Advanced Software Engineering

Last Lec

CSE608
Software Development Life-Cycle Models

Program's life from requirements -> Ready to work at client's side.

Dr. Islam El-Maddah

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

As a client, I'm getting the benifit of the program when I have a "tested" code.
Here, we are trying to work on the idea of "models/functions" where each function is independent from one another.
Eventhough there might be some function dependent, we can release V1 with working features and then release V2.

Each Phase has an "Output"

Phase	Output
Requirements analysis Elicitation, Analysis, Specification	Software Requirements Specification (SRS), Use Cases
System Design	Design Document, Design Classes
Program Implementation	Code
Test	Test Report, Change Requests

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

Models

- Different projects may interpret these phases differently.

- Each particular style is called a

  “Software Life-Cycle Model”

“Life-Cycle” Models

- Single-Version Models

Created once and that's all, the program is only used/ran once

- Incremental Models
  - Single-Version with Prototyping

Creating a program then updating it and adding on top of it. without removing old features.

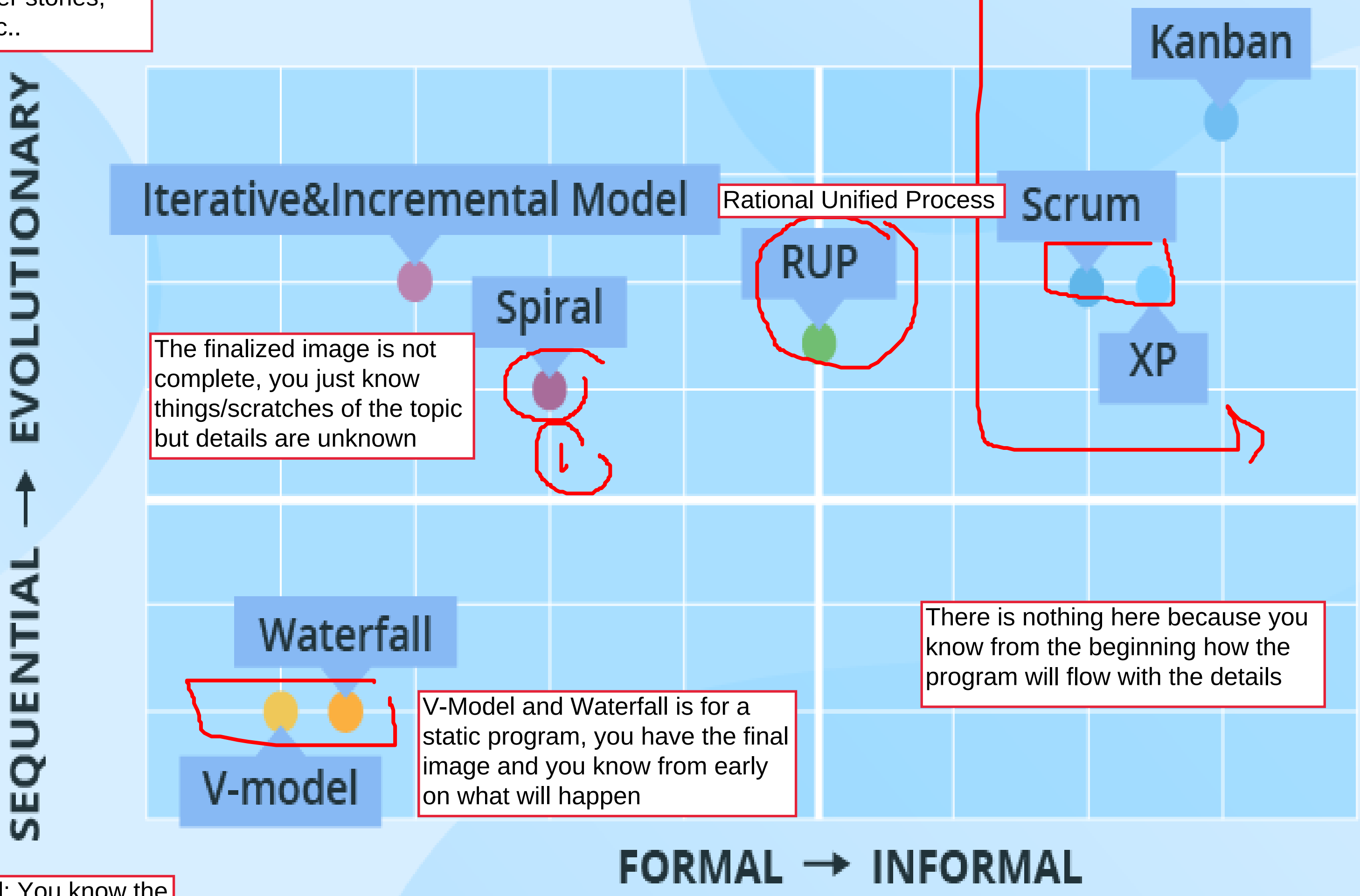
- Iterative Models

Each time, you start all over, like you're trying to create something that works all over again.

Incremental vs Iterative
- Each time you try to add new things up on top | Each time it's like have a higher version and have stronger features than before

SLDC models

A new ideas may occur after a while, new user stories, new updates etc..



Rational Unified Process

The finalized image is not complete, you just know things/scratches of the topic but details are unknown

V-Model and Waterfall is for a static program, you have the final image and you know from early on what will happen

There is nothing here because you know from the beginning how the program will flow with the details

Sequential: You know the program flow exactly and how you will split it

Requirement Formal : we understand what exactly is going to happen

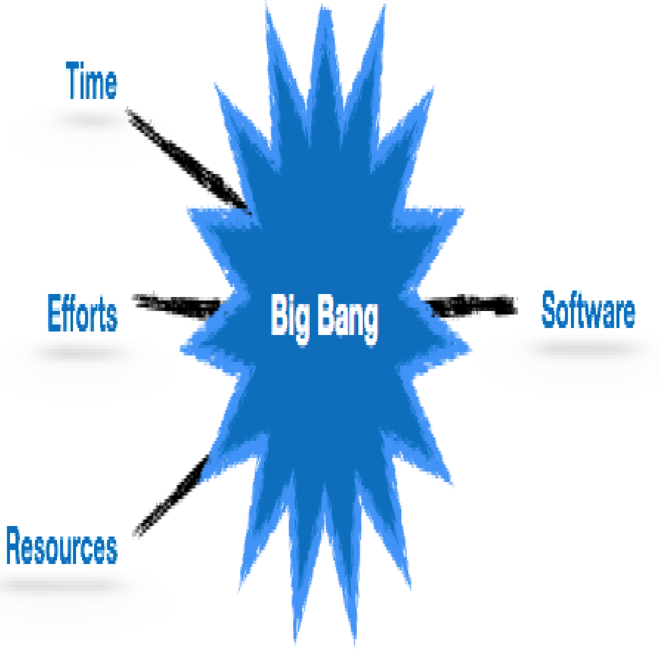
“Life-Cycle” Models (1)

- Single-Version Models
  - Big-Bang Model
  - Waterfall Model
    - Waterfall Model with “back flow”
  - “V” model: Integrating testing

Big-Bang Model

Big-Bang is trivial and nearly no one do it unless it was a single developer that can do all the program by himself. It's like one time, we are trying to push ourselves to the limit and creating everything we know without requirements/design/steps.
It has a higher risk for failure and size must be limited so it can operate well.

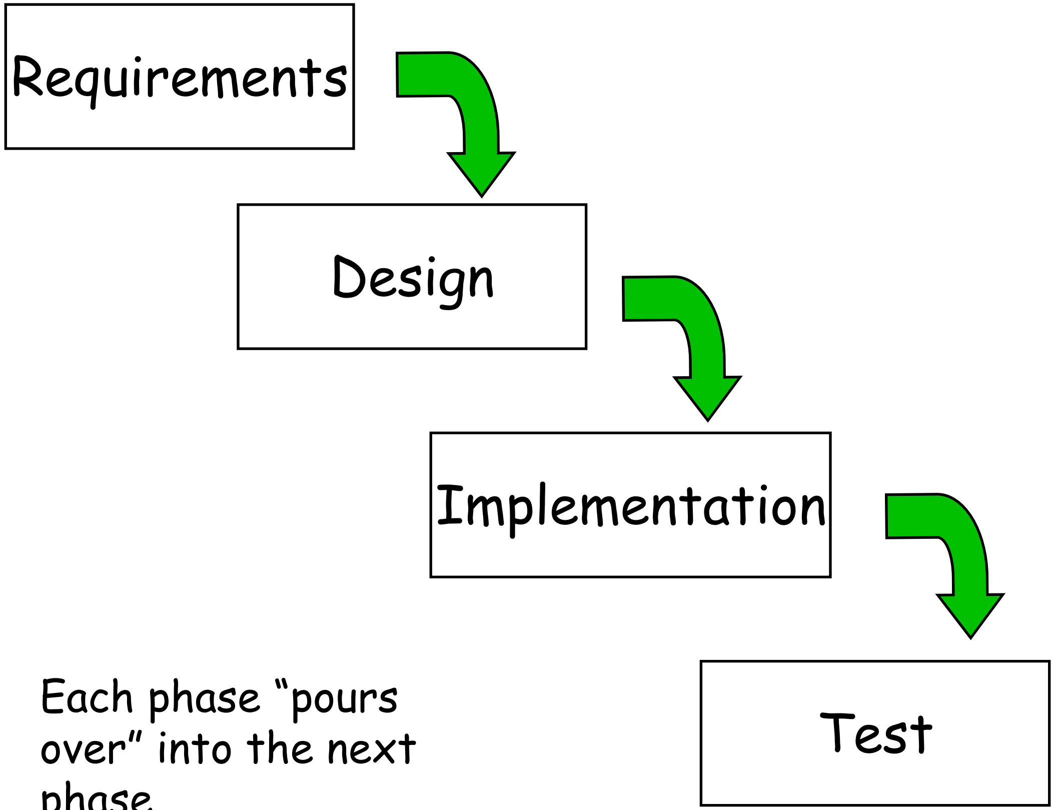
- Developer receives problem statement. Isolation
- Developer works in isolation for some extended time period.
- Developer delivers result.
- Developer hopes client is satisfied.



It's like the developer understands the client better than the client himself

Waterfall Model

This model is built on the 4 previous steps.
Finish everything and then go to the next phase.



