Advanced Software Engineering

CSE608
Software Development Life-Cycle Models

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

Each Phase has an "Output"

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

Models

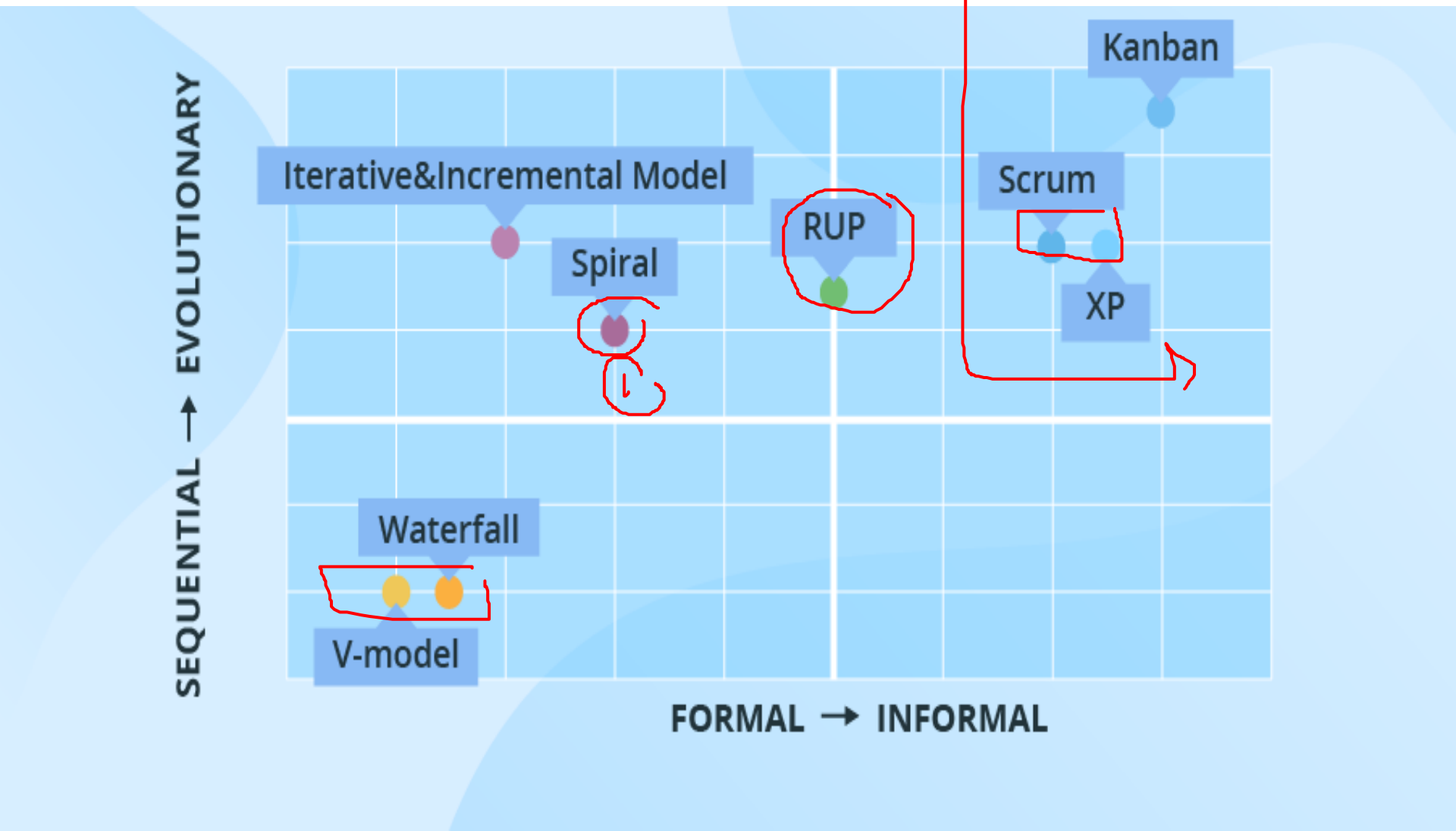
- Different projects may interpret these phases differently.
- Each particular style is called a

  "Software Life-Cycle Model"

"Life-Cycle" Models

- Single-Version Models
- Incremental Models
  - Single-Version with Prototyping
- Iterative Models

SLDC models

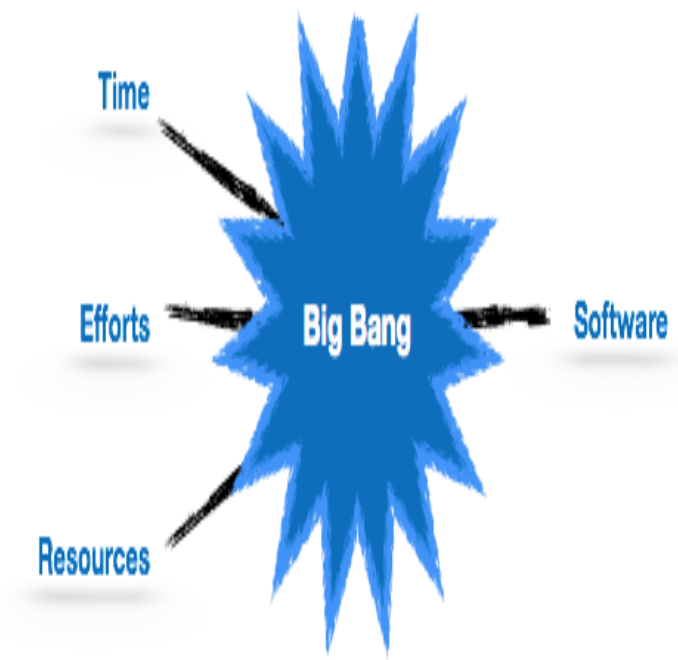

Kanban
Iterative&Incremental Model
Scrum
RUP
Spiral
XP
Waterfall
V-model
SEQUENTIAL → EVOLUTIONARY
FORMAL → INFORMAL

“Life-Cycle” Models (1)

