50.003: Elements of Software Construction

Week 1

Software Development: Life Cycle and Methodologies

Plan for the Week

- Overview of Software Engineering and Software Development Processes

- More on Software Development Processes

Software Engineering

User Requirements

↓

the magical programming machine

↓

System Implementation

***The synthesis problem (i.e., synthesizing a program from a specification automatically) is undecidable (i.e., there doesn't exist an algorithm which could solve the problem in finite time).

Software Engineering

User Requirements

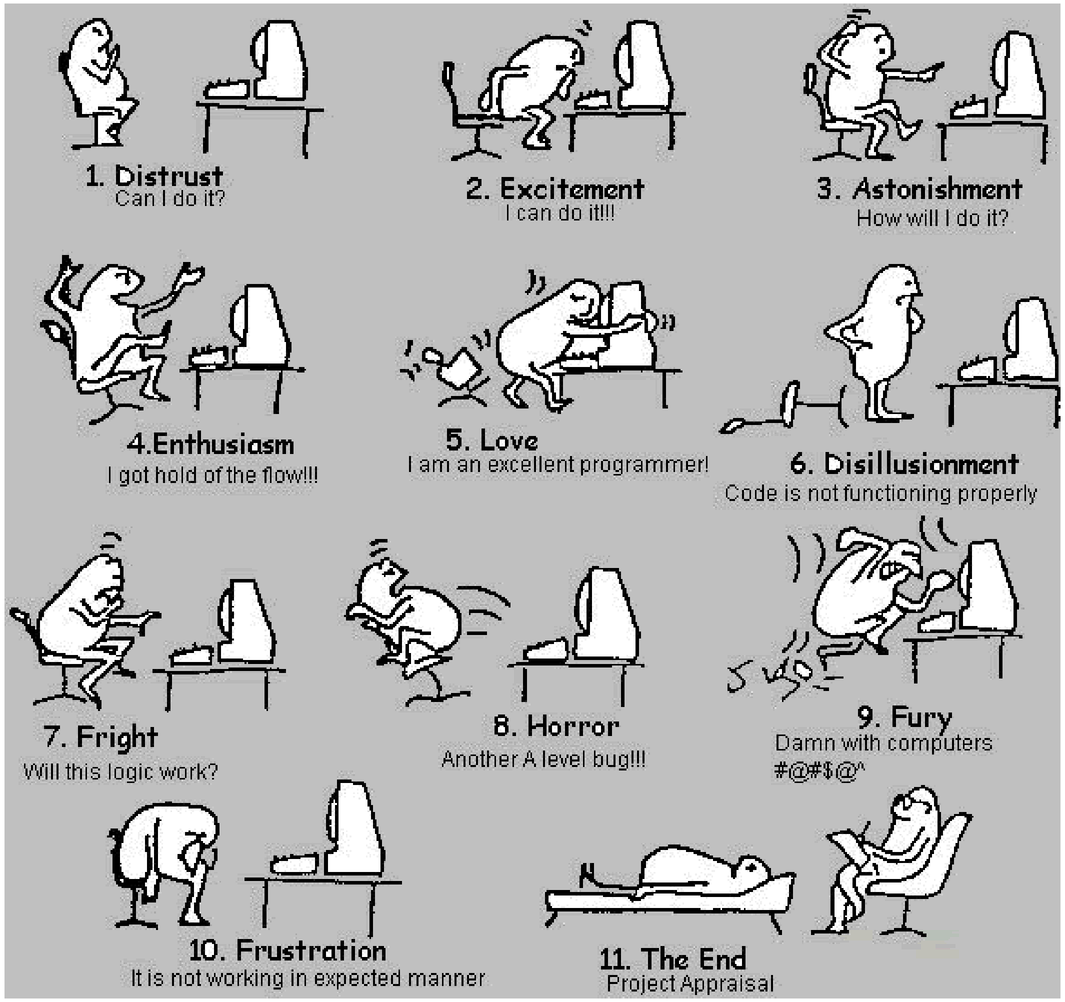
$\downarrow$

The species we called programmers

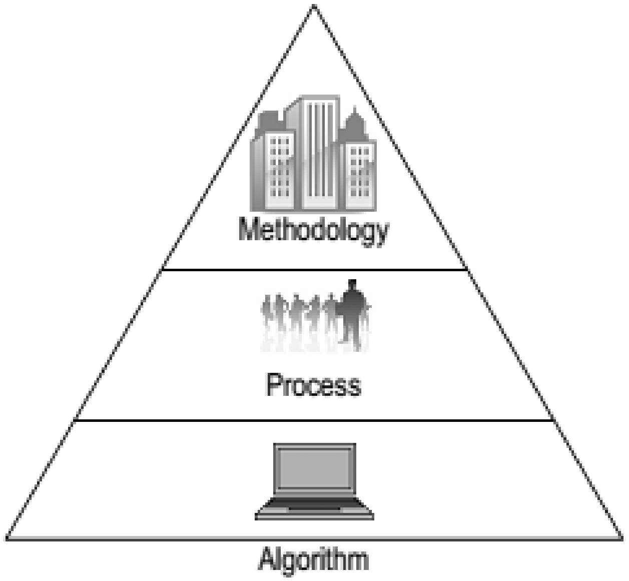
$\downarrow$

System Implementation

A Programmer’s Life



More Than Just Coding Skills



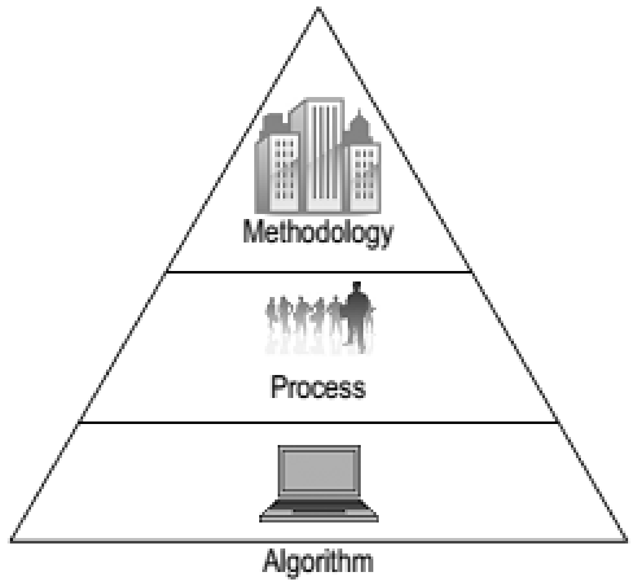
where we are at

Coding Skills

The Big Questions

- Why are we worried about methodologies?
- What is the software development life cycle (SDLC)?
- What are the different software development methodologies?
- How to choose one methodology over another?

More Than Just Coding Skills



Coding Skills

where we are at

Methodologies

- Code-a-bit-test-a-bit (CABTAB)
- Waterfall
- Rapid prototyping
- Iterative and incremental
- Agile methods

CABTAB

- Code-a-bit-test-a-bit (CABTAB) is hardly recognized as a methodology, although it is widely used in programming.
- It is unsuitable for anything other than very small systems of limited scope.