Requirements

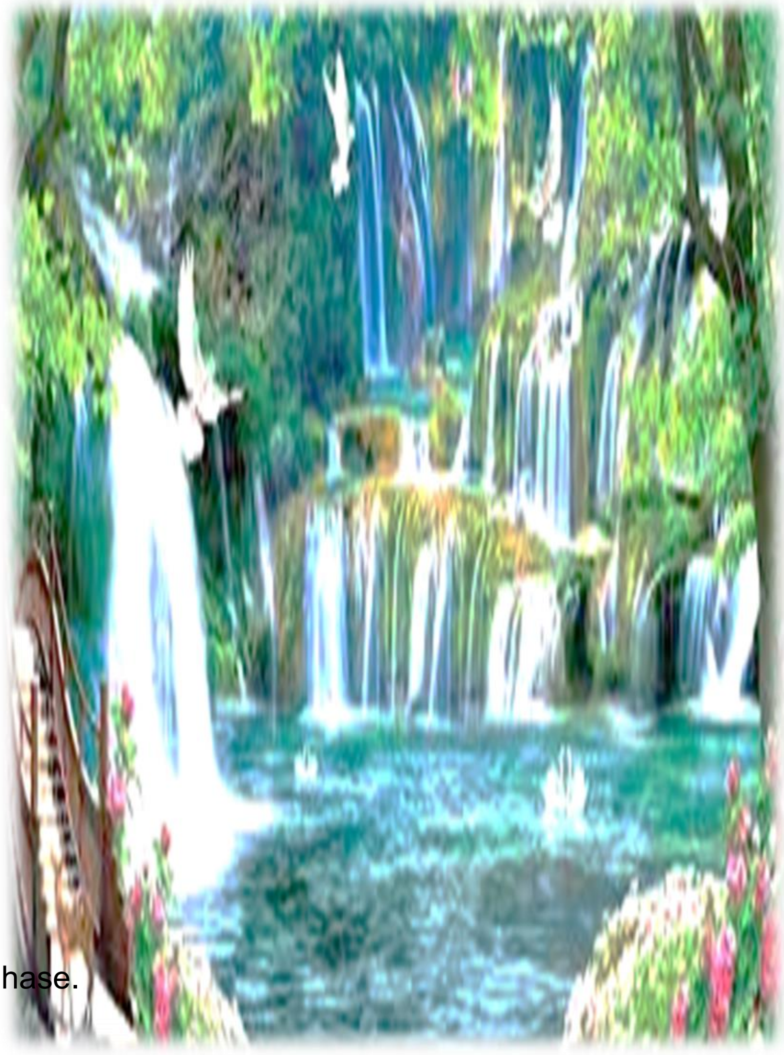
Design

Implementation

Test

Each phase "pours over" into the next phase.

Pure model: when finishing a phase, you don't return to it again.
So we need to take care and take our suitable time so we can push into next phase.
Because a mistake in previous step will induce a change in the next step.
Rarely someone do this

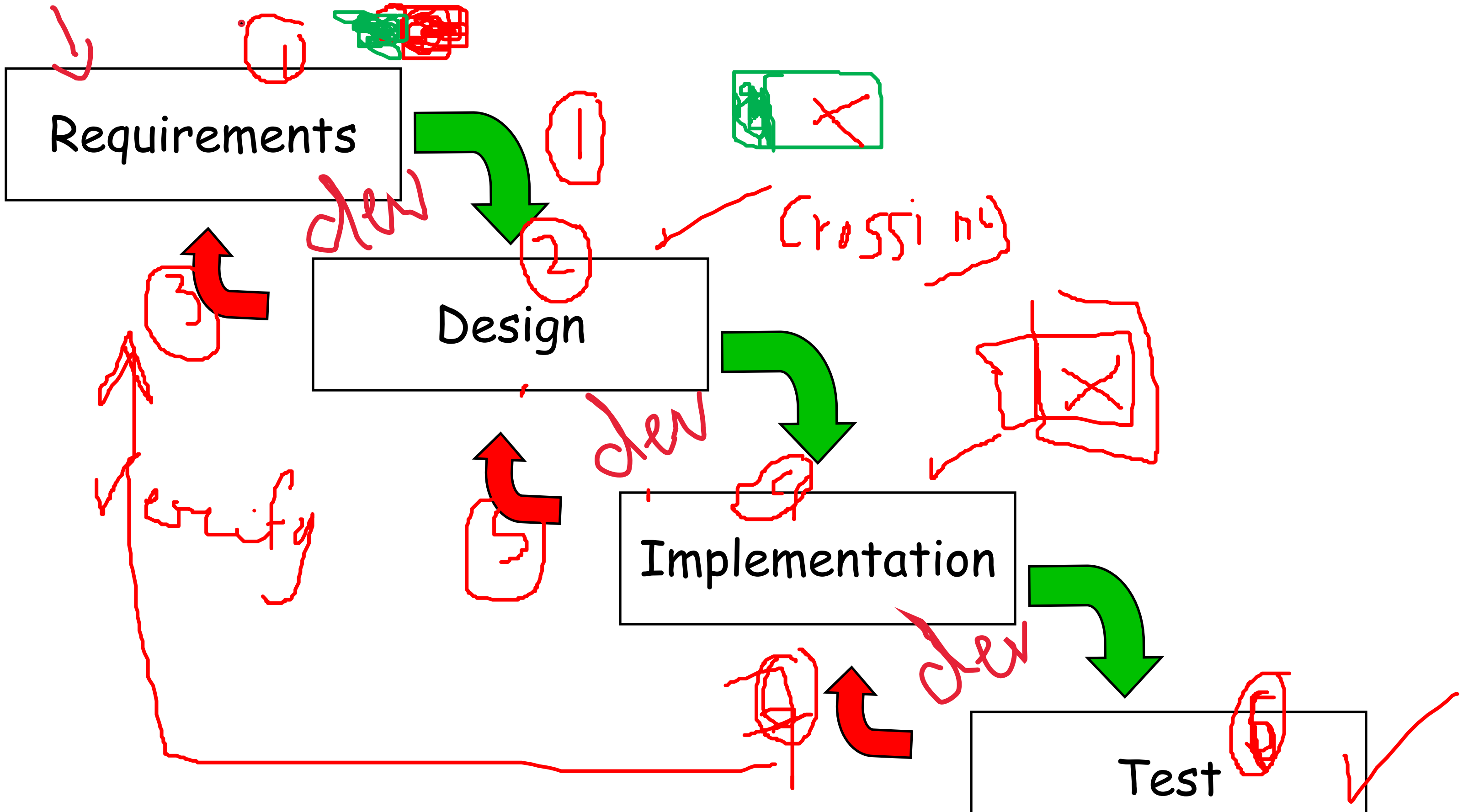


This is the correction of the previous slide, we have a backflow so we can fix whatever problems we face in next/current stages. There are some problems in Waterfall model, that sometimes we need that the client does participate in the development, so his opinion is heard. Sometimes there is a problem in designing, so test won't come in handy,

Waterfall Model with Back Flow

(sometimes this is implied by "waterfall")

client

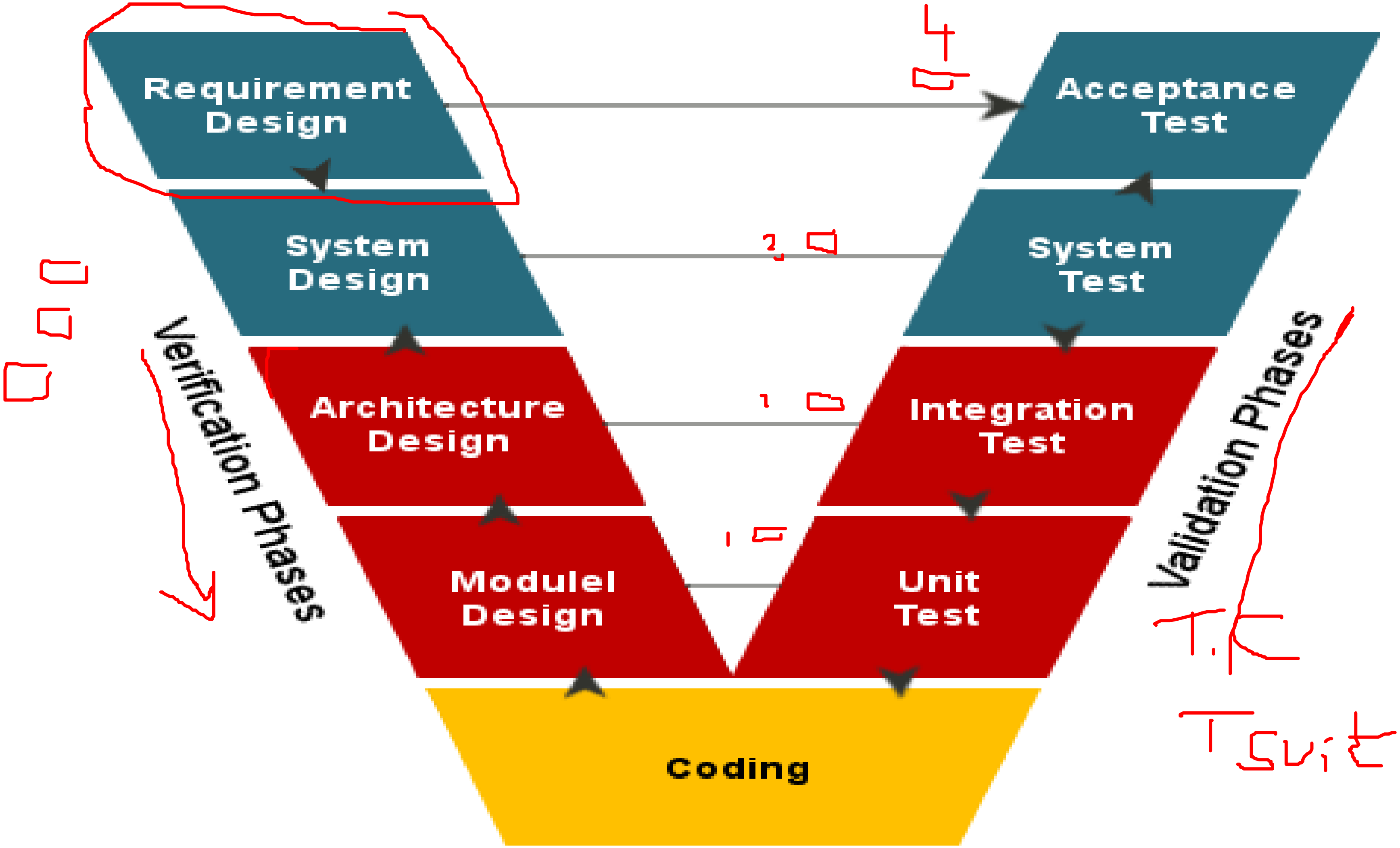


Adjustments made to immediately previous phase based on issues with successive phase.

Make sure that each phase is working appropiately, and can integrate together.
We have a kind of link between each phase and next step/test.
We need to fill gaps also.

