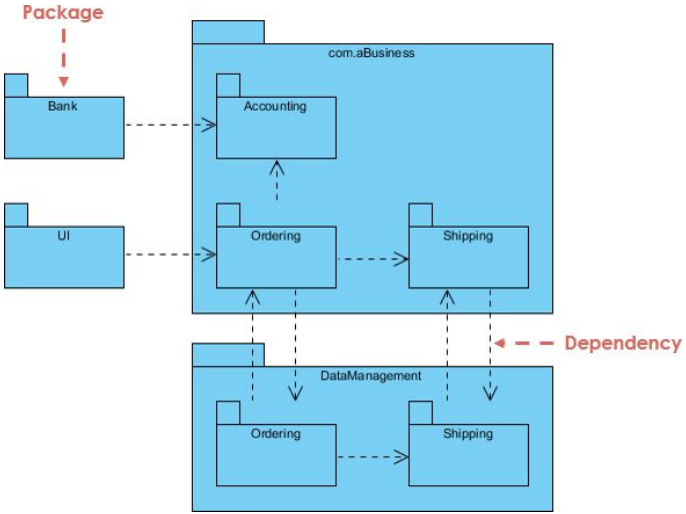
ZOOM has a white board function that lets you do this virtually and in a meeting

Example of other online whiteboard https://ideaflip.com/



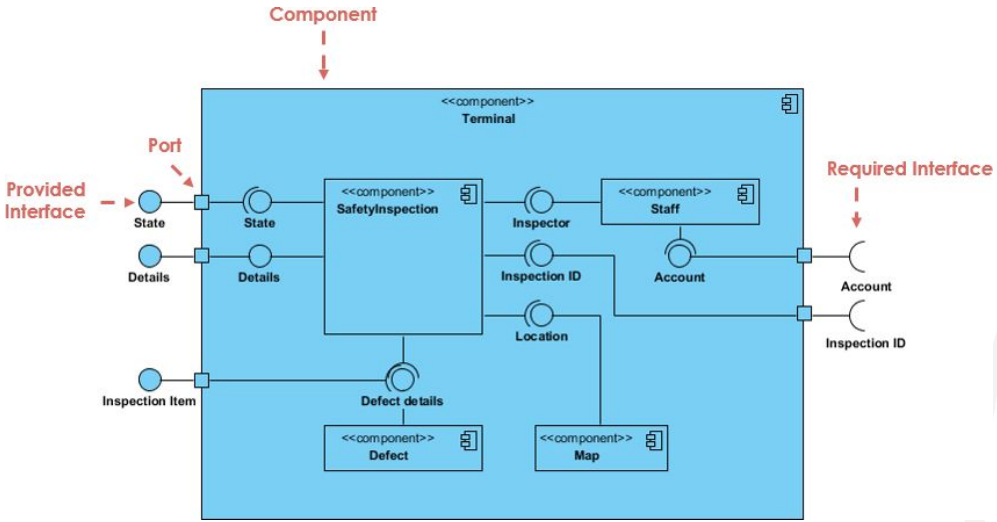
Online Zoom White board demonstration was carried out

Package Diagrams



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

Component diagrams show the deployment structure of your system



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

Packaging your module for others to use

In python the common package repository is PyPI Python Package Index accessed through PiP

https://packaging.python.org/en/latest/tutorials/packaging-projects/

In java the common way to distribute your module or as it is known i java package is in a jar file.

https://docs.oracle.com/javase/tutorial/deployment/jar/

In R packaging and submitting your R code to CRAN is the best way to distribute your package

https://cran.r-project.org/ and your package needs to comply with the repository guidelines found on the webpage

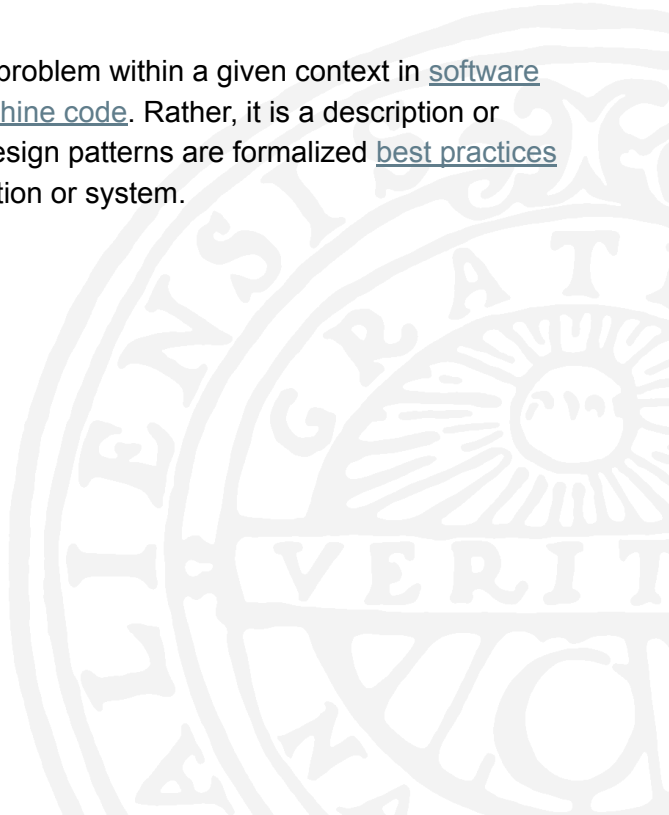
To help you create a package this book https://r-pkgs.org/ will guide you through the process.

C and c++ library creation is a bit more complex as you have static and dynamic linked libraries one such tool is https://cmake.org/cmake/help/latest/guide/tutorial/index.html

Fortran also has a more complex packaging and library structure both of the above have multitude of tools and ways of compiling libraries https://fortranwiki.org/fortran/show/Build+tools.

Design Patterns

a **software design pattern** is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.



- Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 978-0-201-63361-0.
- Brinch Hansen, Per (1995). *Studies in Computational Science: Parallel Programming Paradigms*. Prentice Hall. ISBN 978-0-13-439324-7.
- Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons. ISBN 978-0-471-95869-7.
- Beck, Kent (1997). *Smalltalk Best Practice Patterns*. Prentice Hall. ISBN 978-0134769042.
- Schmidt, Douglas C.; Stal, Michael; Rohnert, Hans; Buschmann, Frank (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. John Wiley & Sons. ISBN 978-0-471-60695-6.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0-321-12742-6.
- Hohpe, Gregor; Woolf, Bobby (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley. ISBN 978-0-321-20068-6.
- Freeman, Eric T.; Robson, Elisabeth; Bates, Bert; Sierra, Kathy (2004). *Head First Design Patterns*. O'Reilly Media. ISBN 978-0-596-00712-6.

Thank you and happy coding

The notes_paradigms.md in the repo contains the material that I have presented before and a bit more

https://github.com/UPPMAX/programming_formalism/blob/main/DevelopmentDesign/notes_paradigms.md

Now Marcus will guide us into the wonderful world of optimisation.