We need a process!

Software development life cycle (SDLC)

The SDLC is the sequence of activities covering

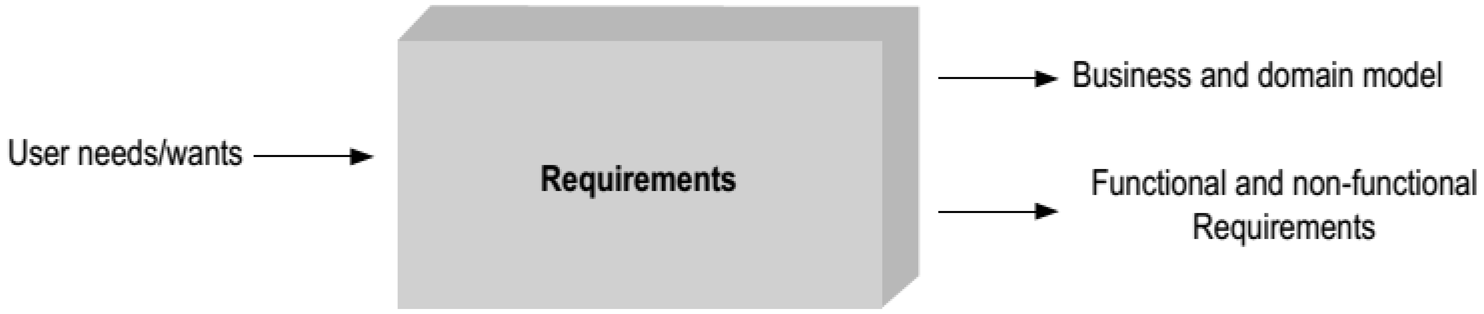
- *requirements*
- *analysis*
- *design*
- *implementation*
- *testing*

over which a software system is developed.

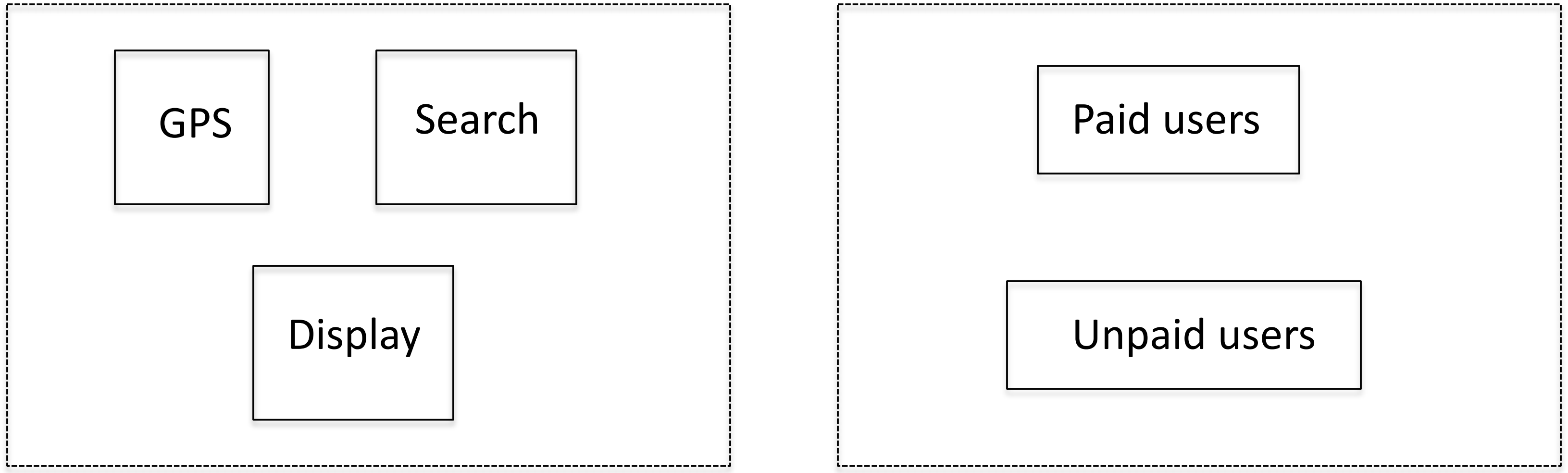
Requirements

- During the requirements workflow, the primary activities include
  - Listing candidate requirements *(very informal)*
  - Understanding the system context through domain modelling and business modelling
  - Capturing functional as well as non-functional Requirements

Requirements


User needs/wants
Requirements
Business and domain model
Functional and non-functional Requirements

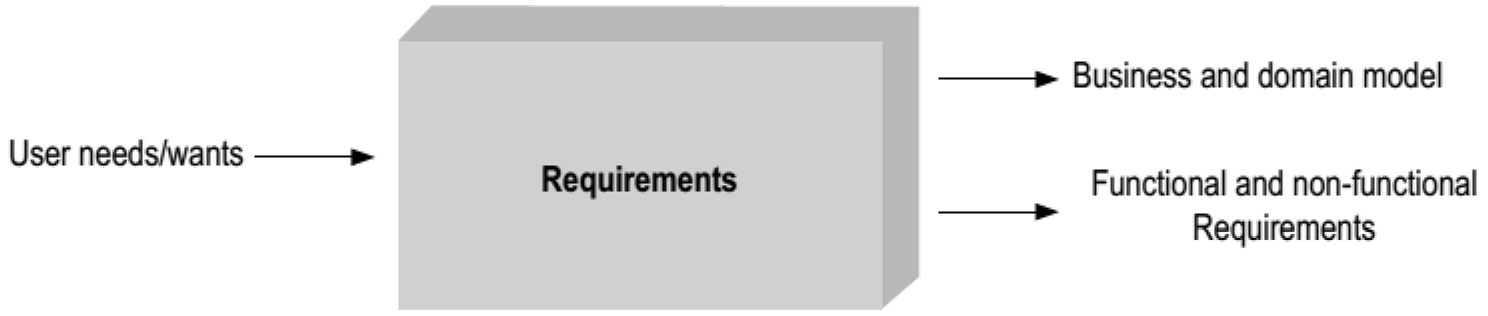
Android App: Show nearby restaurants



Domain

Business

Requirements


User needs/wants
Requirements
Business and domain model
Functional and non-functional Requirements

A functional requirement is one on the **functionality** of the system, e.g., *"the system must book movie ticket if the user has paid for it"*.

A non-functional requirement might be on the **performance** of the system, e.g., *"the system must book the ticket within 30 seconds"*.

Requirements

- Requirements should be captured in the language of the user. *user here is customer*
  - Use cases help distil the essence of requirements as sets of action-response transactions between the user and the system (e.g., as user cases).
- Example:
  - *Action <Put $1 -> Press Button>, Response <Dispense Coffee>*
  - *Action <Press Button>, Response <Nothing>*

Look at the interaction where you put coin first in one case and in the other case, you just want to know whether things are available. Because when you talk with the customer, he himself does not know everything that she wants. A key job of the developer is to help customer find out what she wants.