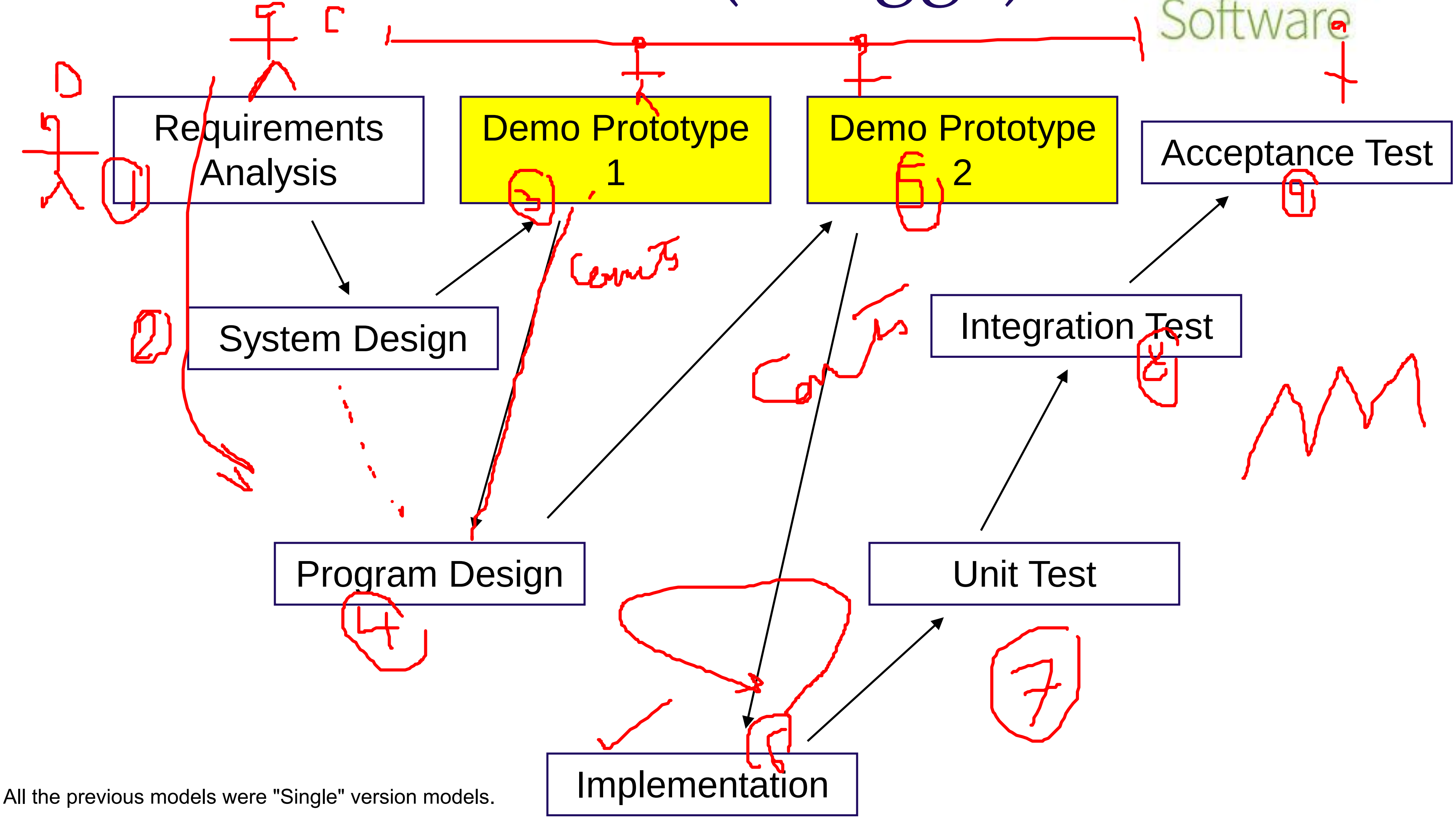
"V" Model

Each phase has corresponding test or ***validation counterpart***



V Model adjusted, after knowing general purposes we can create a demo prototype.
Here, the user can also interact with the yellow prototype, so we can flow correctly.
After knowing the program design, we can know the details etc.. so we can create another prototype
for user to inspect. Higher change of success because design, implemetation are based on the user's opinion.
It's like we created 2 milestones from requirements to the acceptence test. In the V Model it's from requirements to acceptence rightaway no in between.

Sawtooth Model (Brugge)



All the previous models were "Single" version models.

V Mode, Sawtooth, eventhough there is a client/user intervention, but they are considered single model because this program is only done once, every phase is combined -> processed together.

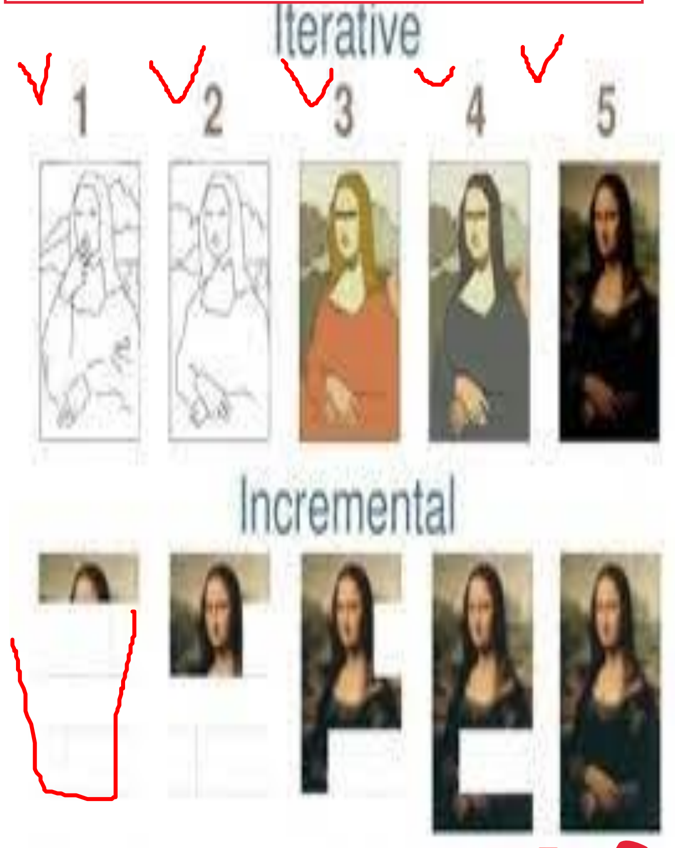
Incremental vs. Iterative

f_1 f_2

f_1 f_1 f_2 f_3

Each iteration has a value and version but latest stage is highest

- These *sound* similar, and sometimes are equated.
- Subtle difference:
  - **Incremental:** ***add to*** the product at each phase
  - **Iterative:** ***re-do*** the product at each phase
- Some of the models could be used either way



Sometimes, in Agile, we get the user stories from different clients, so we can increment or iterate in agile.

Value is finalized here

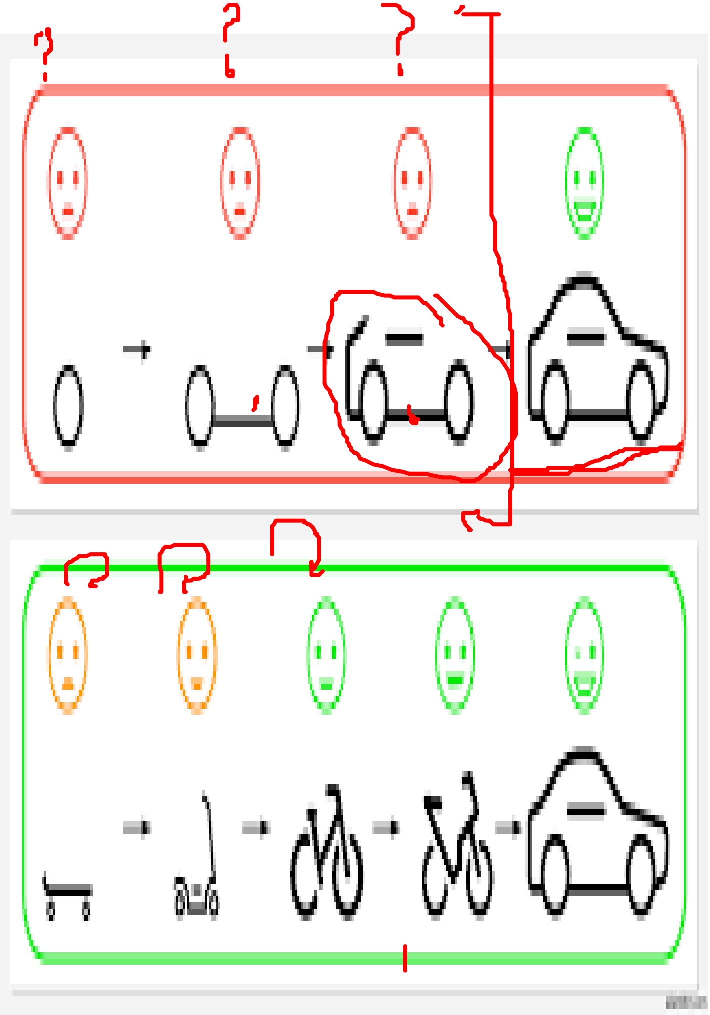
Example: Building a House

- **Incremental**: Start with a modest house, keep adding rooms and upgrades to it.

The most important thing is building the basics, but sometimes, this is almost impossible, because you want the home's infrastructure is already built so we need design etc..
You may not reach your end goal.

- **Iterative**: On each iteration, the house is re-designed and built anew. It's like each time you start from scratch.

- Big Difference: One can live in the incremental house the entire time! One has to **move** to a new iterative house.

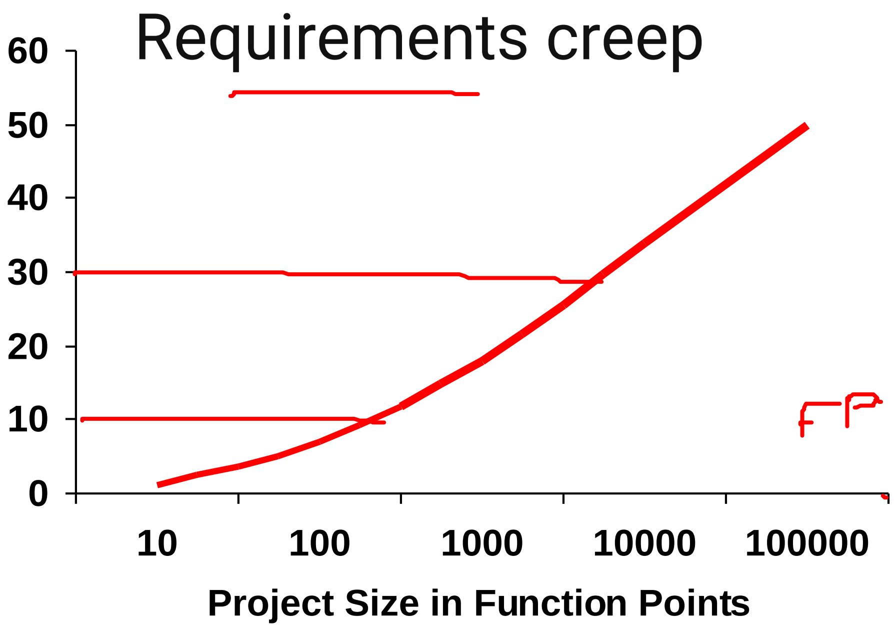


- Waterfall: When you fail at something -> You fail at the whole program.
- Agile: you finish step by step so it doesn't depend on each others

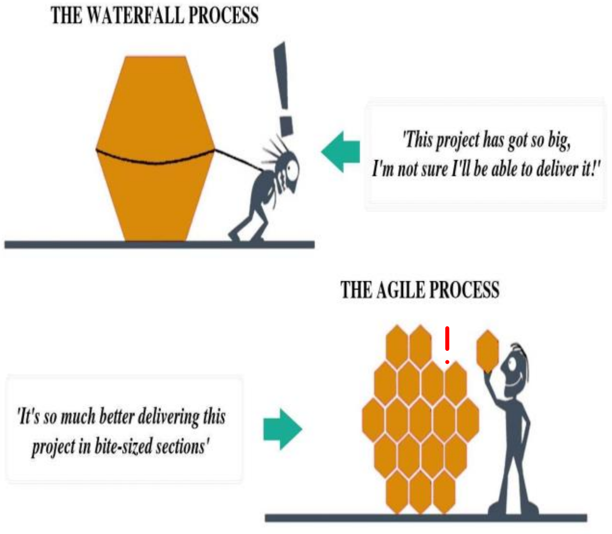
Why Not Waterfall?