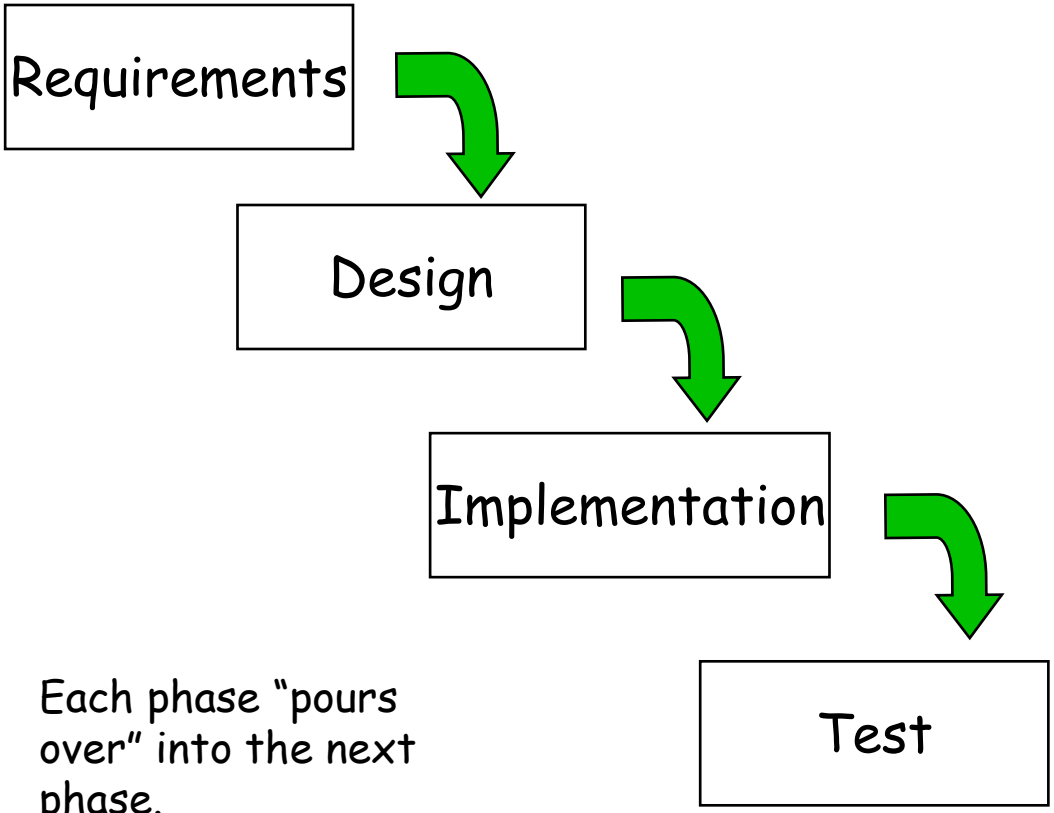
- Single-Version Models
  - Big-Bang Model
  - Waterfall Model
    - Waterfall Model with “back flow”
  - “V” model: Integrating testing

Big-Bang Model

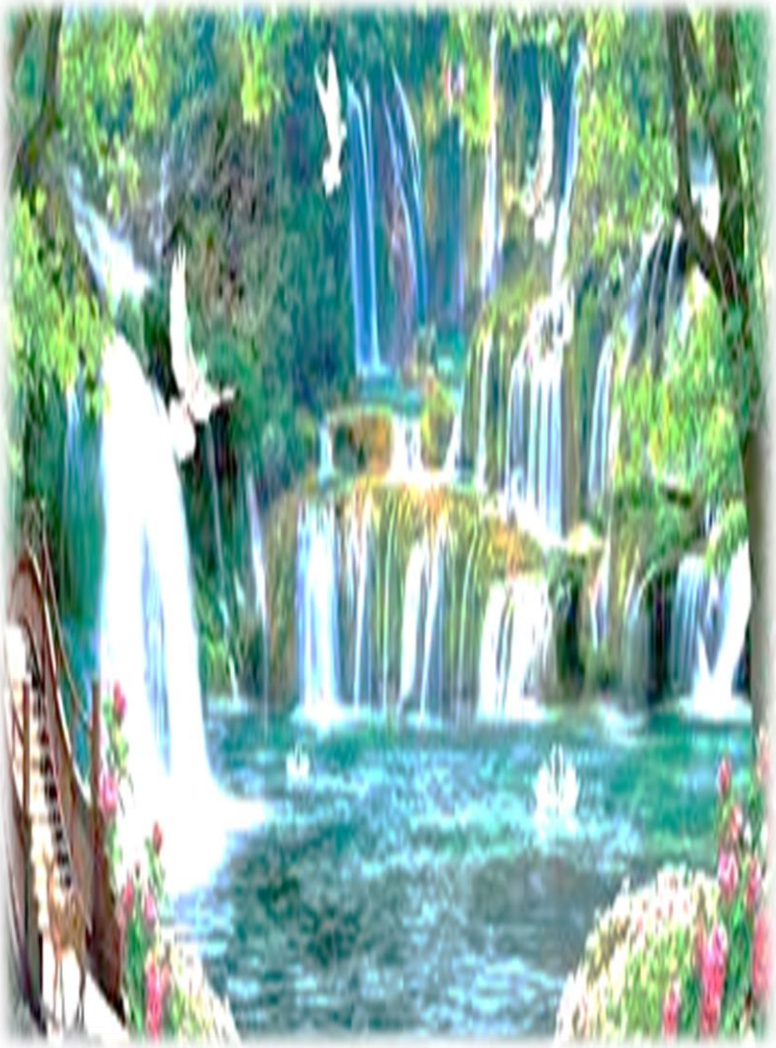
- Developer receives problem statement.
- Developer works in isolation for some extended time period.
- Developer delivers result.
- Developer hopes client is satisfied.