Analysis

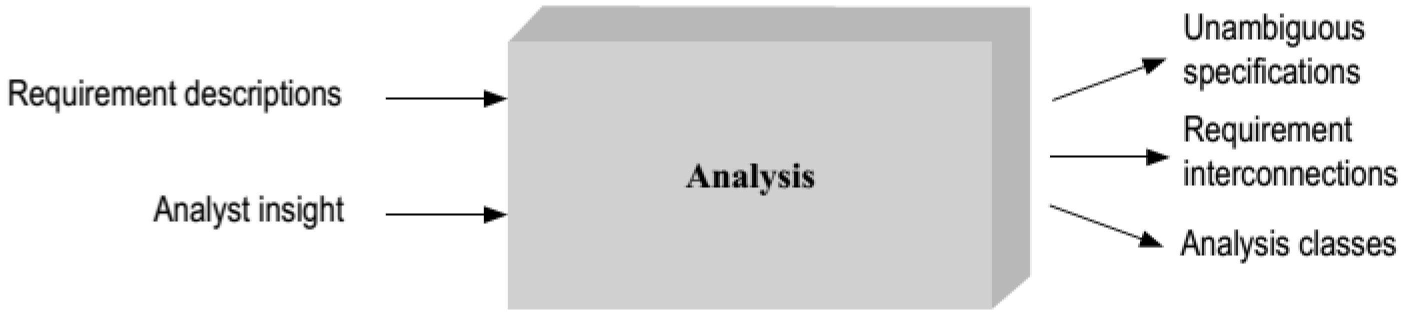
- A key theme of the analysis workflow is to understand how and where requirements interact and what it means for the system.
- In an ideal world:
  - *Action <Press "a">, Response <Do "this">*
  - *Action <Press "b">, Response <Do "that">*
- In real world:
  - *Action <Put $1 -> Press "tea">, Response <Dispense "tea">*
  - *Action <Press [any]>, Response <Display "No stock" if tea is finished>*

How many tickets can you book at a time?
What if the machine does not have enough tickets?

Analysis

- Analysis also involves
  - Detecting and removing ambiguities and inconsistencies amongst requirements
    - *Requirement 1: "the system must book movie ticket if the user has paid for it"*
    - *Requirement 2: "the system must book tickets within 30 seconds"*
  - Developing an internal view of the system
  - Identifying the analysis classes and their collaborations
    - Analysis classes are preliminary placeholders of functionality

Analysis


Requirement descriptions
Analyst insight
Analysis
Unambiguous specifications
Requirement interconnections
Analysis classes

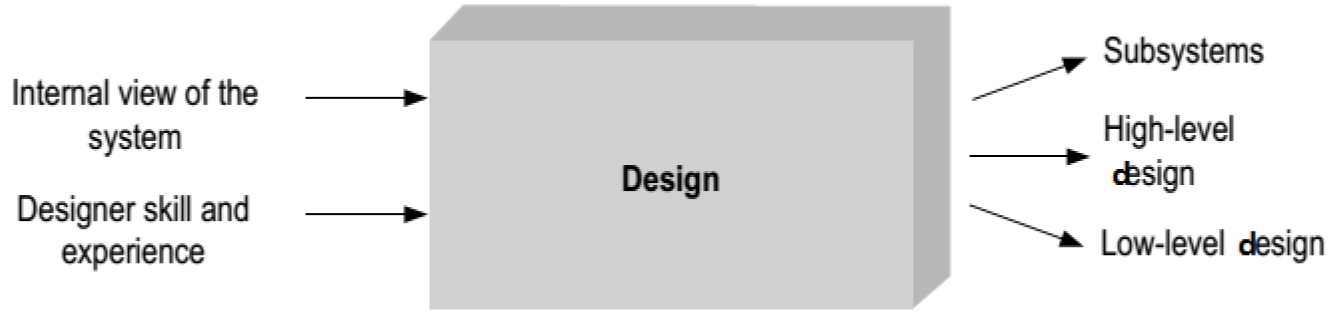
Design

- Deciding on the collaboration between components lies at the heart of software design.
  - A component fulfils its own responsibility through the code it contains.
  - A component exchanges information by calling methods on other components, or when other components call its own methods.

Explain high-level design and low-level design. Highlight that low-level design is kind of coding and implementation.

Design

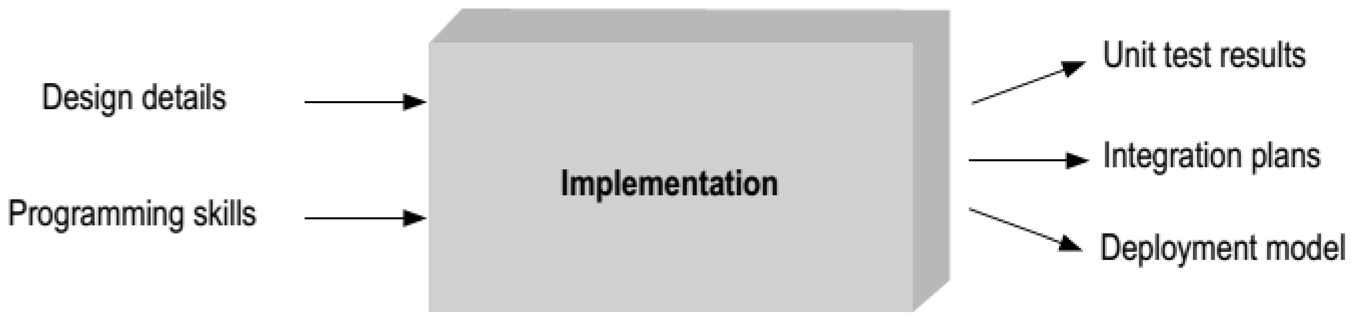
- The design workflow involves
  - Considering specific technologies (e.g. which language and platform?)
  - Decomposing the system into implementation units



Implementation

- A large part of implementation is programming.
- Implementation also involves
  - Unit testing
  - Planning system integrations (consider cases where the system communicates with external hardware)
  - Devising the deployment model (how do you ship: binaries, libraries etc.)