Sự bất

1. Complete Requirements Not Known at Project Start



Source: Applied Software Measurement, Capers Jones, 1997. Based on 6,700 systems.



THE WATERFALL PROCESS
'This project has got so big, I'm not sure I'll be able to deliver it!'
THE AGILE PROCESS
'It's so much better delivering this project in bite-sized sections'

Requirements creep refers to new requirements entering the specification after the requirements are considered complete.

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

We care about the idea of measurement, We use LOC and FP.

Late

Requirement

LOC vs FP

- A FP **function point** is a unit of complexity used in software cost estimation. Function points are based on number of user interactions, files to be read/written, etc
- **SLOC** means number of source lines of code, also a measure of program complexity.

Why Not Waterfall?

2. Requirements are not stable/unchanging.

- The market changes—constantly.
- The technology changes.
- The goals of the stakeholders change.

Why waterfall is not stable, because there is always changing.
technology changes, requirements changes -> we need to respond to these changes.

Source: Craig Larman

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

Why Not Waterfall?

3. The design may need to change during implementation.

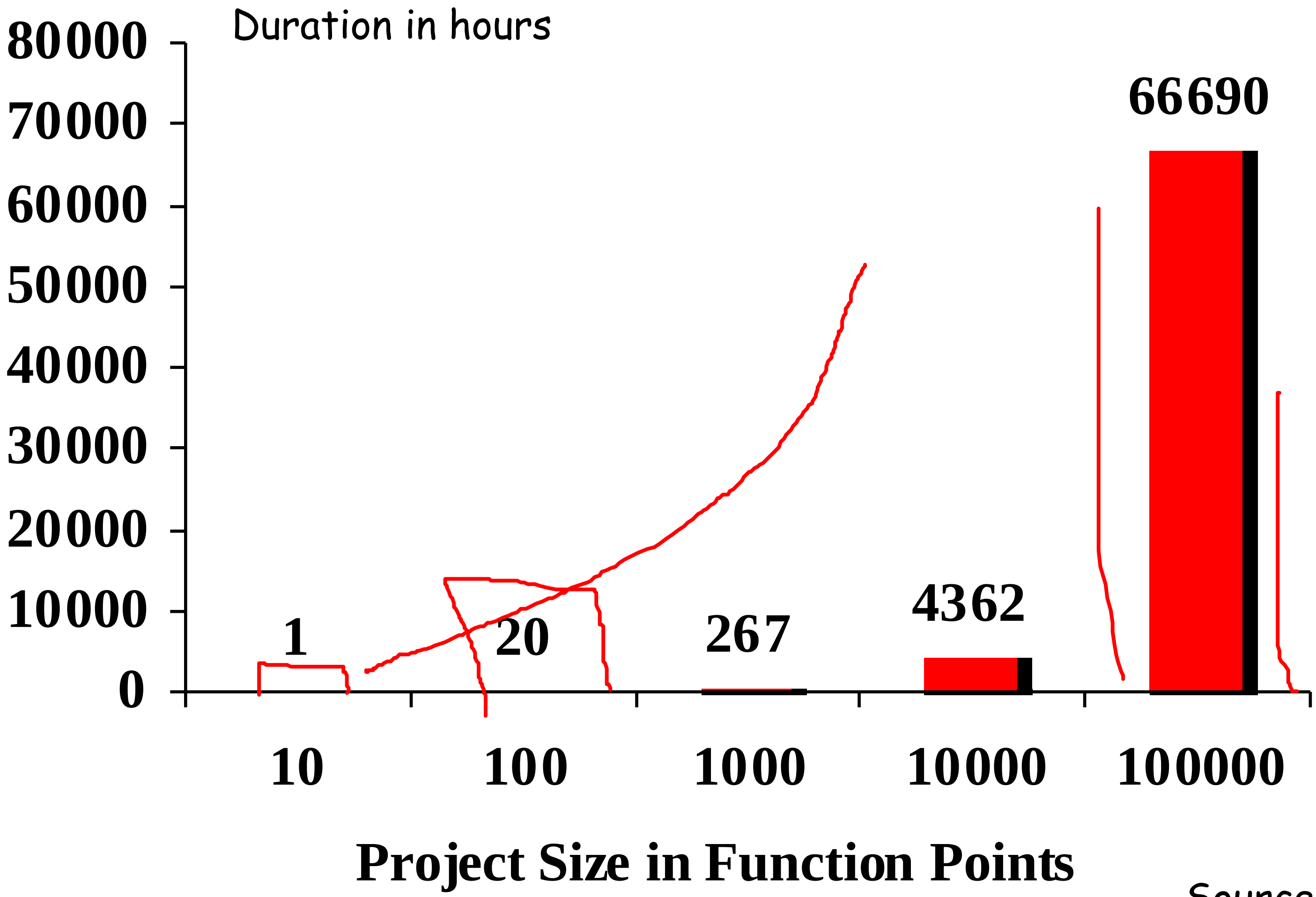
- Requirements are incomplete and changing.
- Too many variables, unknowns, and novelties.
- A complete specification must be as detailed as code itself.
- Software is very “hard”.
  - Discover Magazine, 1999: Software characterized as the most complex “machine” humankind builds.

Source: Craig Larman

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

Large vs. Small Steps:

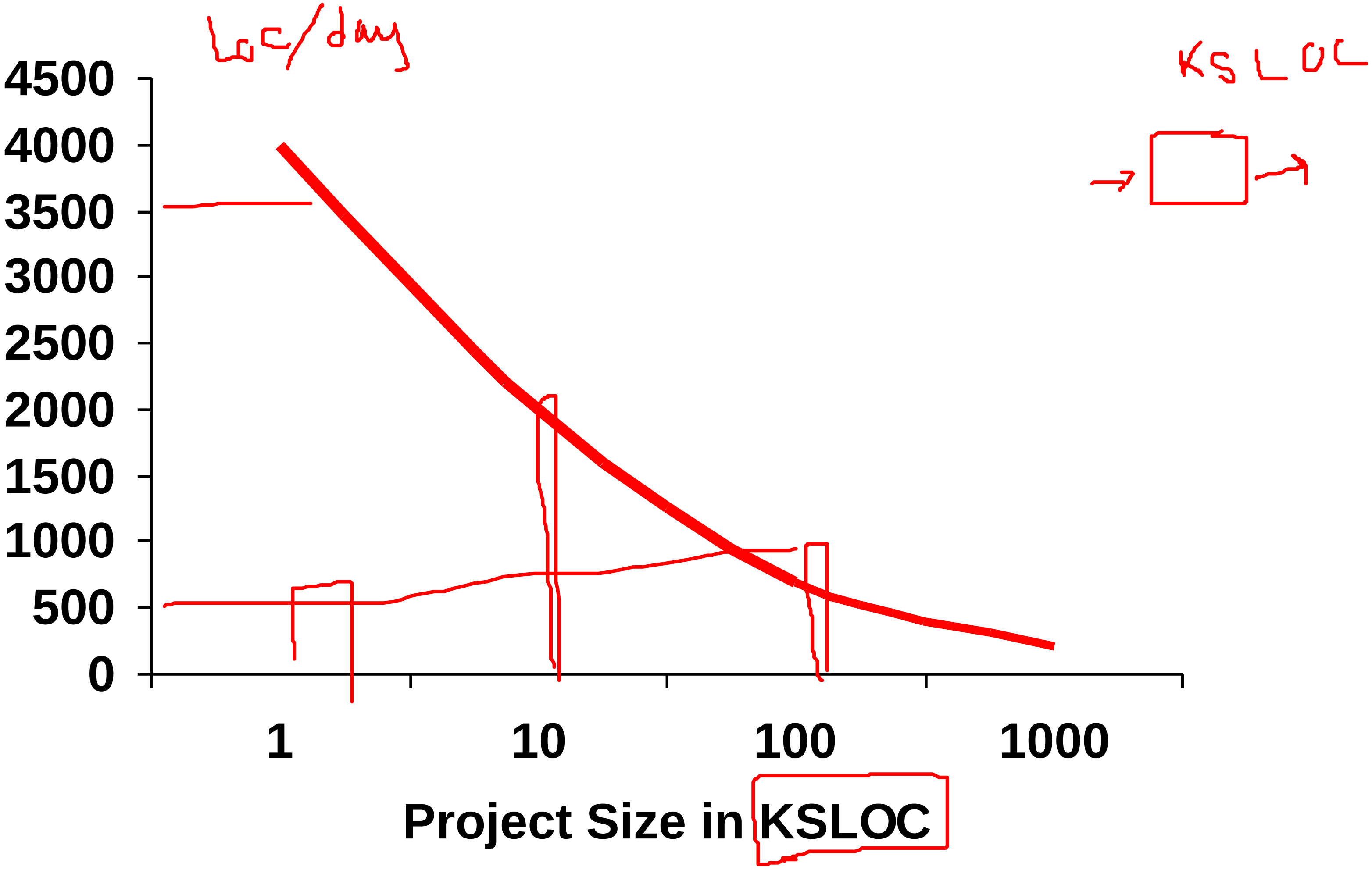
Project Duration



Source: Craig Larman

These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

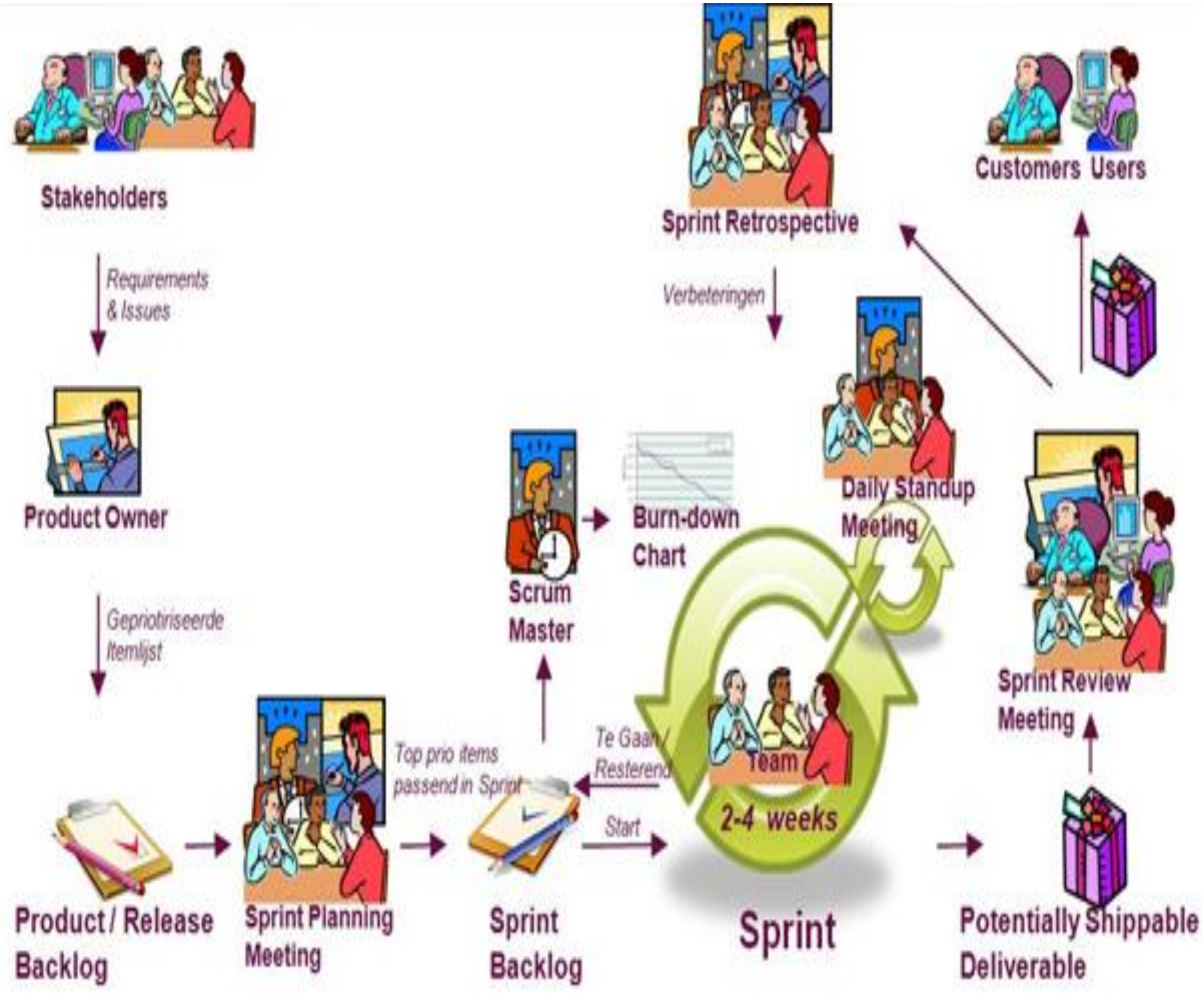
Large vs. Small Steps: Productivity



These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

“Life-Cycle” Models (3)

- **Iterative** Models
  - Spiral Model & Variants
    - ROPES Model
    - Controlled Iteration Model: Unified Process
    - Time Box Model
  - Scrum Model
    - Fountain Model

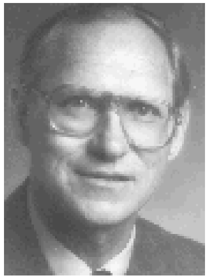


Boehm Spiral Model

(of which some other models are variants)

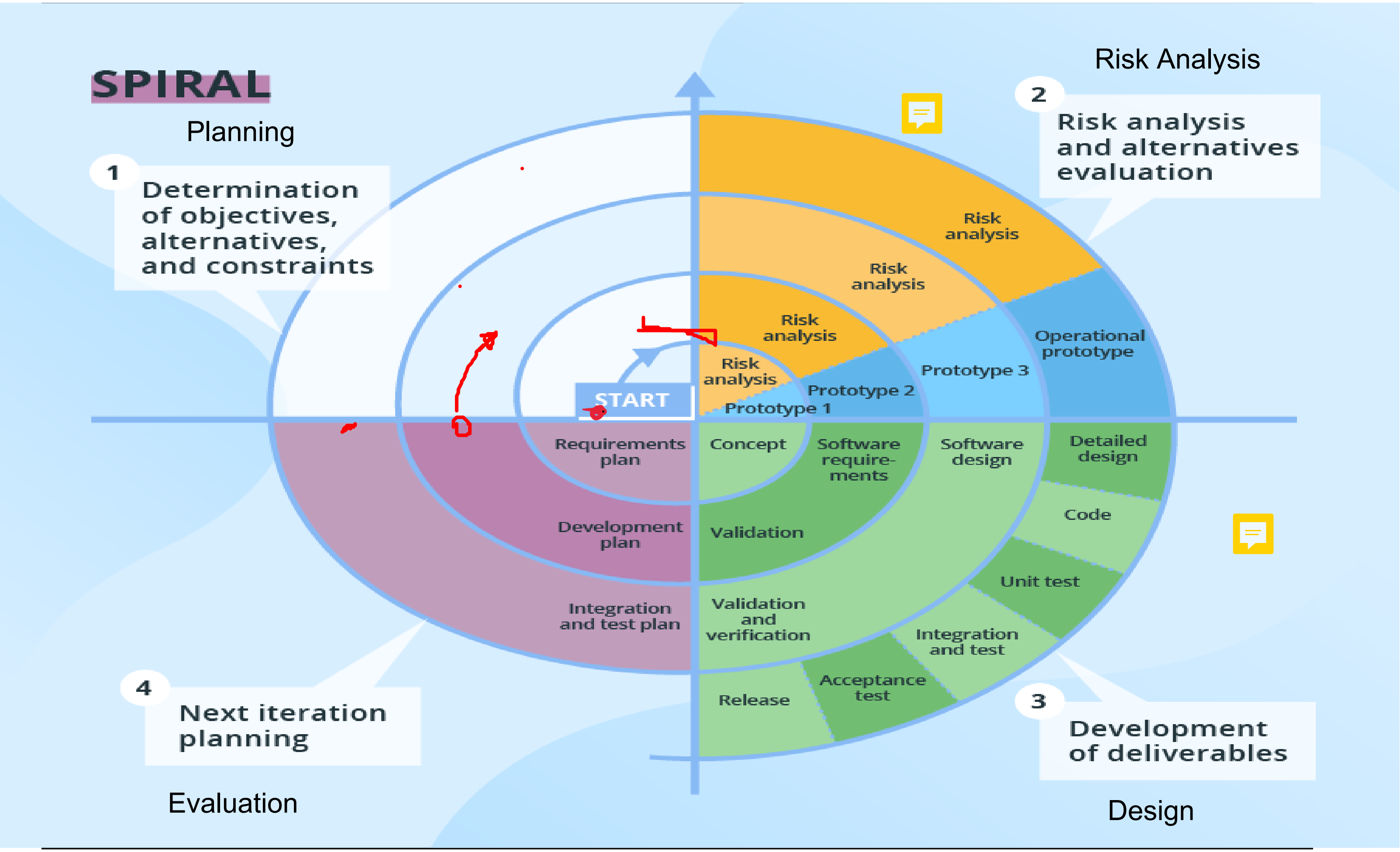
Each stage is considered an iteration, each iteration has its purpose/requiremnts

- An iterative model developed by Barry Boehm at TRW (1988), now Prof. at USC
- Iterates cycles of these project phases:
  1. Requirements definition ?
  2. Risk analysis
  3. Prototyping
  4. Simulate, benchmark
  5. Design, ~~implement~~, ~~test~~
  6. Plan next cycle (if any)



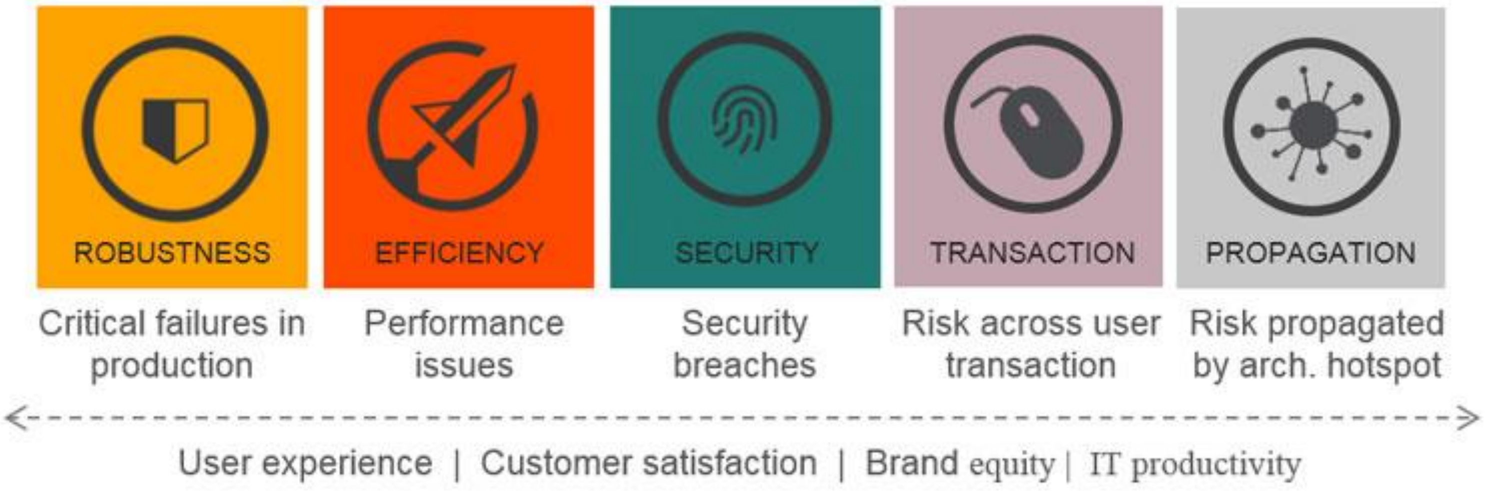
Prof. Barry Boehm

Boehm Spiral Model



These slides prepared by Prof. Bob Keller of Harvery Mudd College and printed for ERAU/Prescott SE300 usage with his kind permission. Slides avaialble online at http://www.cs.hmc.edu/courses/2005/spring/cs121/slides/slides05.ppt

Risk? What risk?