Waterfall Model

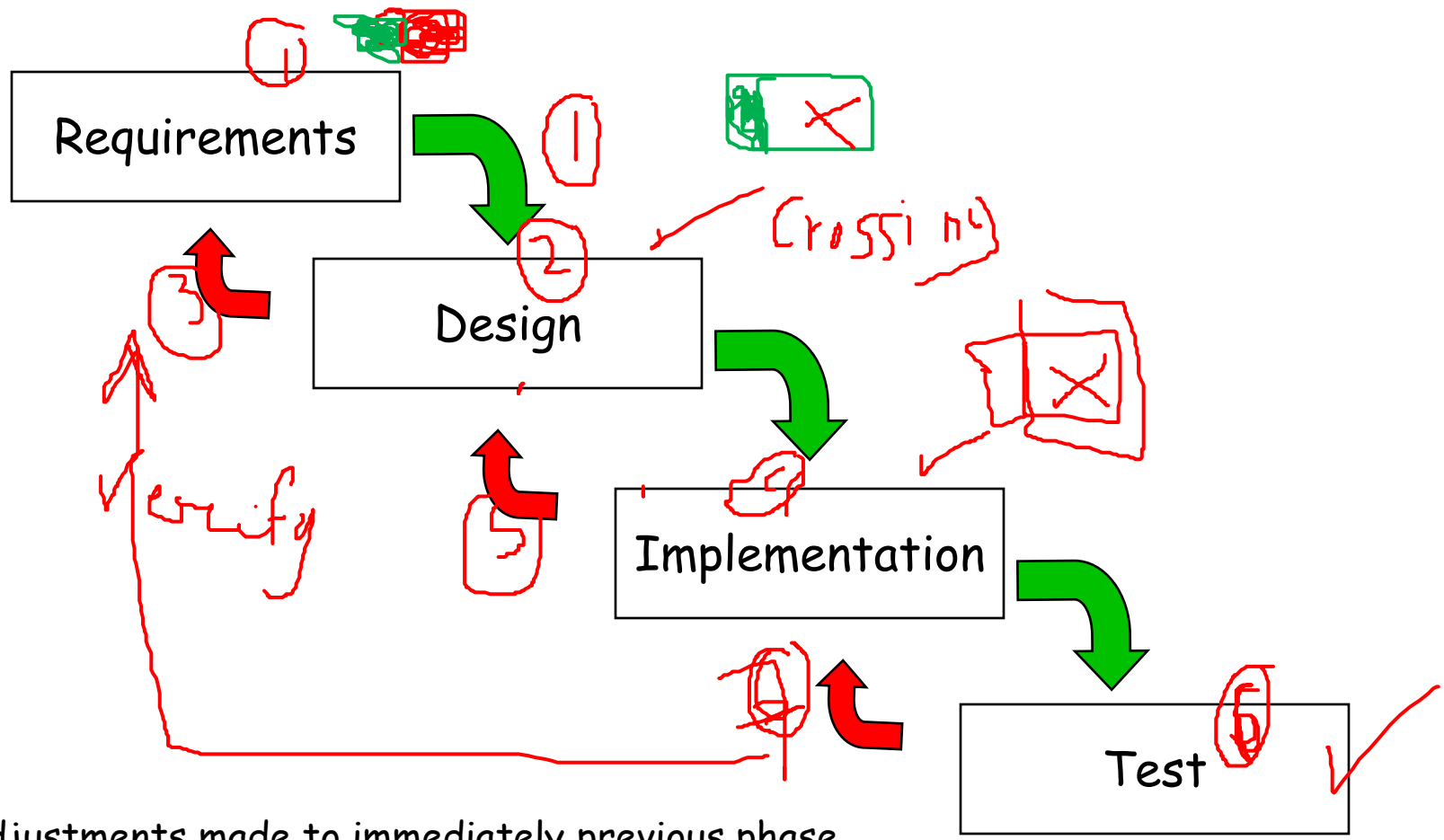


Each phase "pours over" into the next phase.



Waterfall Model with Back Flow

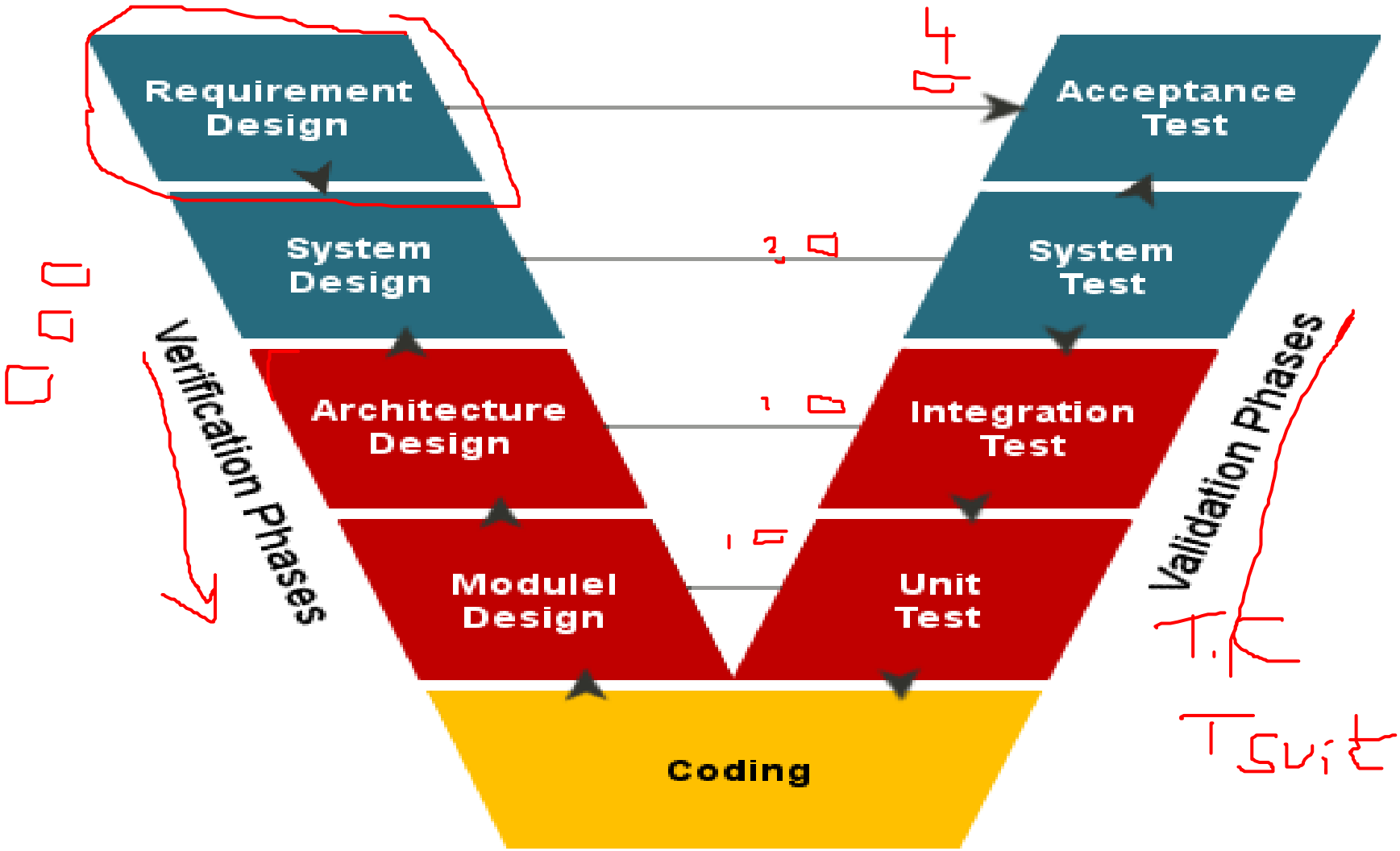
(sometimes this is implied by "waterfall")