Implementation

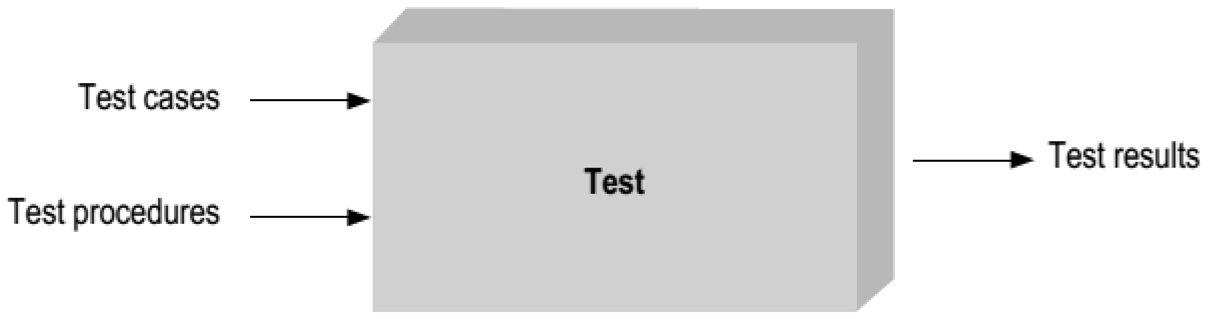


Testing

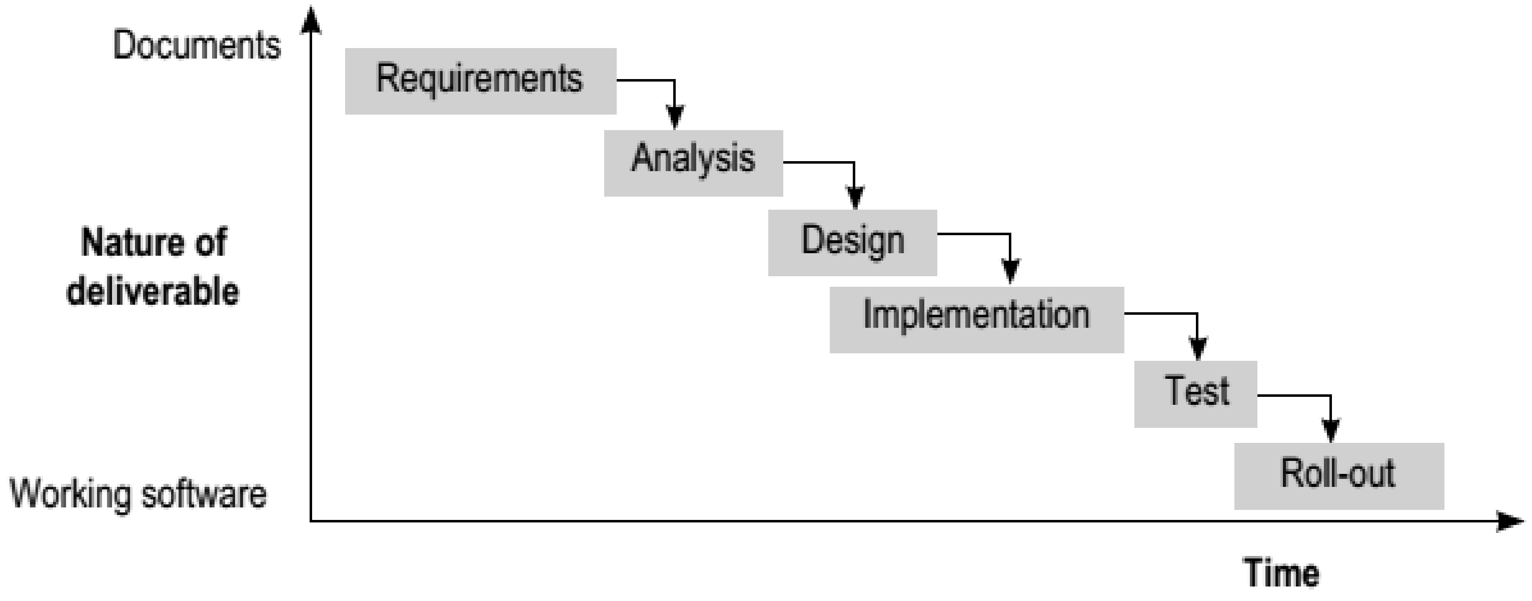
- The primary activities of the test workflow include
  - Creating test cases (manual or automatic),
  - Running test procedures, and analysing test results.
- Due to its very nature, testing is never complete.

Test

Test results: pass or fail.



Waterfall model



Explain that we have discussed different stages of software development in isolation, but we have not discussed how this stages interact among them. Now we will describe a few standard models that capture the relationship between different models.

Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Waterfall model

- The Waterfall model uses the metaphor of falling water to underline the sequential nature of software development.
- It is suited to projects of limited uncertainty and risks.

Waterfall Model

https://www.youtube.com/watch?v=0NAsk0noT_U

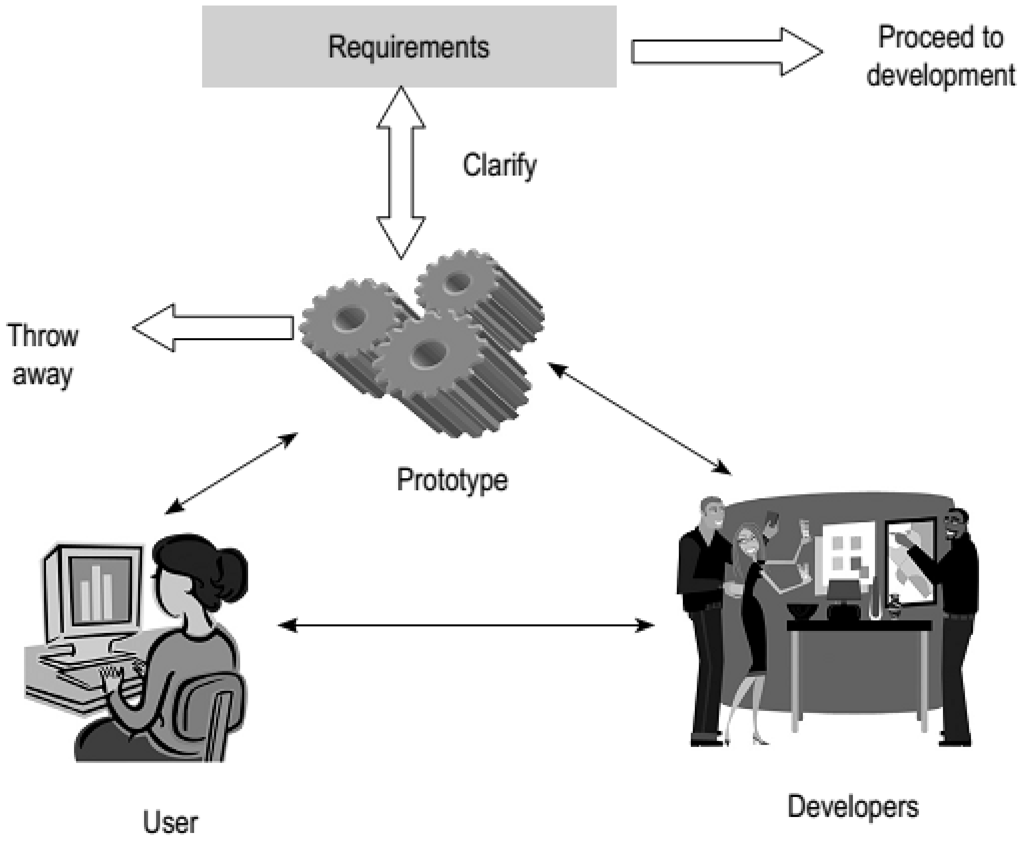
Cohort Exercise I (10 min)

In his 2000 Turing lecture ‘The Design of Design’, Frederick Brooks said, ‘The Waterfall Model is Dead Wrong’. I suppose you would agree to certain extent.

The waterfall model assumes that the different stages of software development are sequential. Discuss with your group on the actual relationship between the 5 activities in SDLC + the stage of maintenance.