ROBUSTNESS
Critical failures in production
EFFICIENCY
Performance issues
SECURITY
Security breaches
TRANSACTION
Risk across user transaction
PROPAGATION
Risk propagated by arch. hotspot
User experience | Customer satisfaction | Brand equity | IT productivity

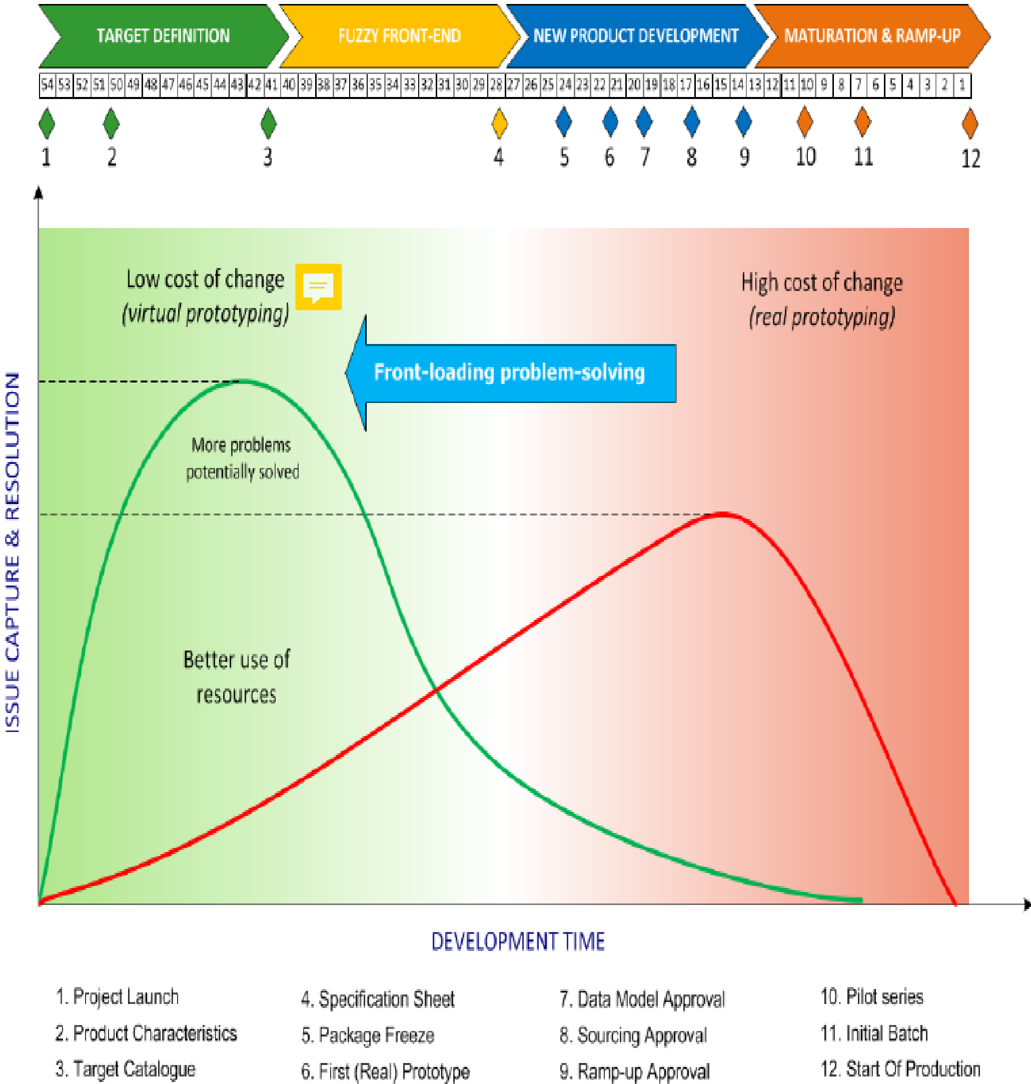
- One major area of risk is that the scope and difficulty of the task is not well understood at the outset.

Ways to Manage Risk

- Risk cannot be eliminated; it must be managed.
  - Do thorough **requirements** analysis before the design.
  - Use **tools** to track requirements, responsibilities, implementations, etc.
  - Build ***small* prototypes** to test and demonstrate concepts and assess the approach, prior to building full product.
  - Prototype **integration** as well as components.

Front-Loading

- Tackle the unknown and harder parts earlier rather than later.
- Better to find out about infeasible, intractable, or very hard problems early.
- The easy parts will be worthless if the hard parts are impossible.
- Find out about design flaws early rather than upon completion of a major phase.



Time-Box Requirement

(can be used in iterative or incremental)

Something is iterated per unit time (1 month - 2 months etc..)

```
Requirements analysis
Initial design
while( not done )
    {
    Develop a version within a bounded time
    Deliver to customer
    Get feedback
    Plan next version
    }
```

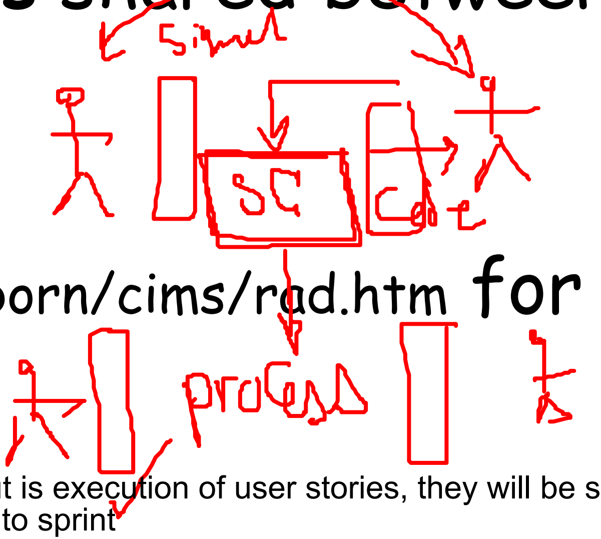
Telling the team what will the next unit time we release etc..
We can adjust our way of work to adapt time-box requirement.

Additional Models/Acronyms

- **RAD** (Rapid Application Development): time-boxed, iterative prototyping

- **JAD** (Joint Application Development): Focus on developing **models** shared between users and developers.

- See http://faculty.babson.edu/osborn/cims/rad.htm for additional points.



Last thing in iterative is Scrum, which is built on top of Agile, Most we care about is execution of user stories, they will be stored in backlog (like sorted depending on easiest/fastest/most important) -> released into sprint
We can focus on different points at same times, that's why its called Fountain instead of waterfalls.

The most thing you care about is coding and get a working one, You don't have neither a design or prototype
We use pair programming to avoid early errors/bugs
they divide the code into smaller parts and implementing it becomes a features.
Good for Operating Systems (Little functionalities on top of each others).

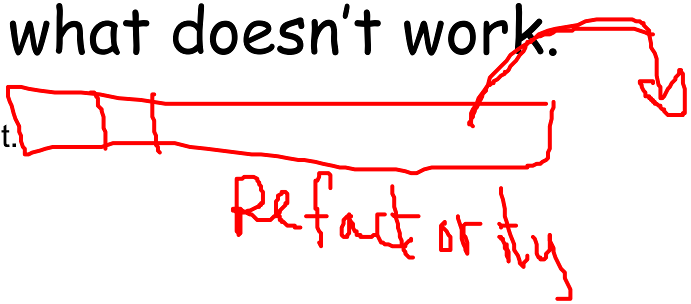
Extreme Programming (XP)

(cf. http://www.extremeprogramming.org/rules.html)

These developers are also in meetings with clients, these clients do the testing infront of the developers, there is no requirements etc.. its like just talking. finally, constructing the design and also refactoring because the program was written so fast, so you improve it.

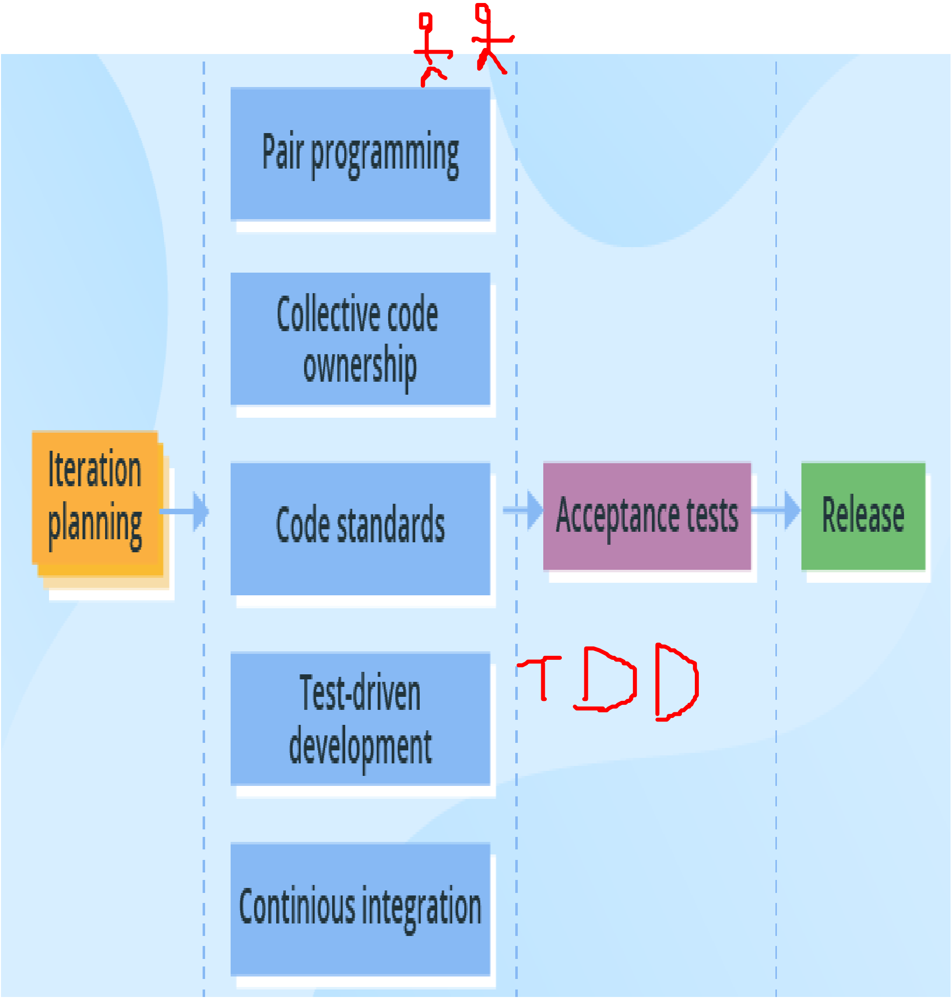
- User **stories** (something like use cases) are written by the customer.
- Complex stories are **broken down** into simpler ones (like a WBS).
- Stories are used to **estimate** the required amount of work.
- Stories are used to create **acceptance tests**.
- A **release plan** is devised that determines which stories will be available in which release.
- Don't hesitate to change what doesn't work.

XP when released had several errors because it was finished so fast.
So we care that before release, we can test and fix error



Extreme Programming (XP)

- Each release is preceded by a **release planning meeting**.
- Each day begins with a **stand-up meeting** to share problems and concerns.
- CRC cards are used for **design**. [XP and CRC were created by the same person, Kent Beck.]
- **Spike solutions** are done to assess risks.
- The **customer** is always available.