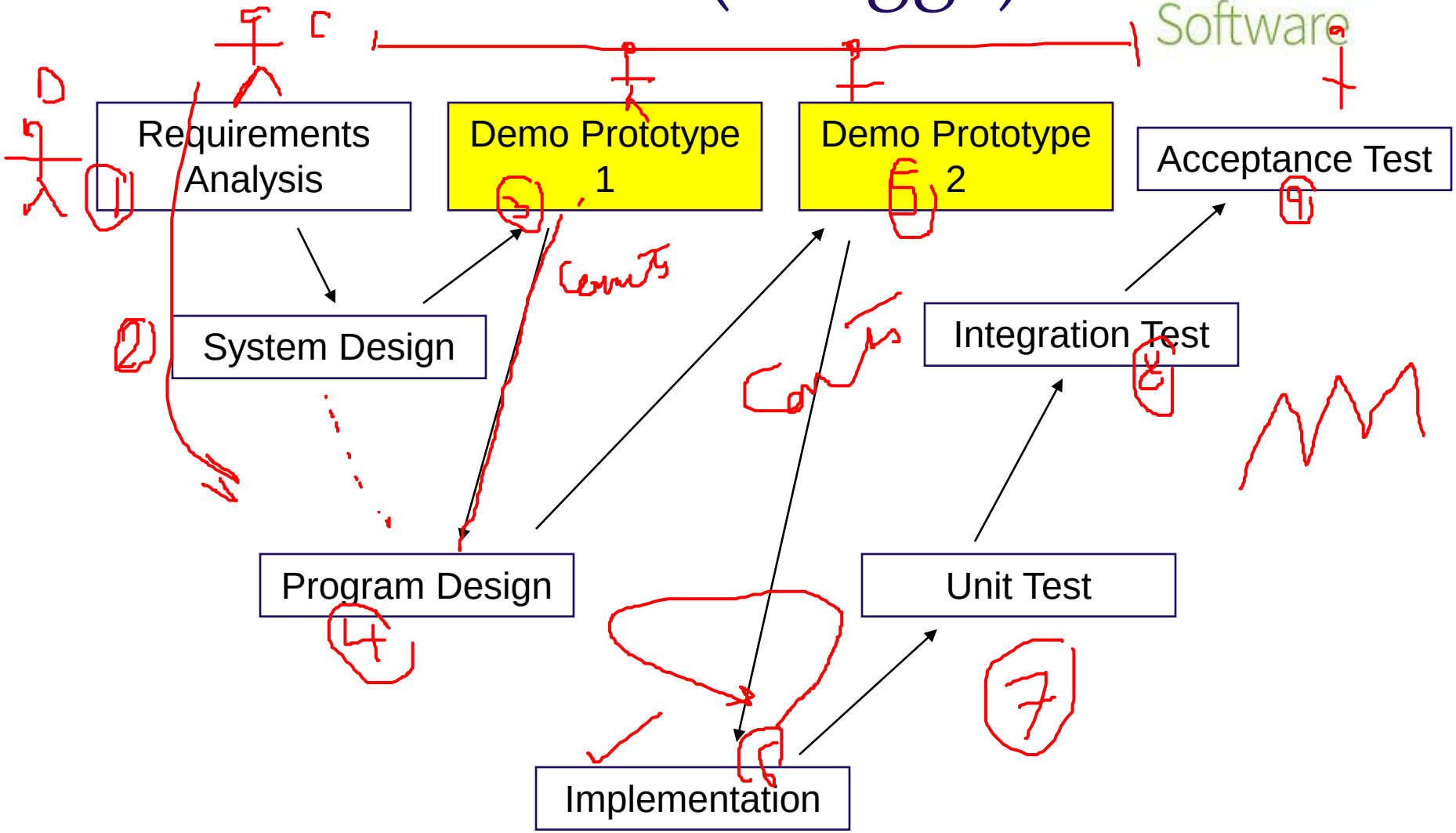
Adjustments made to immediately previous phase based on issues with successive phase.

"V" Model

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



Sawtooth Model (Brugge)