Prototype gives an initial idea to the user how the product will look like. It has to be built fast in order to fix any ambiguities in the requirement. This way rapid prototype can avoid errors and ambiguities that appear much later in the software development process and allow opportunities to fix them quickly. Prototype should only include core functionalities, other extra functionality or non-functional requirement such as security and performance can be ignored.

Rapid Prototyping



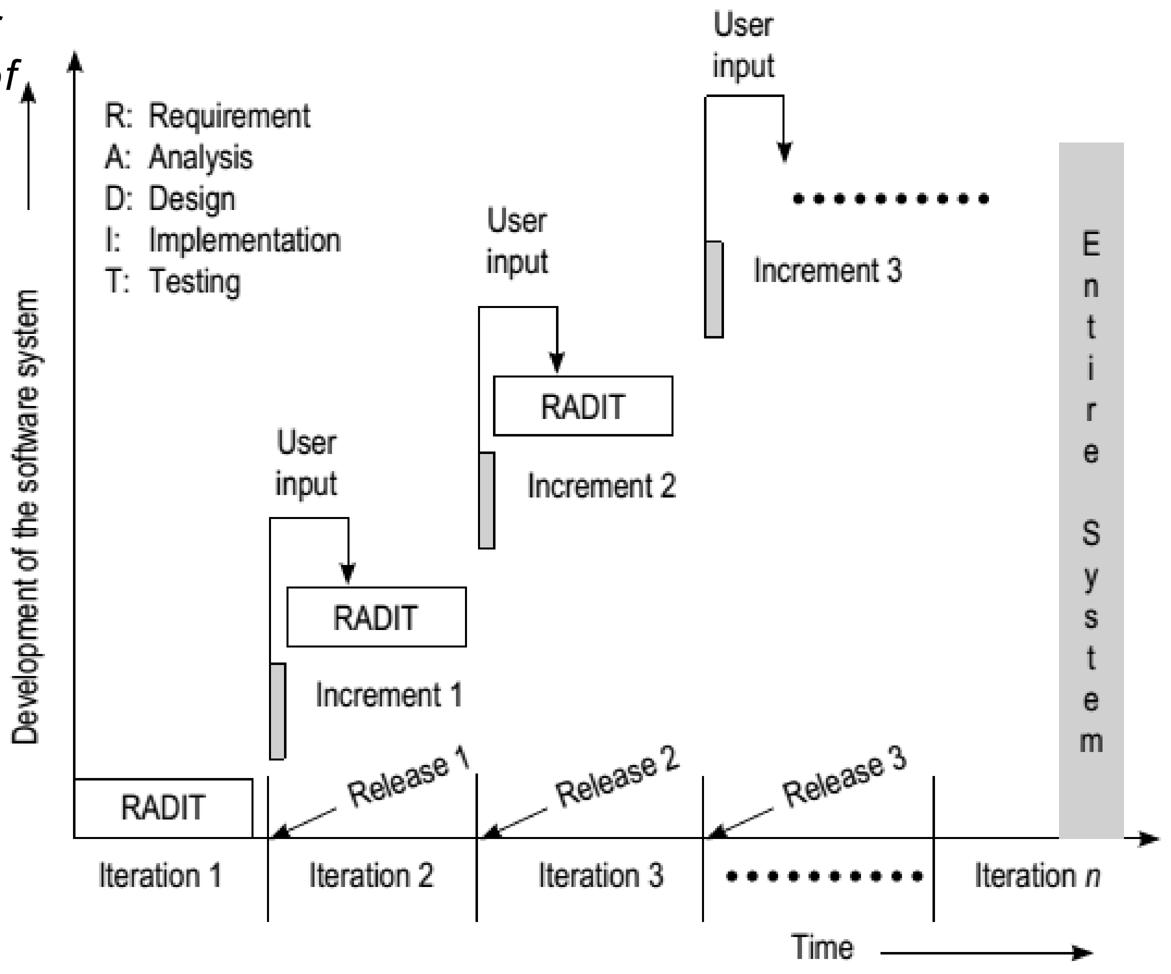
Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Rapid Prototyping

- Rapid prototyping recommends the building of prototypes to clarify requirements and system scope.
- The prototypes, however, should never become the final system.
- Once the requirements have been sort out, the system is built formally
- *Timing is crucial for the prototype*

Iterative and Incremental Development

Rapid Prototyping can use waterfall. Throw away prototype after testing stage of first waterfall.



Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Iterative and Incremental Development

- In iterative and incremental development, the software system is built through a series of time-boxed development cycles—iterations—leading to tangible and testable additions to the overall system functionality—increments.
  - This is an expedient model for building systems with initial ambiguity of scope and changing requirements.

Note the difference with rapid prototyping. Here we do not throw away the previous incremental developments.

***rapid** -> proof of concept, dont build on top of it*

***iterative** -> work towards the final product, dont throw stuff, build more features across time*

The Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over Processes and tools
Working software over Comprehensive documentation
Customer collaboration over Contract negotiation
Responding to change over Following a plan

That is, while there is value in the items on the right, we value the items on the left more"

http://agilemanifesto.org/ February 2001

12 Agile Principles

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Self-organizing teams
- Regular adaptation to changing circumstance

Cohort Exercise 2 (5 min)

You have a sense of the course project by now. If you don’t, please make sure you do ☺

Read the Agile manifesto and principles and discuss with your team on what it means in terms of developing your project.

What is the Agile Method?

https://www.youtube.com/watch?v=-zDct5d2smY

Cohort Exercise 3 (10 min)

You have a sense of the course project by now. If you don't, please make sure you do ☺

On the basis of what we have discussed so far, which methodology do you think would be suitable for developing your class project? Agree on some timeline on some milestones.

