SSW-555
Lecture 1

Software Development > Coding
Although programming of software is an important part, it usually is 1/4 of the total effort.
Most software project failures are the result of:
- Poor planning
- Inadequate understanding of requirements
- Inadequate attention to quality
- Failure to respond to problems until too late

Should the project be done?
What needs to be done?
Who does what and when?
Creating the software?
Feasibility/profitability, requirements, planning and controlling, implementation, delivery and maintenance, support, and teamwork
^^All important on projects.
Examples to Consider
Some software systems are so important that without it there could be a loss of life.
This could be say for Boeing, for the military, etc. all of these impact life. It is important that testing and quality control is up to par for these softwares.
Nobody is going to die from the running app for the apple watch going down.
Feasibility and Profitability
What is the market? Who will pay for and use the system? How much will they pay?
How expensive will the project be? What are the risks?
Requirements
An extensive process. What features should be produced? What are the non-functional requirements? Who knows what is needed?

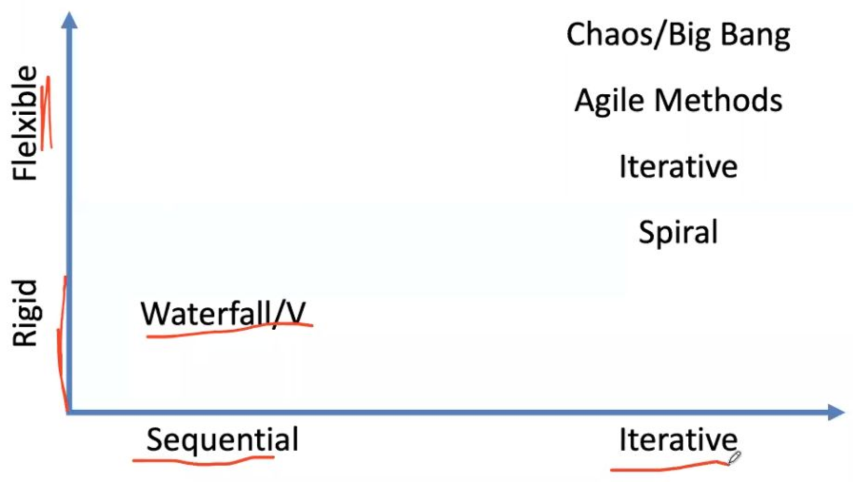
> **Medical system non-functional requirements:** must be available 24/7; patient sensitive information must be protected; run-time requirements, need to work with clients and users. This is not something you rush to market!!
> Compare this to the running app where you should deploy as soon as possible.

Planning and Controlling
Long term planning: release schedule and lifetime of service or product
Project Planning: who does what, relationship and communication with stakeholders, scheduling of tasks
Implementation
High level and low level design; architecture; programming, verification and validation (what do you mean by google maps? You want it to be a call? What exactly do you want me to do?)

Delivery and maintenance
Who is keeping this product alive? There needs to be support tools for this if it goes down.
Teamwork
Software development is not a solitary activity. Communication between team members is essential.

Introduction Extreme Programming/XP

What is the right level of planning for software projects? It depends on the task! How do we decide, what is the domain, how complete are the requirements, how stable are the requirements, what is the cost of doing the wrong thing, what is the cost of doing the right thing too slowly? Etc, etc
Software Development Life Cycle
Software specification, software development, software validation, and software evolution. The validation piece is more difficult than you think.



Waterfall is good for a very complex system that does not necessarily need to change. Take a telephone system.
Waterfall plans everything in sufficient detail so we can get it right the first time. This is a very formal process with extensive documentation and a serial execution.
Must get it right the first time as the cost of failure is very high.
You can add verification and validation in the Waterfall model in which every level is validated and verified. You cannot step back up from this process it is a very linear process.
Boehm's Spiral Model
Recognizes the limitations of waterfall method and adds focus on risk assessments. This encourages incremental development and iterations. You learn from previous iterations and

each spiral includes **Objective setting, risk assessment and reduction, development and validation, and planning**
This allows for constant change and tries to break it into simpler processes.
Requirements are mostly stable but may change.