Extreme Programming (XP)

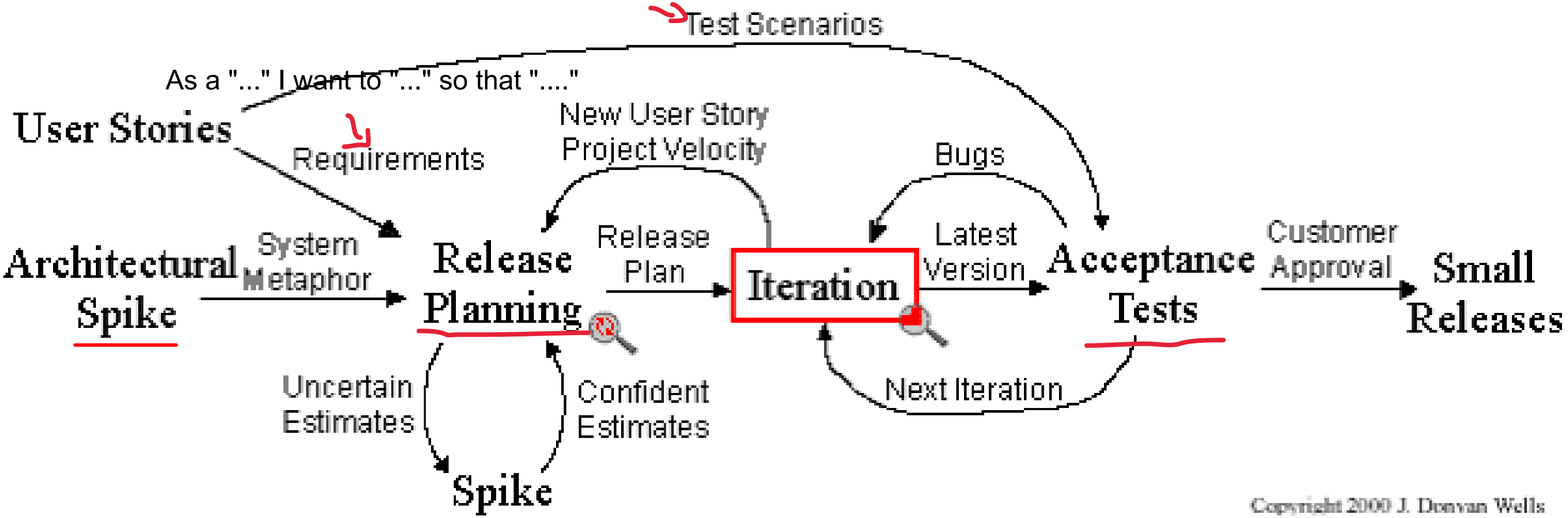
- All code must pass **unit tests**, which are coded before the code being tested (**test-driven design**).
- **Refactoring** is done constantly.
- **Integration** is done by one pair.
- Integration is done frequently.
- Optimization is done **last**.
- **Acceptance tests** are run often.

Extreme Programming

Extreme Programming Project

Release is a version, so early on we can plan which user stories will occupy what release.
Any mistakes -> reiterate.

Test acceptance in user stories (criteria for the success of story implementation)



As a "..." I want to "..." so that "...."

Use Case vs User Stories
- Use Case sees how requirement can be reached/implemented | User story is like what is the goal of this requirement.
- Has details of implemetation/High level of specification | low details/Has low leve of specification

The rational unified process

Everything here is worked altogether -> faster - effort in some places more than others

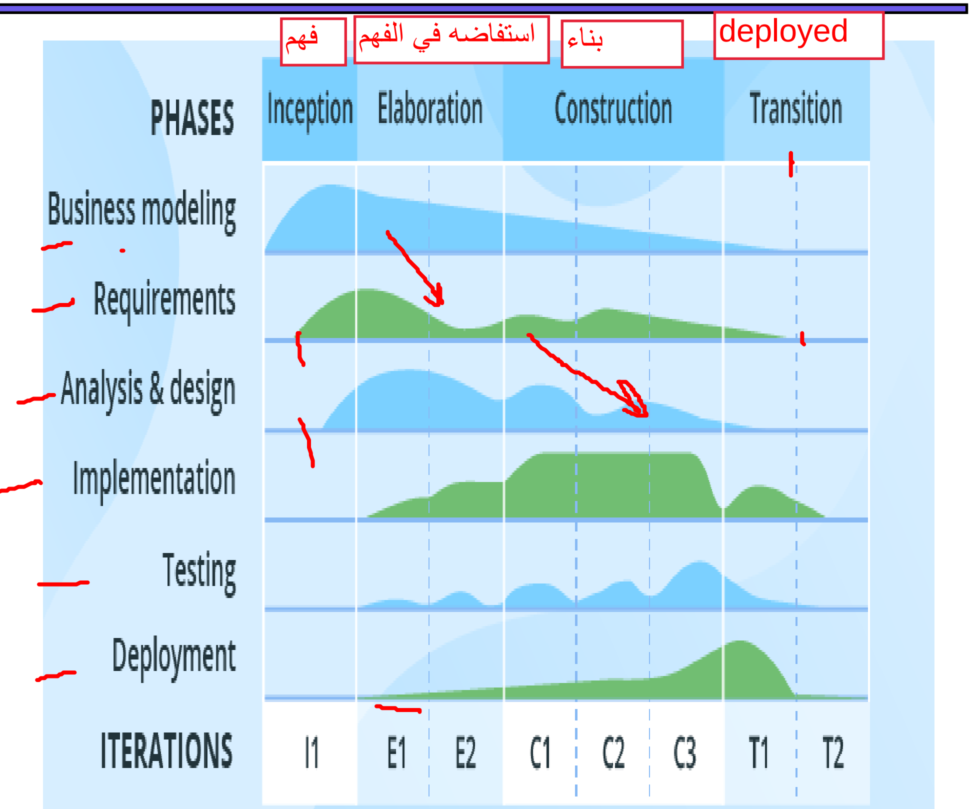
- a combination of linear and iterative frameworks.
- 4 phases – inception, elaboration, construction, and transition. Each phase but Inception is usually done in several iterations.
- All basic activities (requirements, design, etc.) of the development process are done in parallel across these 4 RUP phases, though with different intensity.
- helps to build stable and, at the same time, flexible solutions,
- still, not as quick and adaptable as the pure Agile group (Scrum, Kanban, XP).
- customer involvement, documentation intensity, and iteration length may vary depending on the project needs

فهم

استفاضة في الفهم

بناء

deployed

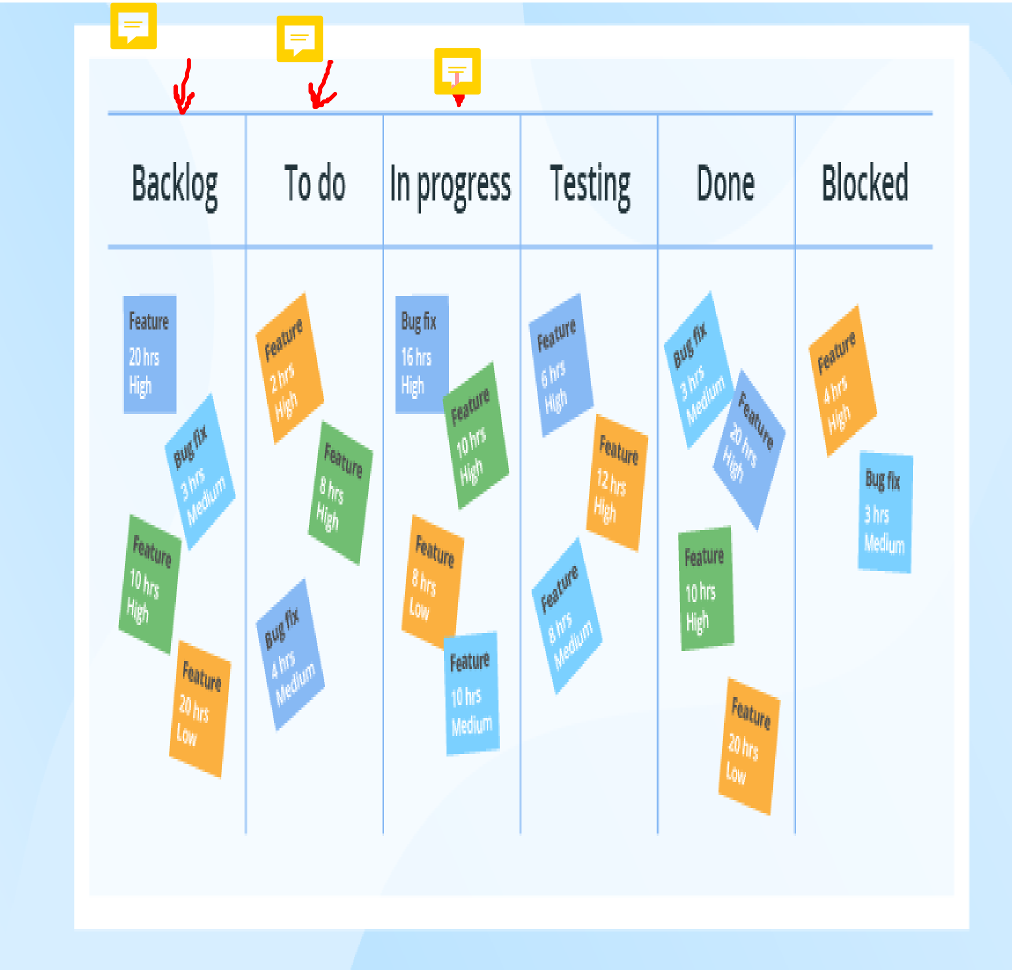


Large and high-risk projects, especially, use-case based development and fast development of high-quality software.