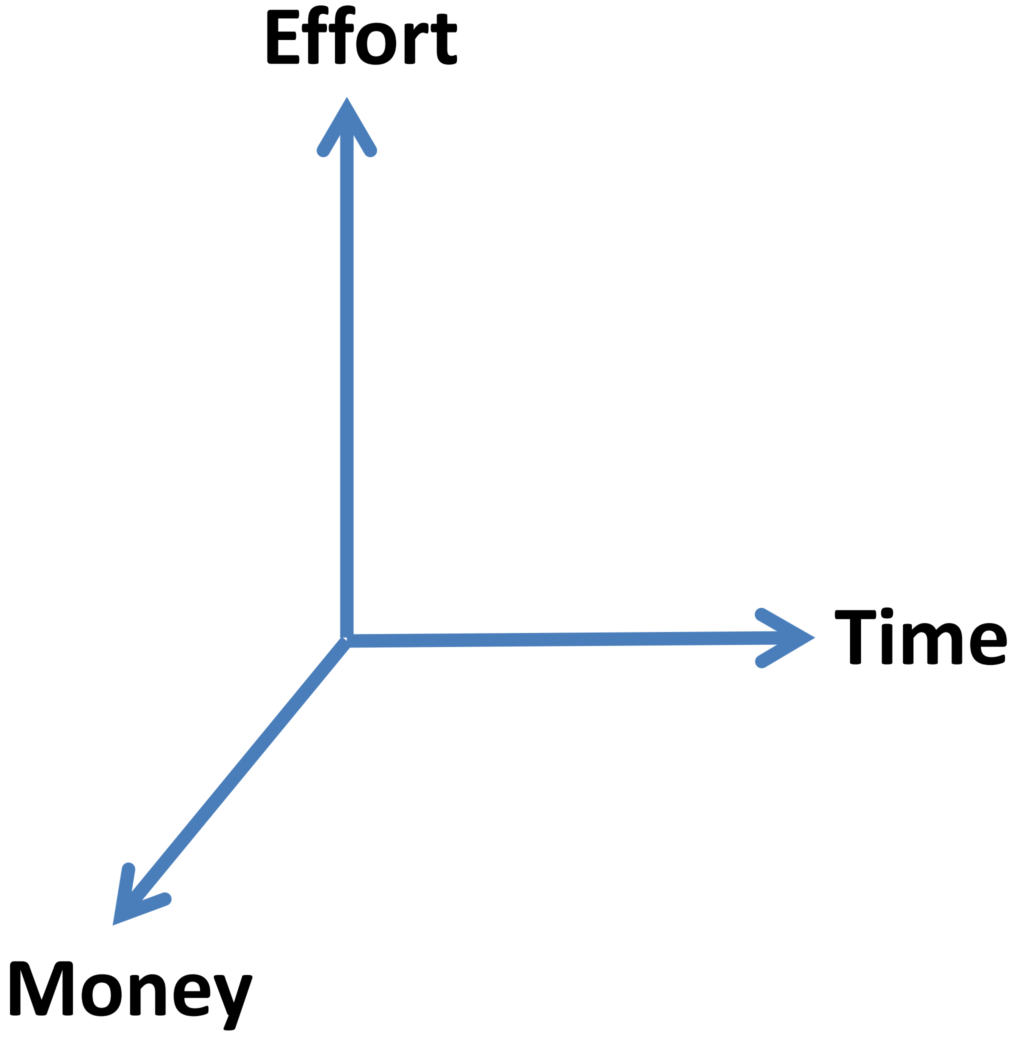
In terms of timeline, how we know how much effort and time a software development project needs?

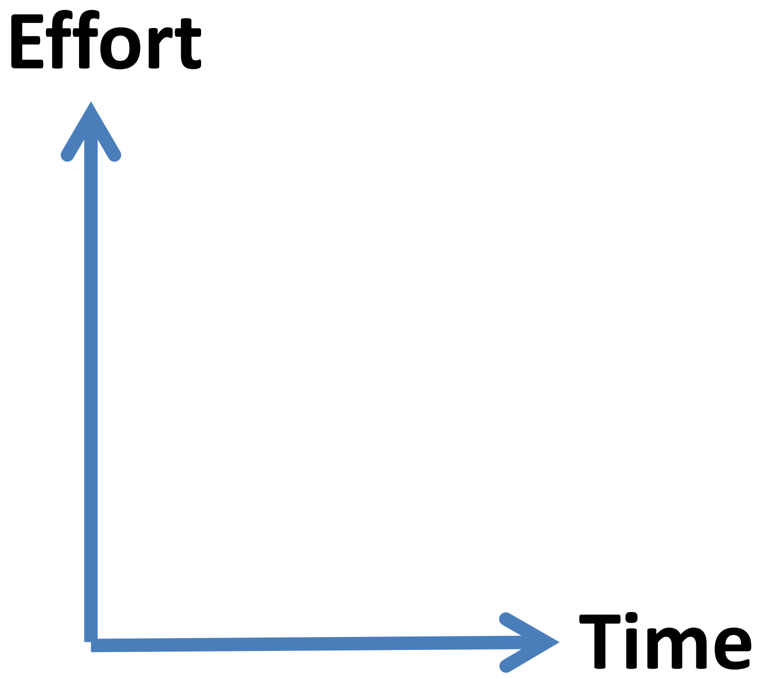
Project

Tom Cargill

The first 90% of the code accounts for the first 90% of the development time.

The remaining 10% of the code accounts for the other 90% of the development time.


Effort
Time
Money


Effort
Time

The Software Equation

$$(B^{1/3} * \text{Size}) / \text{Productivity} = \text{Effort}^{1/3} * \text{Time}^{4/3}$$

B = Special skill factor
Productivity = a productivity parameter (depends on team)
Effort = Human effort in years or months
Time = Time to complete project in years or months

The models basis was formed through analysis of productivity data collected from over 4000 modern day software development projects.[1]

The software equation was derived from the Putnam-Norden-Rayleigh Curve which can be used to show the non-linear correlation between time to complete the project and applied human effort.[2]

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

Depends on you and your team

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

Depends on the scale of your project

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

Inverse relationship

Solving for effort

$$\text{Effort} = B * (\text{Size}/(\text{Productivity} * \text{Time}^{4/3}))^3$$

- Effort: project effort measured in person-months or person-years
- Size: lines of code estimate for the project
- Time: length of project measured in months or years
- B: a special skills factor
- Productivity: a productivity parameter (depending on your team)

Defining B

- B, the special skills factor, is related to the size of the product.

B Value	Size of Project
0.16	5-15K
0.18	20K
0.28	30K
0.34	40K
0.37	50K
0.39	> 50K

Defining Productivity

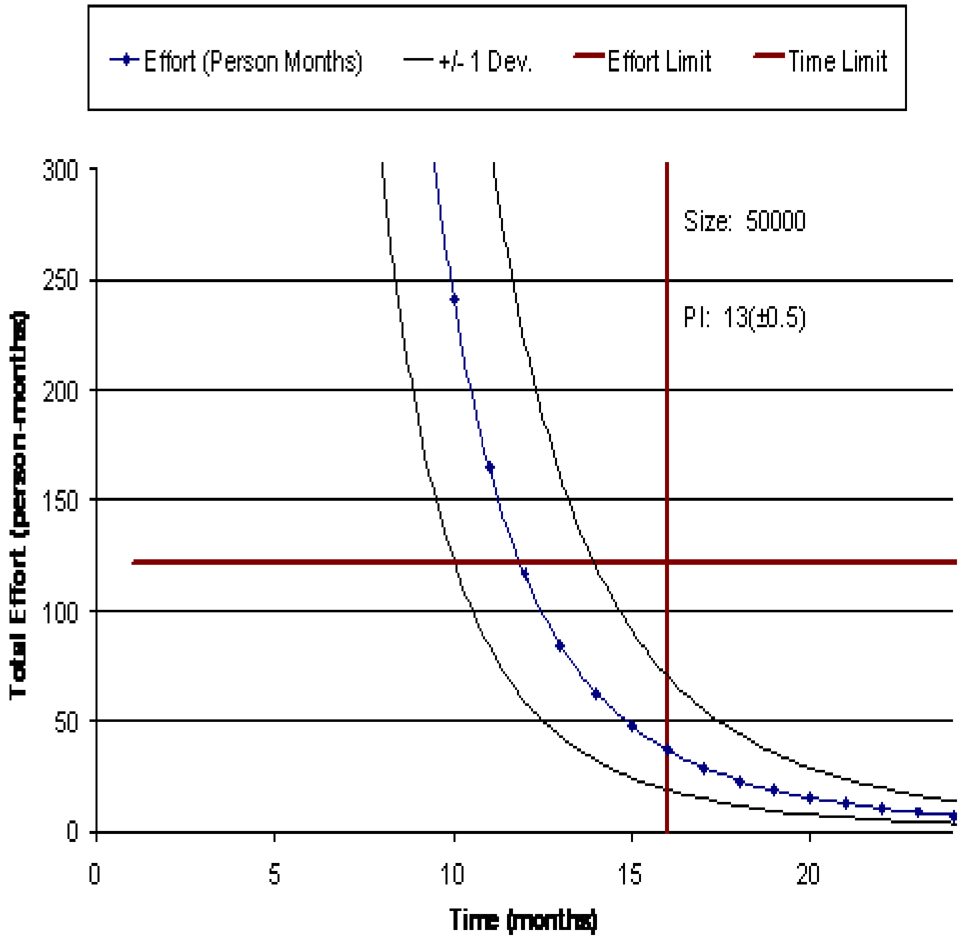
- Research from the collected productivity data supplies initial values for productivity determined by the type of software being developed.

