**Observations on Estimations**

Problem: There are things you have to commit to, even though you don’t have the big picture/know what the future holds.

You need quantifiable values/metrics to do such estimates, but at the start such information does not exist.

Therefore, there is a lot of risk involved: Based on the team, clients, business ect.

As time goes on, estimates become easier (and can readjust).  
There will always be things you cant control, so you need to plan for the best and worst cases.

In order to make an estimation on resources/schedule/cost/effort required to produce a software product, there are **3 things:**

**Experience:**

**Access to good historical information (metrics):**

**The courage to commit to quantitative predications when qualitative information is all that exists:** You can’t account for anomalies, or every risk you are going to face during the project.

**Project risks:**

**Project Complexity:** This effects the project time immensely, eg Login functionality, is encryption needed? More complex is **both more difficult to estimate**, and carry **more risk**.

**Project Size:** As there is a guarantee that the project scope will change along the way, the bigger the size, the bigger the changes. So how do you estimate these future changes that you don’t know about.

**The degree of structural uncertainty:** What are your resources like, what are their resources like

**The availability of historical information**: Looking at your teams past experiences.

**Software Scope and Feasibility**

**Software Scope:** Functionality, data inputs and outputs, user content, performance, interface, reliability, constraints… Basically the entire project.

**Scope:**

1. **A narrative description of software scope is developed after communication with all stakeholder:** (Deliverable 1) This is a procedural discussion with sponsors to determine scope
2. **A set of use cases is developed by end users:** Data that client gives for scenarios

**Resources in a project**

**People:**

Number: What is the size of the teams  
Skills: What their abilities are. Do you need less people at a higher/lower level?  
Location: Where is everyone located?

**Environment:** Most companies will give these tools to you. You need to understand the communication between the developers and what they need to do.

Software tools:  
Hardware:  
Network resources:

**Reusable software:**COTS Components:  
Full experience Components: Fully developed and tested components.  
Part experience components: Partially develop/not fully tested.  
New Components: Non developed component.

These heavily effect resources; Using a fully developed tool you don’t have to do a lot, but a new tool takes a lot of time.

**Decomposition Techniques**

**Software project estimation is a form of problem solving**:

**Too complex in one piece:** Modularity, as its impossible to give an estimate to the whole scope – breaking it up into smaller, estimable parts is needed.

**Before an estimate can be made, you must understand the scope:** Deliverables

**Software Sizing**

1. The degree to which you have properly estimated the size of the product to be built
2. Ability to translate the size estimate into human effort
3. Degree to which the project plan reflects the abilities of the software team
4. The stability of the product requirements and the environment that supports the software engineering efforts.

**Problem Based Estimations**

Lines of code (LOC) and Function points (FP) data used in two ways during software project estimations:

1. Estimation variable to “size” each element of the software
2. Baseline metrics collected from past project and used in conjunction with estimation variables to develop cost and effort projection.

FP: Breaks it into functions being developed  
LOC: Breaks it into lines of code

**Base the estimate on the process that will be used:** eg Waterfall vs Agile, Waterfall huge change cost as redo everything, Agile has a smaller change cost.

**Decomposed:** Activities, Actions, Tasks, Effort

**Estimations with use cases**

Insight into software scope and requirements: You know exactly what’s involved, allowing you to more accurately estimate size.

**The use case points must take the following characteristics into account:**

1. The number and complexity of the use cases in the system.
2. The number and complexity of the actors on the system: You can have different interfaces for different actors.
3. Various non-functional requirements that are not written as use cases: Something you don’t directly address with a single function, quantifiable…
4. The environment in which the project will be developed: You are given this, there are no uncertainties anymore