Speaking Notes
PADM 5501
Weeks 10, 11 and 12 Notes
Dr. Neubauer

WHERE WE ARE

- Midterm has passed.
- I plan to spend three weeks on systems development.
- We will skip the Portfolio assignment. You will get credit for it.
- I need to write Discussion Forum 4A assignment soon.

**Systems Development -- Textbook Chapter 10**

It is often not possible for an organization to buy the software it needs OFF THE SHELF and going with an ERP is either not desired or not possible.
There are two major ways for an organization to obtain customized software for its needs.

- Hire consultants to create the new software.
- Have in-house programmers create the new software.

There are potential problems either way.

- You may not want to share information and data with consultants.
- You may not want to enter into a long-term working relationship with consultants.
- You may not have the necessary programming expertise on payroll.
- You may not have the budget or the long-term need to add additional programmers on payroll.

EITHER WAY, you and some of your fellow employees are likely to become involved in the ANALYSIS necessary to build a new software application.

If this is a major project, be sure your have a qualified PROJECT MANAGER. This person probably has a programming background and also has the skills to participate in BOTH ANALYSIS AND DESIGN and to manage the project.

Get some "real" END USERS involved. There are at least two good reasons to include end users.

Have a DEVELOPMENT METHODOLOGY for the entire SOFTWARE DEVELOPMENT LIFECYCLE (SDLC).

There needs to be a SPONSOR within your organization -- someone who WANTS THIS TO HAPPEN and who has the money and/or clout in the organization to help make it happen.

There should be a SYSTEMS ANALYST on board. This person UNDERSTANDS THE NEEDS OF THE ORGANIZATION (and its "business" processes) and has a clear understanding of HOW the new software is going to be used.

SPECIFICATIONS ("specs") should be written, whether or not the (major) project is to be put out for bids.

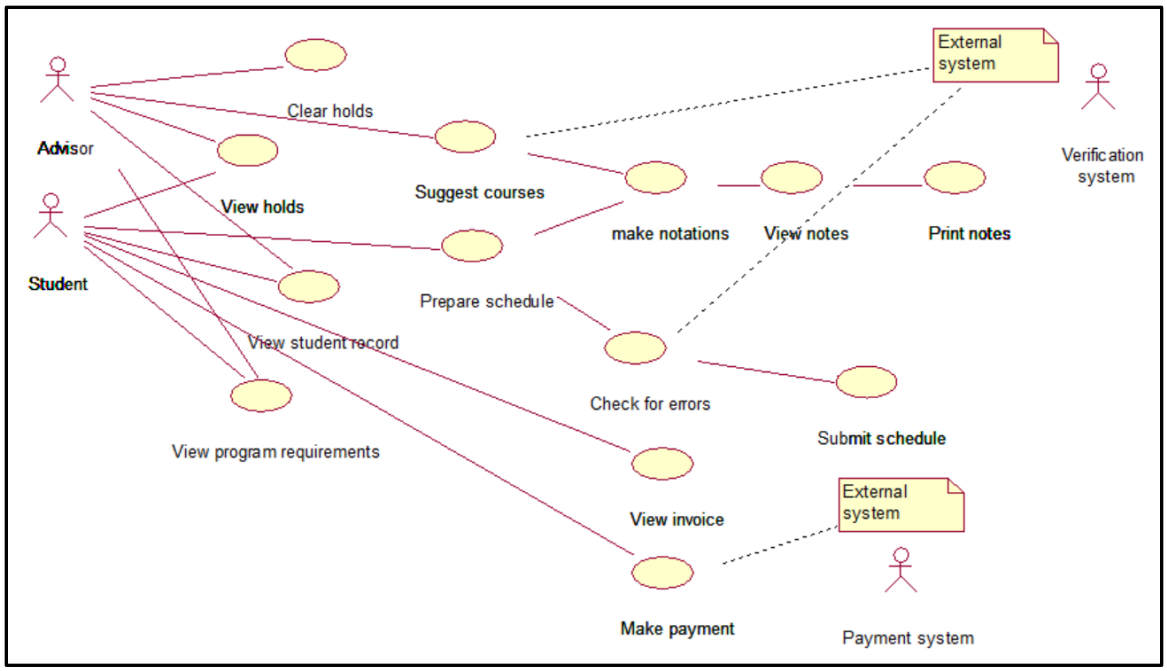
AVOID internal "power user" self development except possibly for a very simple, minor application.