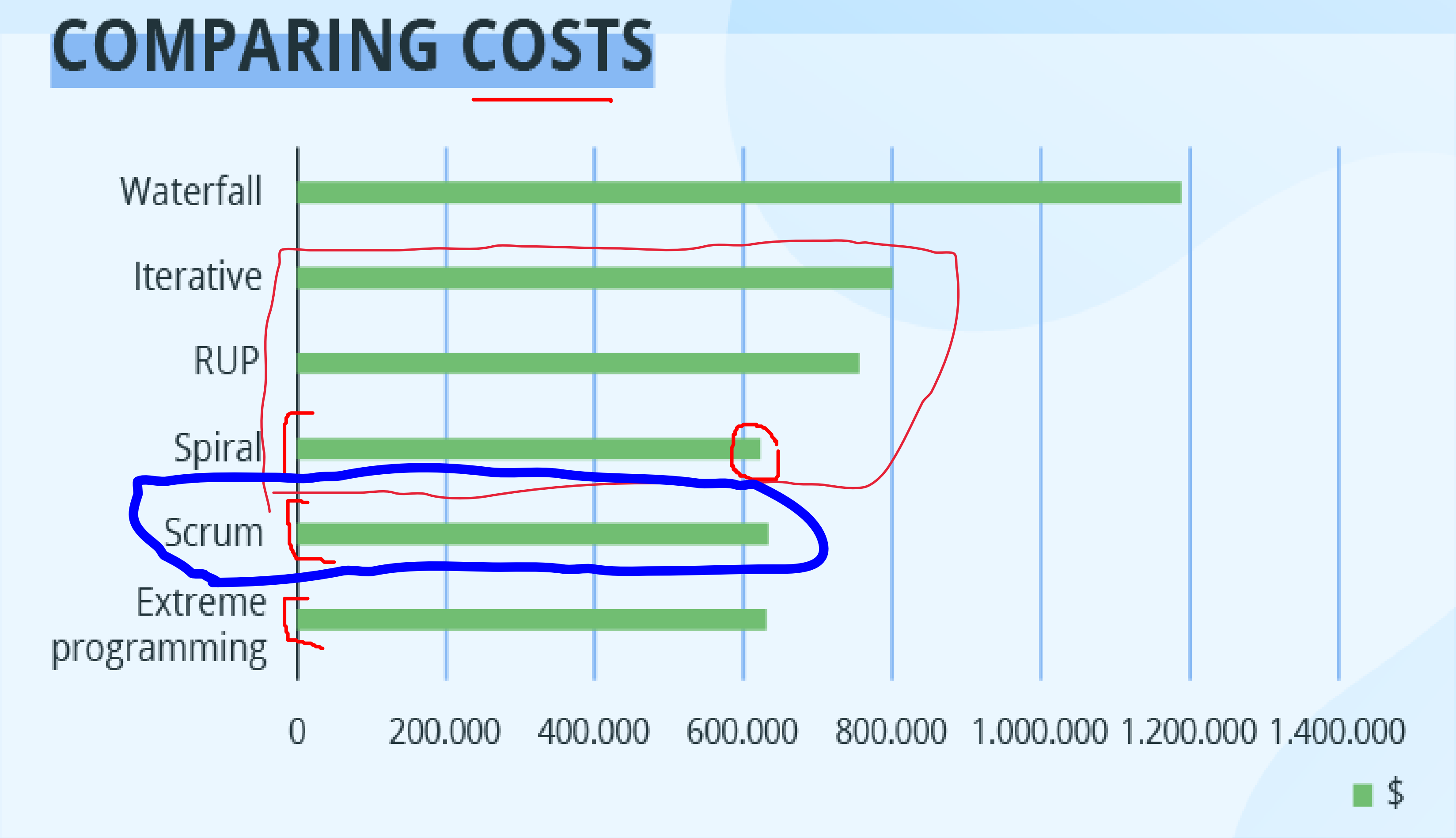
Kanban is mainly for efficiency, how to do thing efficiently

Kanban

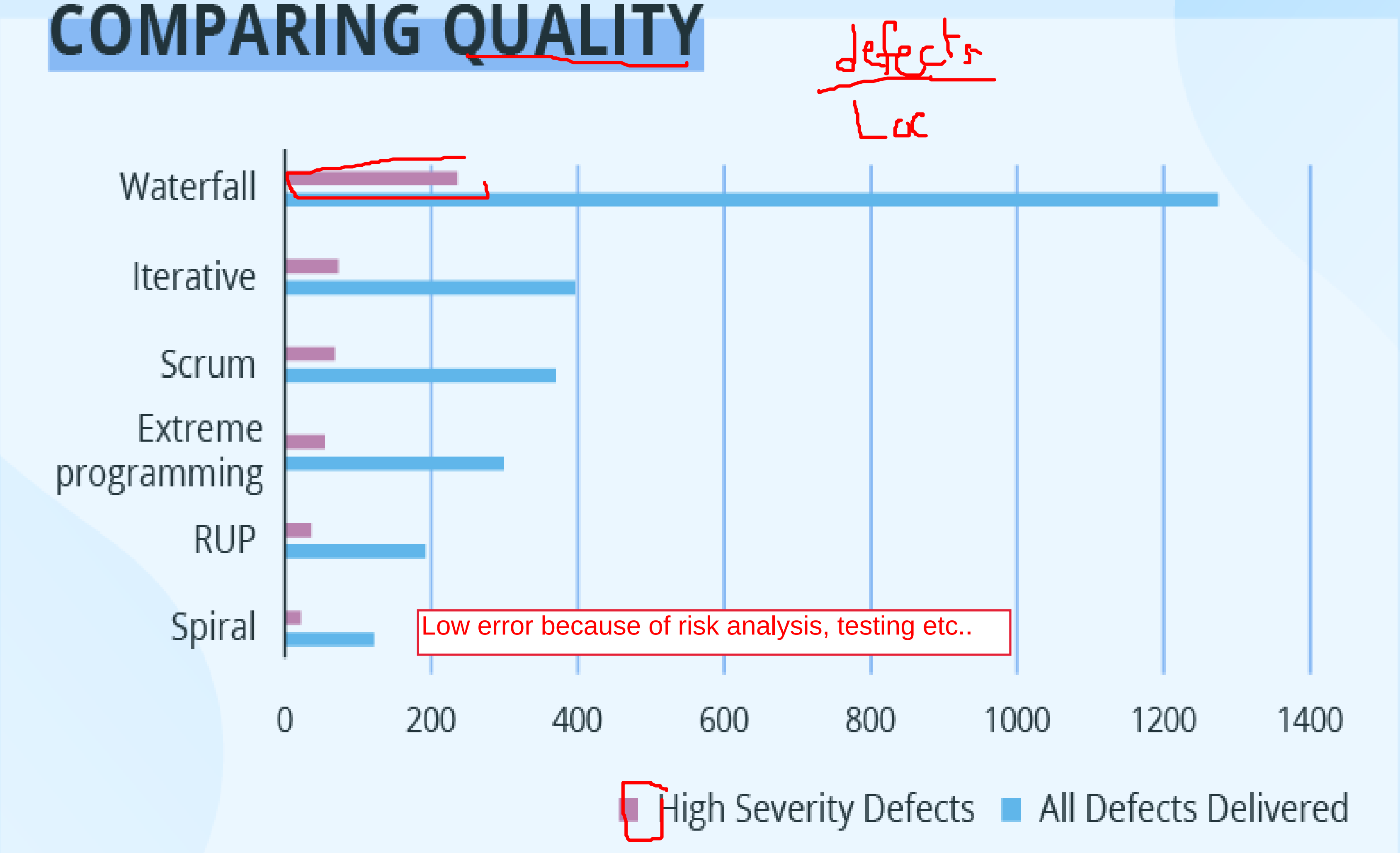
- the absence of pronounced iterations.
- the emphasis is placed on plan visualization. The team uses the **Kanban Board** tool
- the model has no separate planning stage
- a new change request can be introduced at any time.
- Communication with the customer is ongoing
- this model is frequently used in *projects on software support and evolution*.



Comparing all models



Comparing all models

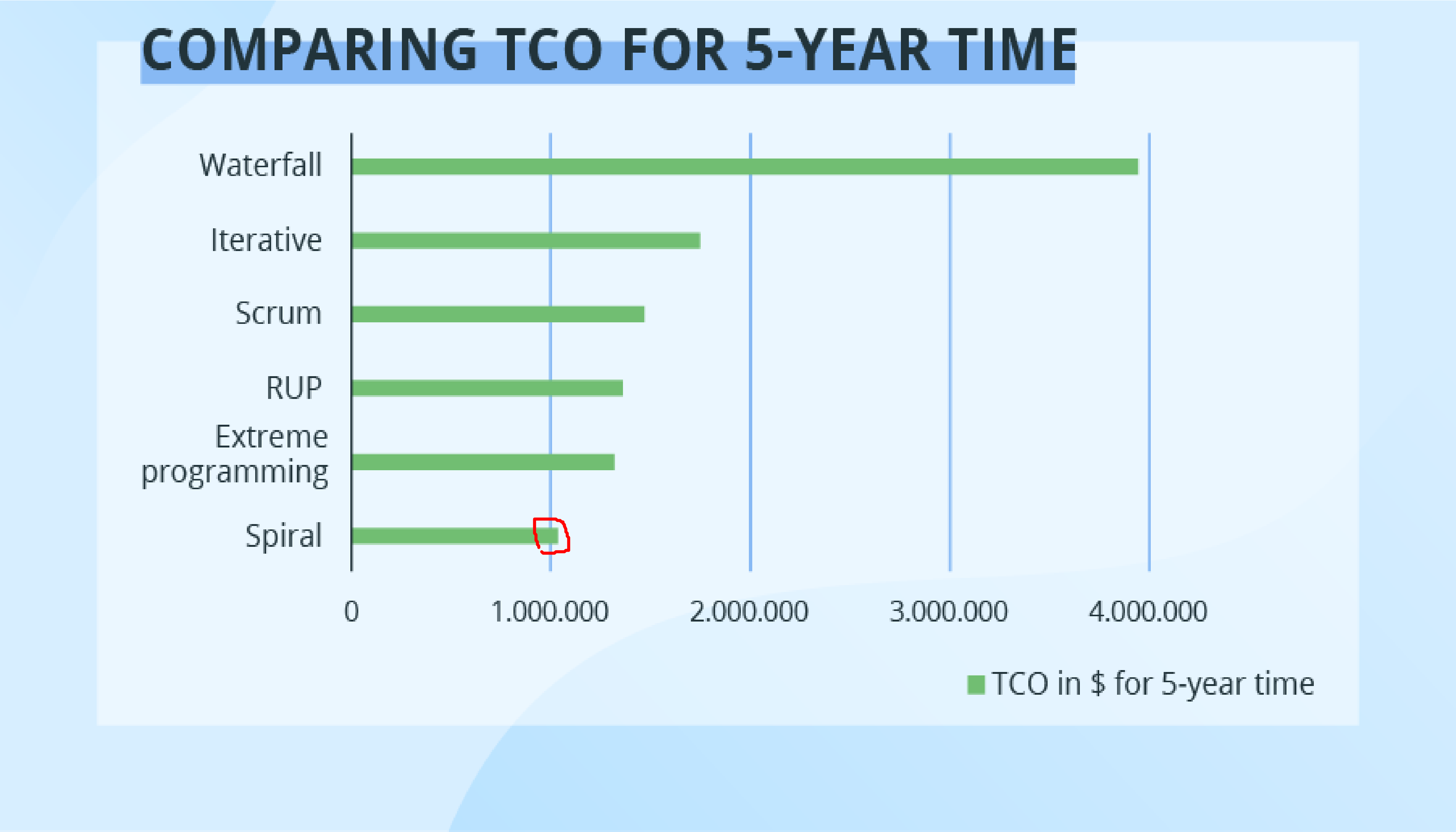


$\frac{\text{defects}}{\text{Loc}}$

Low error because of risk analysis, testing etc..

Maintaining Cost and ease of maintaining

Comparing all models


COMPARING TCO FOR 5-YEAR TIME
Waterfall
Iterative
Scrum
RUP
Extreme programming
Spiral
🟩 TCO in $ for 5-year time