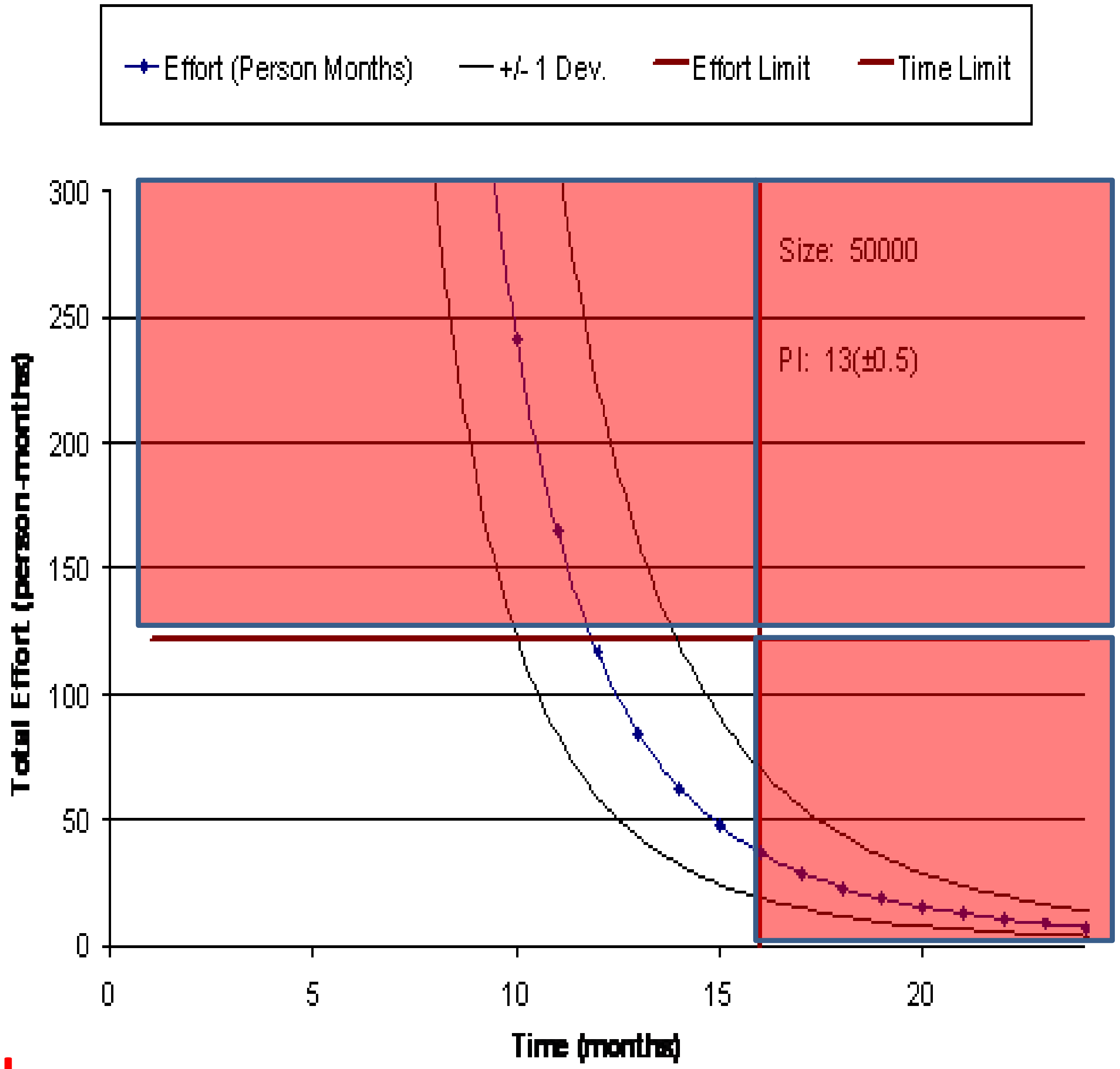
Productivity Value	Description
2,000	Real time embedded software
10,000	Telecommunication software
12,000	Scientific software
28,000	Business system applications