DON'T GET PULLED INTO BUILDING MORE STOVEPIPE APPLICATIONS!

A BUSINESS PROCESS can be modeled visually and/or written out in detailed text. In order to design software to support a business process, it is necessary to identify the ACTORS, the USECASES each actor must be able to perform. An ACTOR is a CLASS OF USERS. A USE CASE is a functionality of the system that allows the user (usually human) to perform. A use case diagram (like the one below) is primary part of ANALYSIS. But it can also be useful in SYSTEM PROTYPING. You can begin to sketch out PROTYPE SCREENS based on the use cases. Advisors and Students do not get the same screens. Usually a particular screen includes the right BUTTONS for advisors and the right BUTTONS for students.

Logging in is not a use case (because no one uses an application because they want to log in. But the log in allows the system to know if the user is a student or an advisor, and to provide the appropriate screens.

A USE CASE diagram is a very important tool during ANALYSIS.




PROTOTYPING during JAD/RAD sessions can be very helpful in making sure everyone "is on the same page." But be sure everyone understands that the PROTYPE is not the real thing. It is a mock up of interface screens. The real work is done later in DESIGN and IMPLEMENTATION. A prototype is, "just a pretty interface."

"SDLC" stands for, "software development lifecycle." It begins, well, at the beginning and, technically, continues so long as the application is being MAINTAINED. The most expensive part is maintenance. "Cutting corners" early on is not a good idea. THE PAIN ENDURES. Early mistakes/misunderstandings can be VERY, VERY COSTLY.

A major project is likely to take 12 to 18 months or longer. The problem is, NEEDS CHANGE almost continuously.

One approach is called the WATERFALL METHODOLOGY. The "spirit" of this approach is, "no going back." The major problem is that it is not agile enough and it delays testing until all the code is written, which is VERY RISKY.

IDENTIFICATION OF A NEED
ANALYSIS OF THE NEEDS
DESIGN OF THE SOLUTION
IMPLEMENTATION (WRITING CODE)
TESTING
DEPLOYMENT
MAINTENANCE

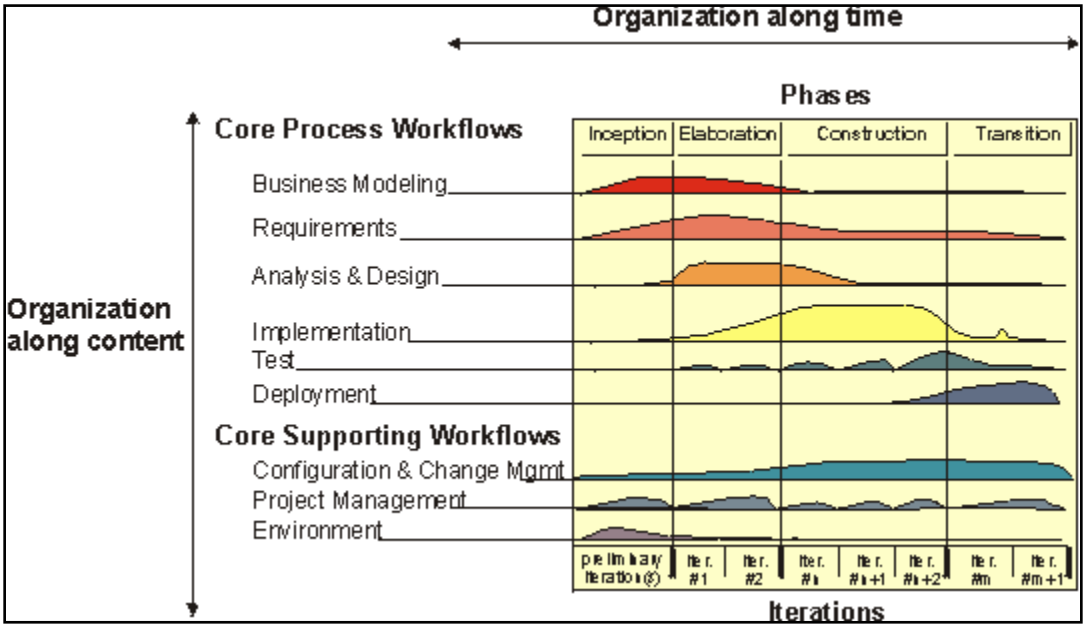
MAINTENANCE includes "bug fixes" and addition of features as it becomes necessary to add features. MAINTENANCE IS THE MOST COSTLY PART OF IT!