Agile SDLC
Frequent iterations and deliverables.
Close collaboration between customers and developers.
For the waterfall method, the customer is only engaged during requirements.
Customer is a critical partner in the process rather than an observer.
Frequent reflection and continuous improvement

Big Bang/Chaos SDLC
Little to no planning, figure it out as you go, typically used for very small projects and not recommended

Software development before Agile
1960s since then we have worked on improving and developing on waterfall and agile. In the 1990s there was the dot com boom and the internet time. There was a backlash to the huge process.
Rational Unified Process: Developed iteratively, managed requirements, use component-based architectures, UML, continuously verify software quality, etc.

> Helped develop software iteratively because solutions are too complex to get right in one pass. Use an iterative approach and focus on the highest risk items in each pass, customer involvement, accommodate changes in requirements. Use cases and scenarios help with this as well.

Unified modeling language: UML
Verify the software quality: focus on reliability, functionality, and performance.
Control changes to software: change is inevitable; actively manage the change request process, control, track and monitor changes
RUP project lifecycle phases
Inception, elaboration, construction, transition
Boehm’s risk exposure profile: how much planning is necessary?

Need to get to market quickly

.oss

;

hare

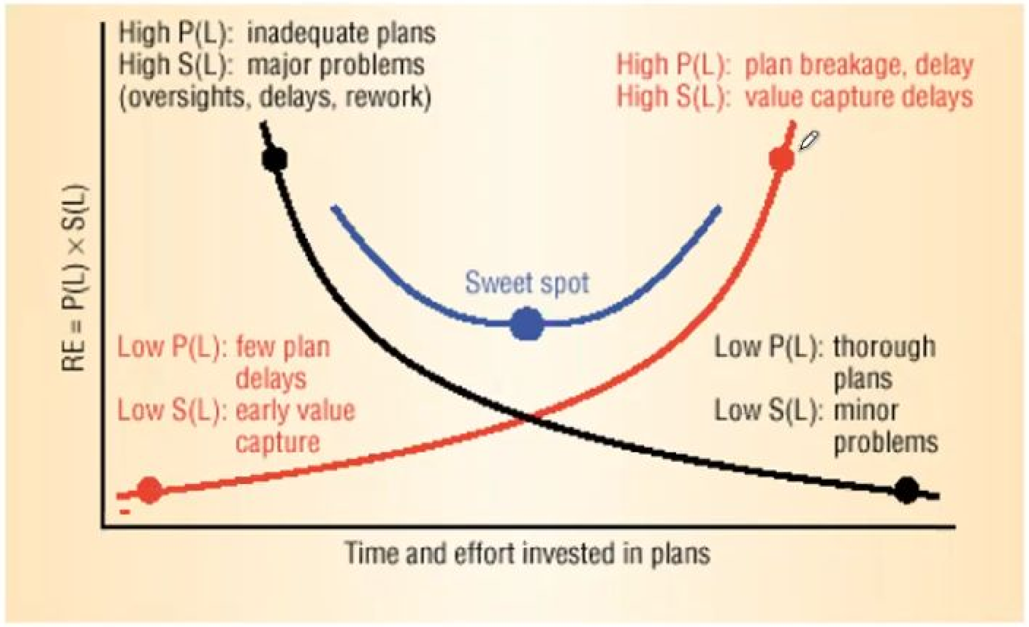


Figure 2. Risk exposure (RE) profile. This planning detail for a sample e-services company shows the probability of loss P(L) and size of loss S(L) for several significant factors.

Source: "Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

Agile Manifesto:

Working software is more important than documentation

12 Principles of Agile Manifesto

Early and continuous delivery

Welcome changing requirements

Working software frequently

Business people and dev people must work together on a daily basis

Build projects around motivated individuals

Most efficient and effective method of conveying information

Working software is primary measure of progress

Sustainable development

COntinuous attention to technical excellence and good design

Simplicity -- the art of maximizing the amount of work not done is essential

Build our own teams (self-organizing teams)

At regular intervals, we reflect!!

Extreme Programming(XP)

Kent Beck created it

12 practices
The planning game: business people decide scope, priority and release dates
Technical people decide estimates of effort, consequences, process and detailed scheduling
Small releases: small as possible
Every release must be tested and be able to run
Metaphor is a simple explanation of the project
SImple design: runs all the tests, has no duplicated logic, states every intention important to programmers
Any feature without an automated test does not exist
Programmers need confidence in correct operation
Customers need confidence in correct operation