Refactoring

- Clients often have a large, complex software system that they have invested heavily in over the years, but no longer meets needs
- Build or purchase/customize a replacement: costly, slow, risky
- Alternative: improve the existing one, or key parts of it
- Variety of possible approaches, depending on nature of problem(s) being experienced or anticipated
- Easier on modular designs that make good use of abstraction and have loose coupling

Problem: new functionality needed

- Hopefully “easiest” of the fixes, assuming we have rights and access to source code and the expertise to attack it
- Identify the appropriate system component(s) for adding or improving the desired functionality
- Identify if new functionality can be added as a unit of its own and ‘plugged in’ to the existing component, or if it needs to be integrated into the existing component

Problem: improved performance needed

- Existing system performance no longer meets our needs (e.g. isn't fast enough, uses too much memory, etc)
- Use profiling to identify the portions of the system actually causing the performance breakdown, project what new target performance needs to be for future use
- Might need to re-write the relevant component(s), again this is easier if good design practices were used for it
- Sometimes can make use of pre-processing techniques, e.g. if bottleneck is slow processing because data reaching the component is unsorted, can we insert a sorting component just "in front of" the problematic part?

Problem: altered data/comm formats

- Systems often interact with other systems (servers, software, databases, etc) with specific data or communication formats
- If one of those other systems change, our existing system may no longer be able to communicate with it correctly
- Could alter our system’s data handling, but in complex systems that can be risky
- Alternative: add a component between our system and the one it is communicating with, to translate between them

Problem: legacy software we can’t update

- Sometimes we don’t have access or rights to source code for parts of our system, but they no longer meet our needs
- Again, could write a replacement from scratch, but can be costly/risky, particularly if we’re unsure of inner workings
- Wrappers: write software to encapsulate the legacy portion. The wrapper adds any desired functionality, communicates with legacy system appropriately, and provides us with a better interface

Mix of problems

- By the time a client gets frustrated enough to upgrade an existing system there are often multiple known problems
- Need to analyze and prioritize the mix of problems, and look for best blend of solutions to satisfy clients needs
- Typical approach is to come up with several solution approaches and debate pros/cons of each (risk, cost, time needed, degree of improvement they provide, etc)