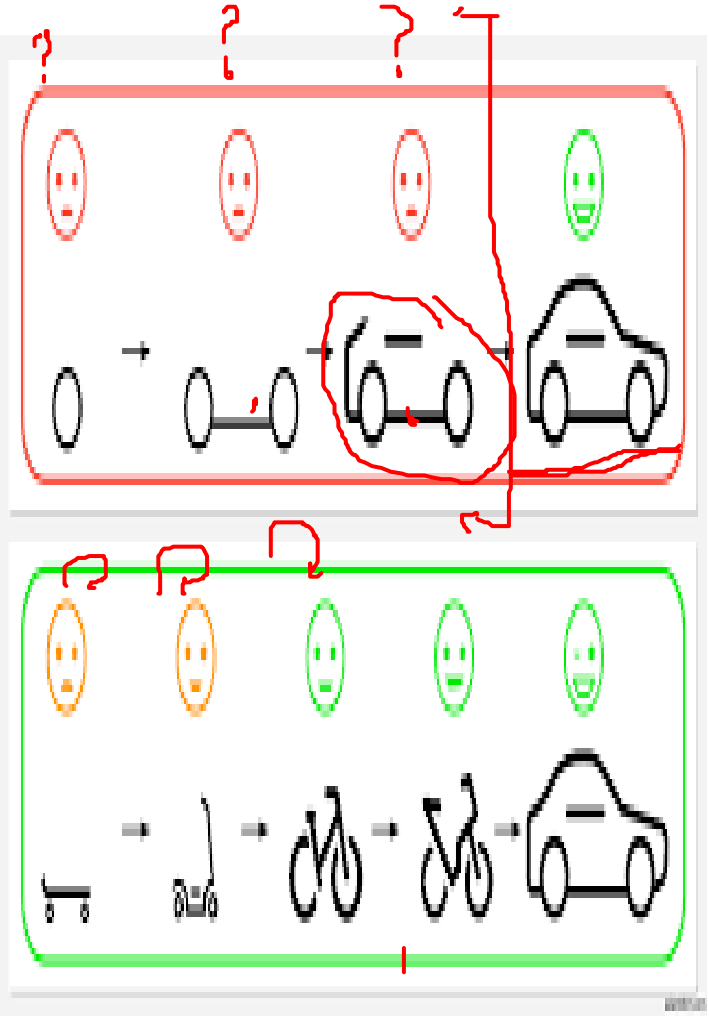
Incremental vs. Iterative

f_1 f_2 f_1' f_2' f_3

- 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

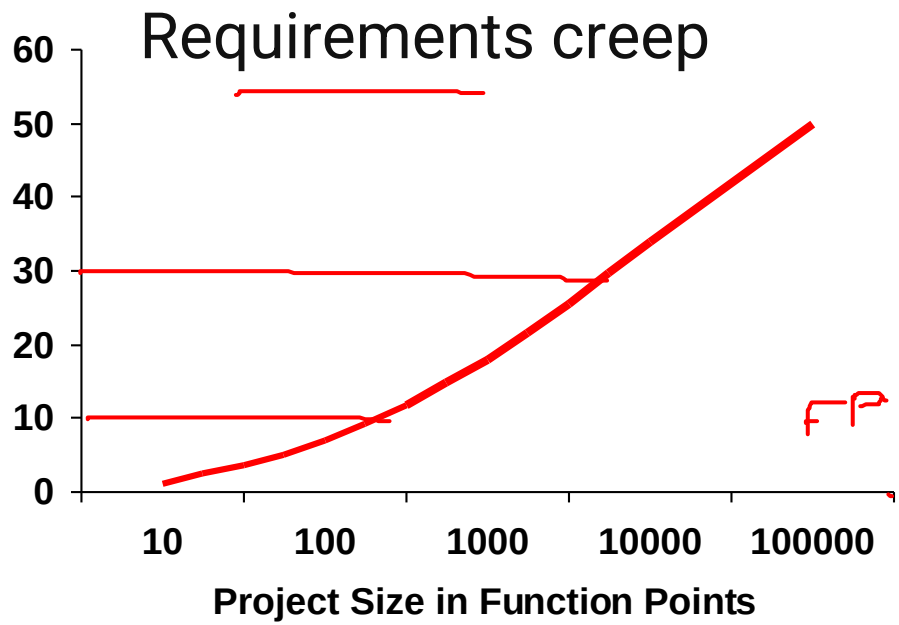
Example: Building a House

- **Incremental**: Start with a modest house, keep adding rooms and upgrades to it.
- **Iterative**: On each iteration, the house is re-designed and built anew.
- Big Difference: One can live in the incremental house the entire time! One has to **move** to a new iterative house.

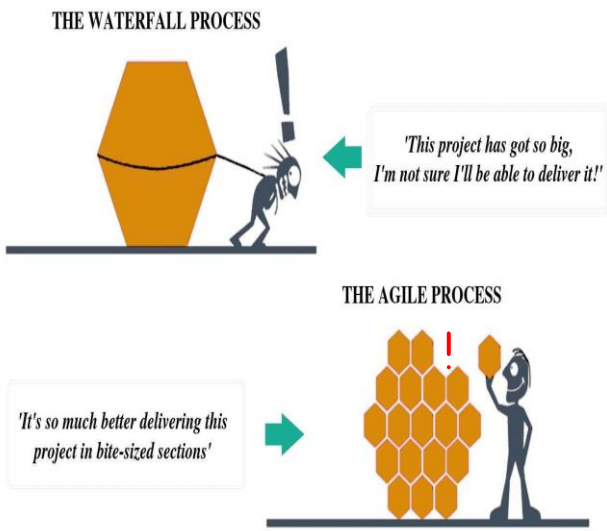


Why Not Waterfall?

1. Complete Requirements Not Known at Project Start



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



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

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.

Source: Craig Larman

Why Not Waterfall?

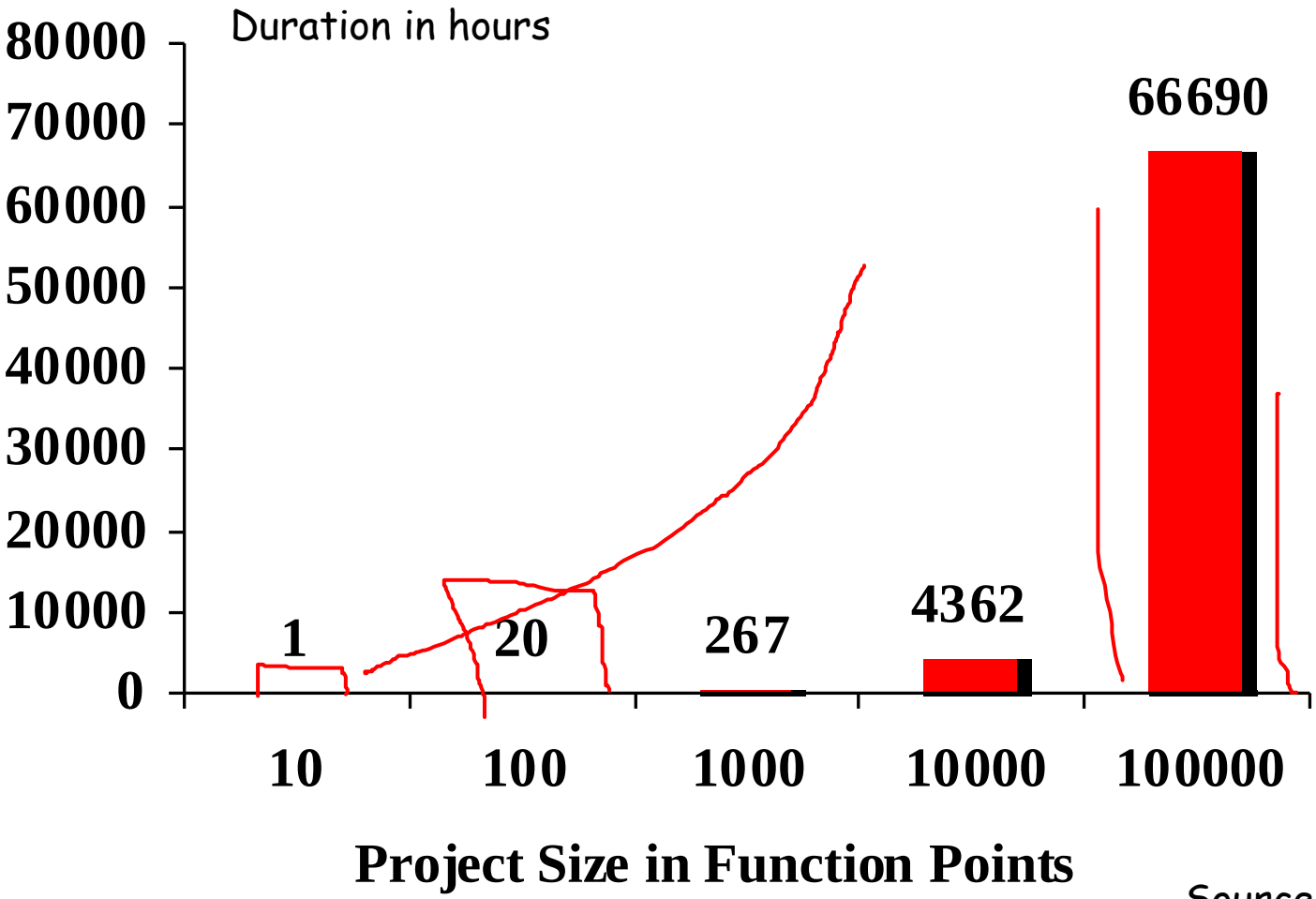
3. The design may need to change during implementation.