Quick Question

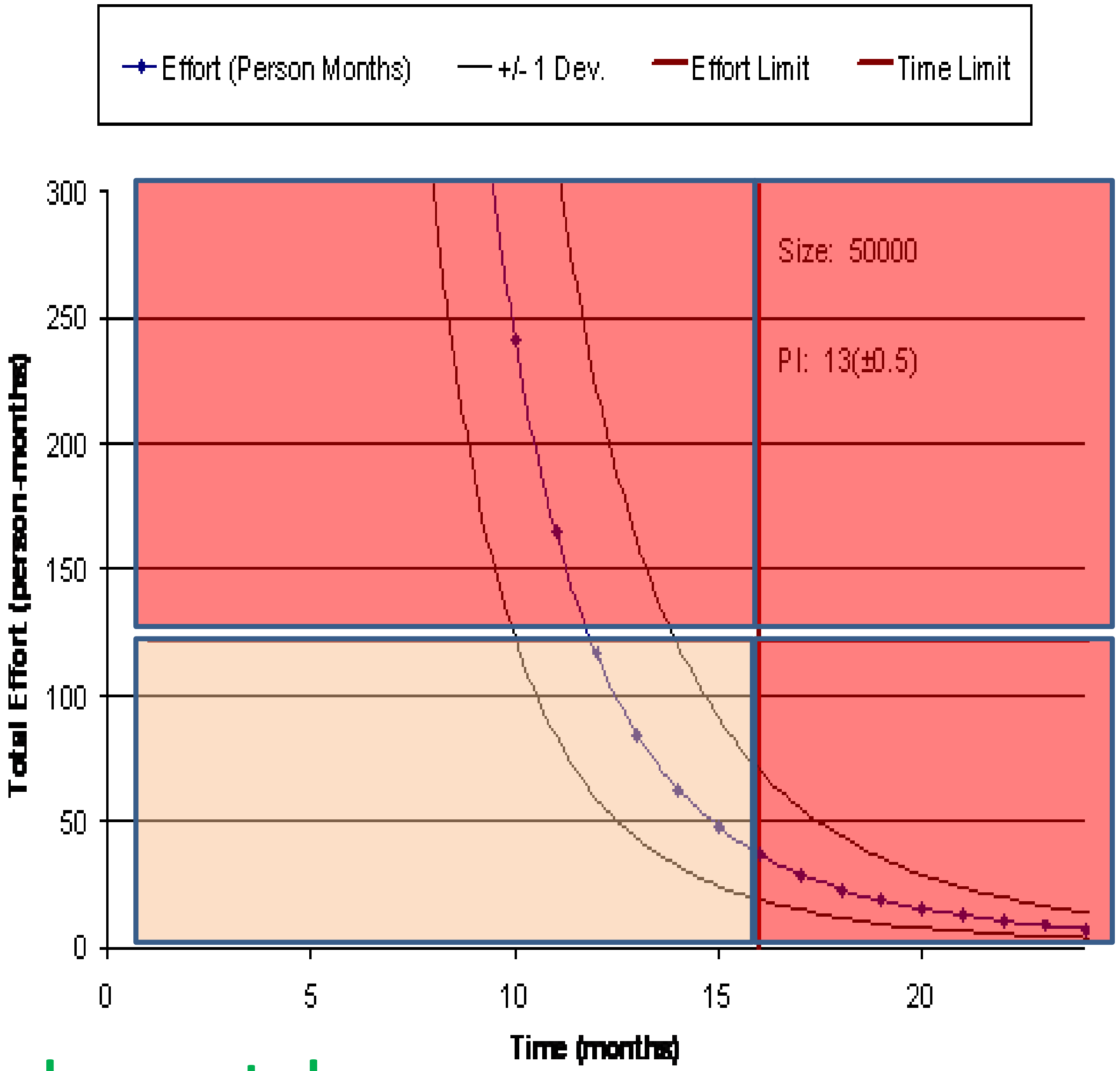
$$\text{Effort} = B * (\text{Size}/(\text{Productivity} * \text{Time}^{4/3}))^3$$

- What is the relationship before time and effort in the equation?

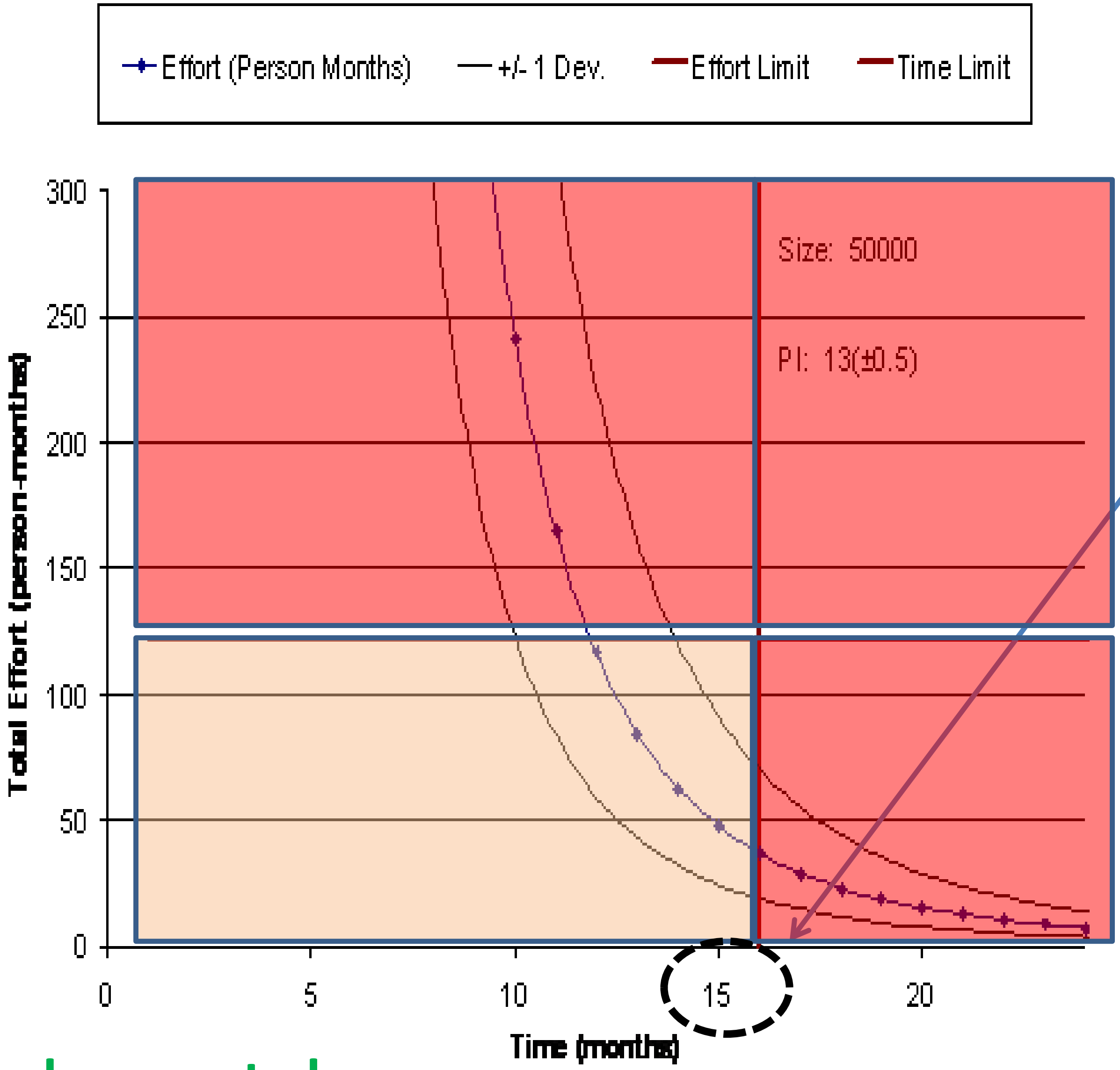

Effort (Person Months)
+/- 1 Dev.
Effort Limit
Time Limit
Size: 50000
PI: 13(±0.5)
Total Effort (person-months)
Time (months)
0
50
100
150
200
250
300
0
5
10
15
20



Impossible zones



You can only operate here



In case you have not noticed, this is just an example. You do not have 15 months, not even 15 weeks!

You can only operate here

You can pour more effort into less time, but there are limits … ☹

Cohort Exercise 4 (10 min)

Your project needs to be completed in 3 months. The estimated size of the system you are developing is 20,000 LOC.

- Find out how much effort your team will need (assume productivity = 1200).
- The team members decide to take a break during recess week (0.25 month). Is it wise?
- Amongst the following choices, select the measure(s) you feel are most appropriate for your team to meet the original deadline *and* have the planned break.
  - Add two new members to the team immediately?
  - Every member of the team work longer hours each day?
  - Re-estimate the size of the system to be lower than 20,000 LOC?

Homework This Week

- Meet and discuss with your group on how you plan to finish the project within the deadline.