There are two major problems associated with the actual use of this approach to the SDLC.

1) During a long project the needs of the organization are changing. It is not acceptable to be unable to accept CHANGE ORDERS until after deployment of the new system.

2) It is VERY RISKY to write all the source code before beginning to test it.

3) Also, you may have programmers sitting around doing nothing because they must wait until others do their part first.




Representation of the Iterative-Incremental Methodology
source: Rational Software Corporation (now part of IBM Corporation)

This is the MORE MODERN and BETTER approach and is possible in large part because of the OBJECT-ORIENTED programming paradigm.

PROGRESS is measured by the completion of PHASES which are broken down into multiple INTERATIONS. There is a MILESTONE at the completion of every PHRASE.

At the end of every iteration you should REFACTOR all the artifacts of the project.

Your development team can be doing multiple kinds of work at the same time. It is not necessary to finish one kind of work before going forward to the next.

You can accommodate (reasonable) CHANGE REQUESTS while still in development.

You start writing code and testing code quite early along the way.

You can begin to deploy in BETA while still completing the programming and testing.

ON THE FINAL EXAM I will ask you to interpret what Core Process Workflows are being done in which project Phases.

The spirit of the Iterative-Incremental approach is WE CAN DO MORE THAN ONE THING AT A TIME. At the end of every phase you REFACTOR everything to be sure you are ready to go forward. Refactoring is basically "getting your ducks in a row" so you can move forward to the next phase of the project.

Things to notice about the iterative-incremental model of the SDLC:

- Testing begins "early and often." As soon as there is code written it is being tested. It is the MODULAR nature of object-oriented programming that makes this possible.
- Analysis and design are combined activities and continue well into the construction phase.
- Deployment begins while code is still being written! This is sometimes called Beta testing. This may not be a good idea. Make sure you and the project manager understand each other in this regard.
- This approach may look sloppy but it is not. It requires AN EXPERIENCED SOFTWARE PROJECT MANAGER. If you have such a person, treat him or her well. They are not easy to find and can make a lot more money in the private sector.

Joint Application Development (JAD)
http://en.wikipedia.org/wiki/Joint_application_design

Rapid Application Development (RAD)
http://en.wikipedia.org/wiki/Rapid_application_development

Iterative Development
http://en.wikipedia.org/wiki/Iterative_and_incremental_development

Software Prototyping
http://en.wikipedia.org/wiki/Software_prototyping

The Iterative-incremental model of the SDLC described above is an example of iterative development. You may also hear reference to the spiral model, which is basically the same idea.

JAD and RAD refer to getting all major stakeholders into the same room at the same time and using a computer and projector to model the new system and make sure everyone is

"on the same page." Hopefully, this saves time and avoids misunderstandings. It is good to start with Use Case models and work on from there.

Prototyping refers to projecting screens of the new system and designing the INTERFACE together in a JAD/RAD session. The important point is that THE PROTOTYPE IS NOT THE REAL THING. It is "just a pretty interface." All the real work has been stubbed out. It may look like the real thing. It may even appear to FUNCTION like the real thing. BUT IT IS NOT. The risk is that an inexperienced person may get the idea that "we are almost there." I think the prototype should be a THROW AWAY PROTOTYPE. In other words, all the REAL WORK comes later. The prototype is just for TALKING (AGREEMENT) PURPOSES. The code should not be considered application source code.

CONCLUSION

Large organizations may maintain their own staff of professional programmers. Small organizations generally depend upon consultants.

THE BIDDING PROCESS serves to separate ANALYSIS from DESIGN and for that reason can be problematic.

A CEO is not usually a PROJECT MANAGER. A CEO should be thinking about the future strategically -- not writing code or managing specific projects. A PROJECT MANAGER is a technical person who knows how to get programmers to work together, which is no small task given the complexity of code. OBJECT-ORIENTATED programming languages help because the code is MODULAR and LOOSELY COUPLED (if the project is well designed).

A good application is WELL DOCUMENTED. It has both good USER DOCUMENTATION and good TECHNICAL DOCUMENTATION.

A good application is DESIGNED WITH AN EYE TOWARD MAINTENANCE. Good programmers don't engage is "slick coding tricks." They write for CLARITY. You don't want an application that is so FRAGILE that in order to modify it or add a new feature you have to trash it and start over again.