- 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

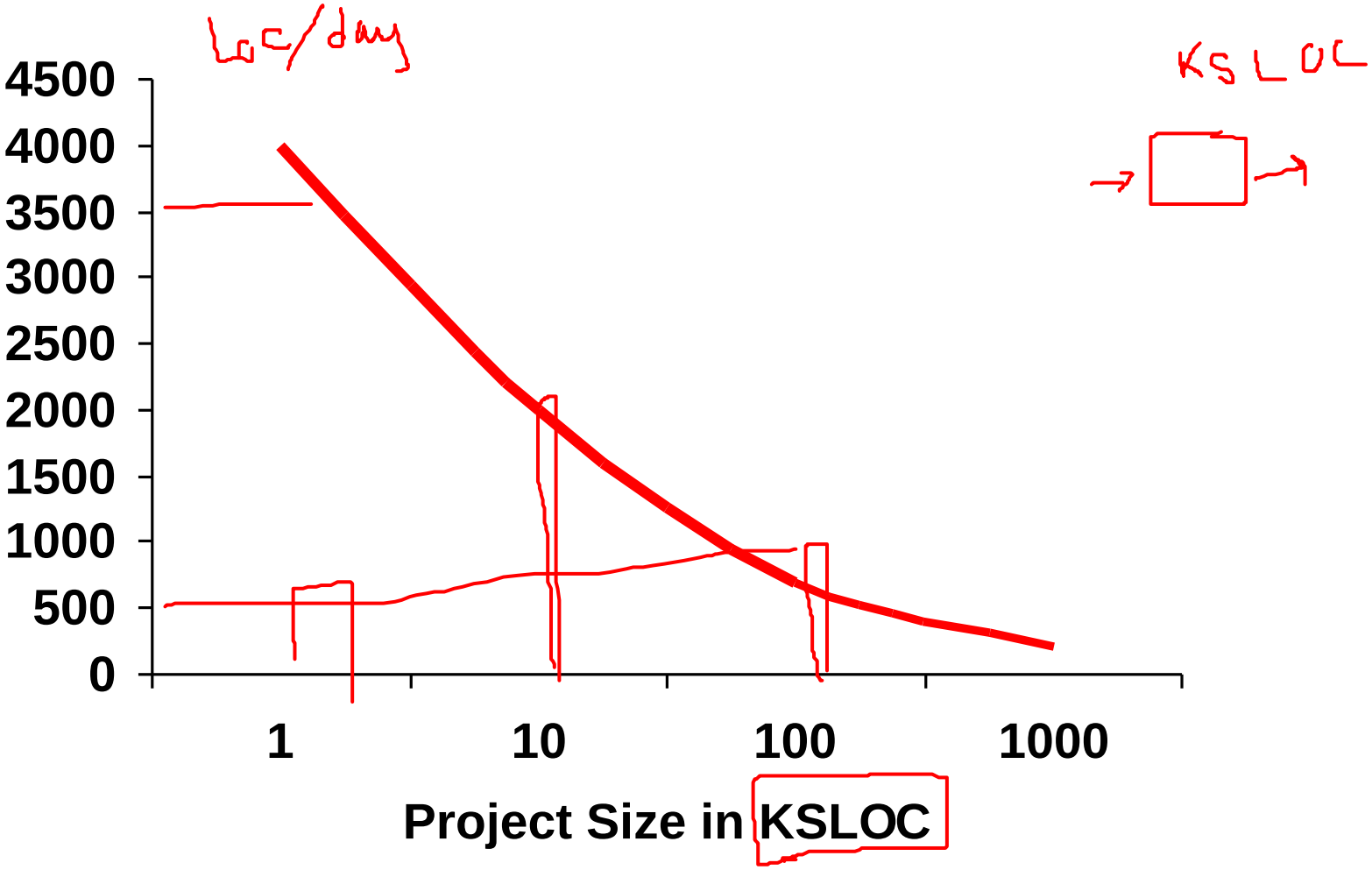
Large vs. Small Steps:

Project Duration



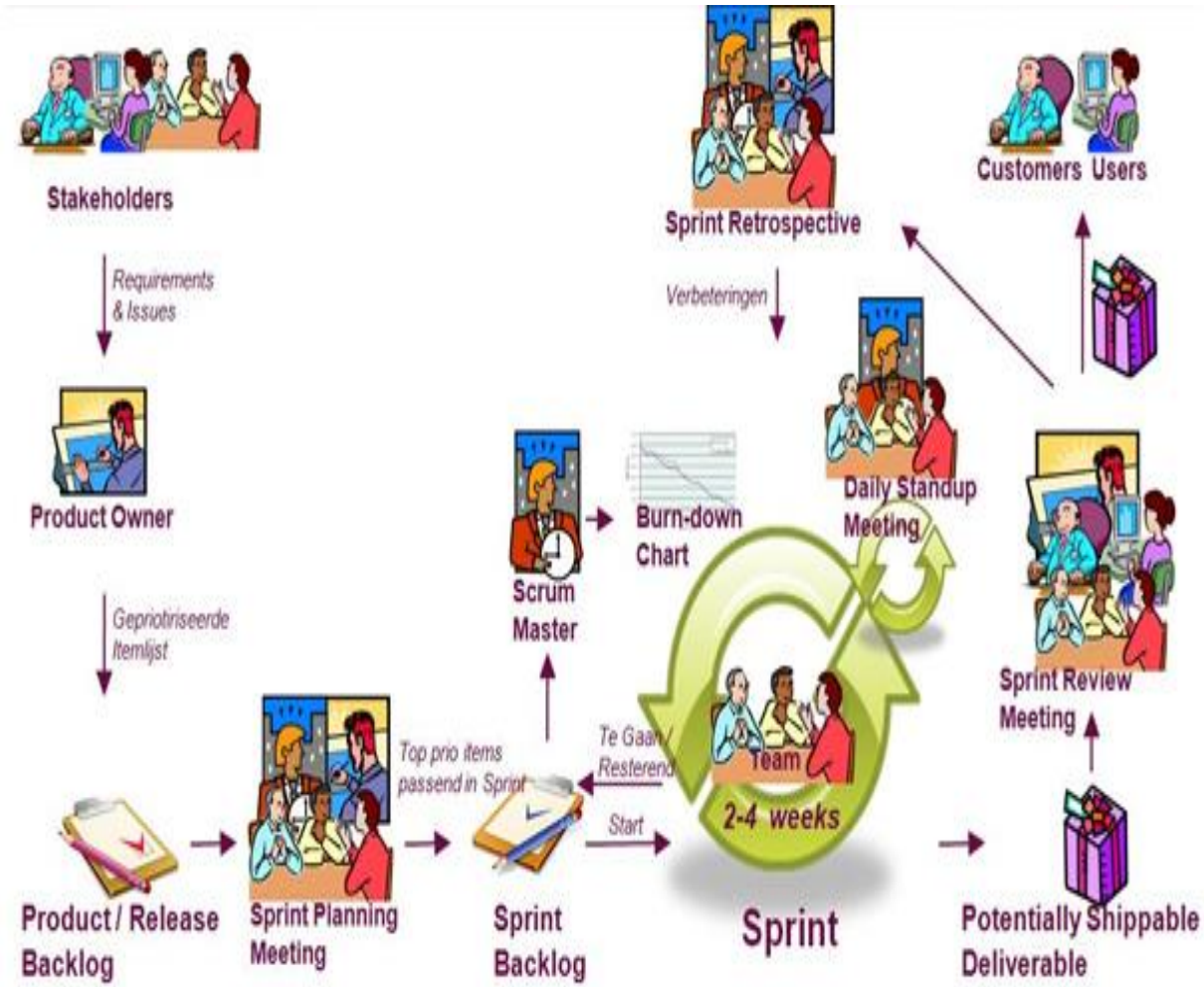
Source: Craig Larman

Large vs. Small Steps: Productivity



"Life-Cycle" Models (3)

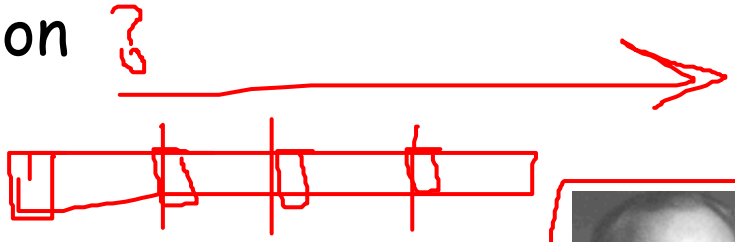
- **Iterative** Models
  - Spiral Model & Variants
    - ROPES Model
    - Controlled Iteration Model: Unified Process
    - Time Box Model
  - Scrum Model
    - Fountain Model

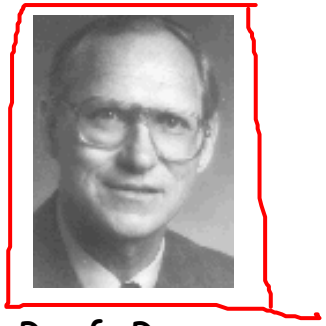


Boehm Spiral Model

(of which some other models are variants)

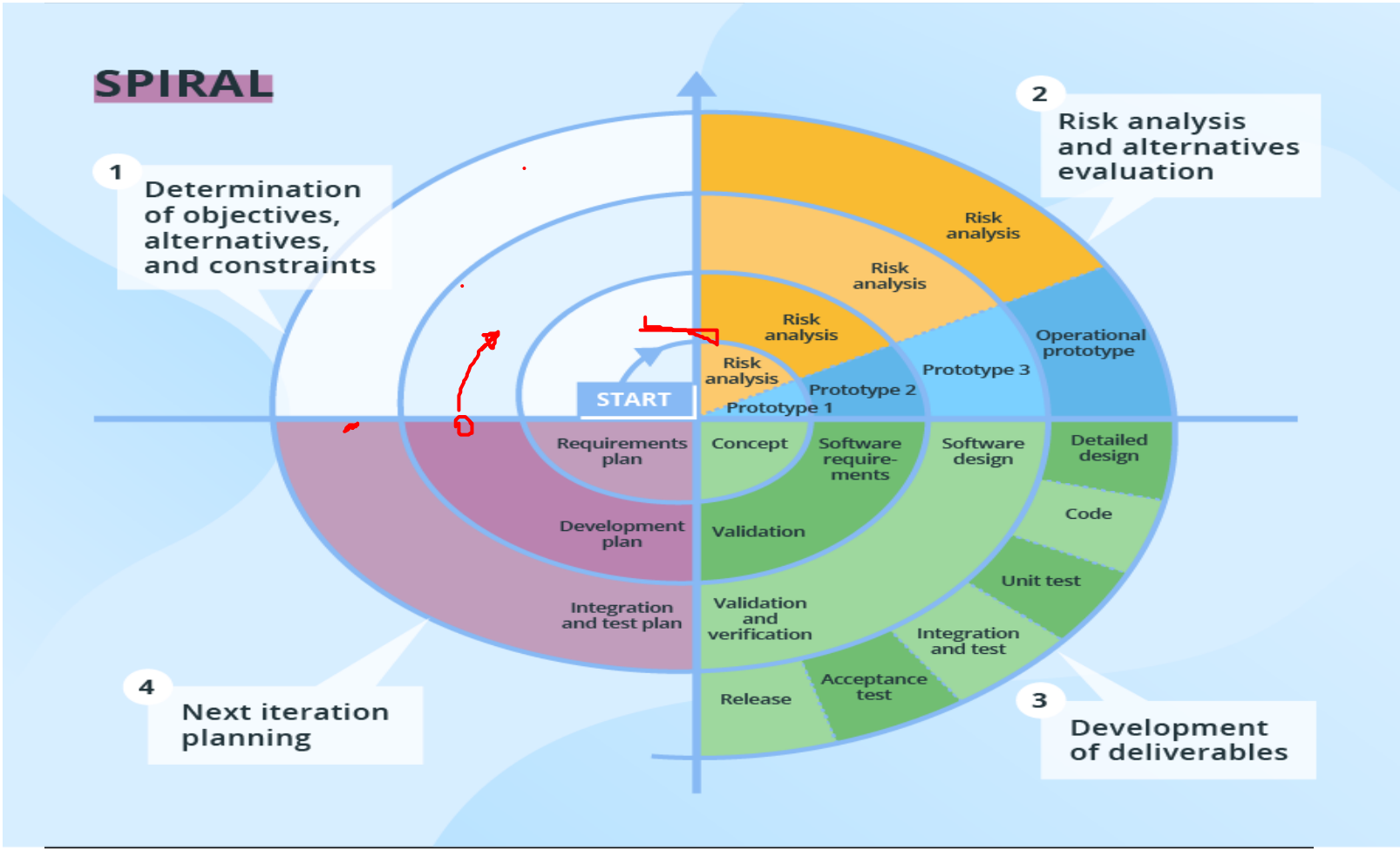
- An iterative model developed by Barry Boehm at TRW (1988), now Prof. at USC
- Iterates cycles of these project phases:
  1. Requirements definition
  2. Risk analysis
  3. Prototyping
  4. Simulate, benchmark
  5. Design, implement, test
  6. Plan next cycle (if any)





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?

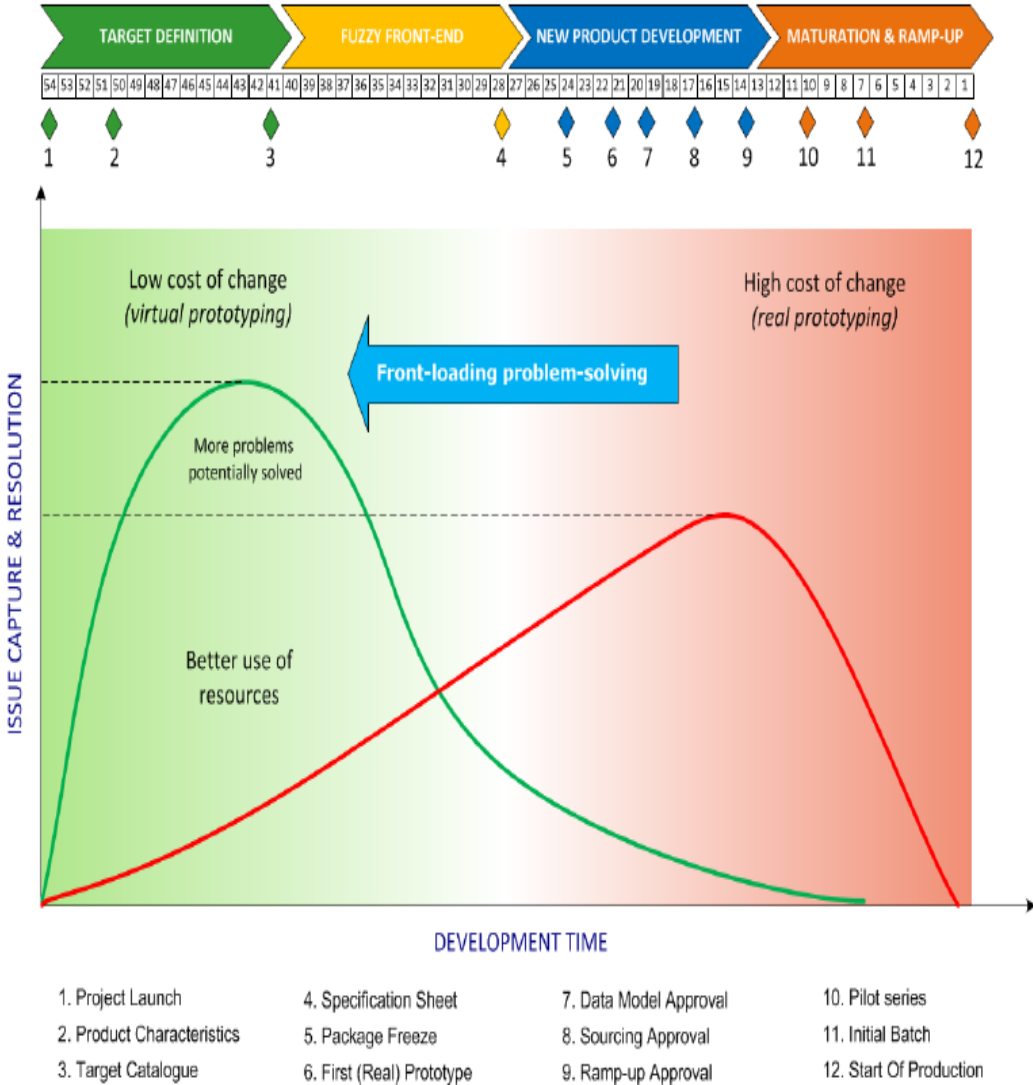
- One major area of risk is that the scope and difficulty of the task is not well understood at the outset.

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)

- Requirements analysis
- Initial design
- while(not done)

  {

  Develop a *version* **within a bounded time**

  Deliver to customer

  Get feedback

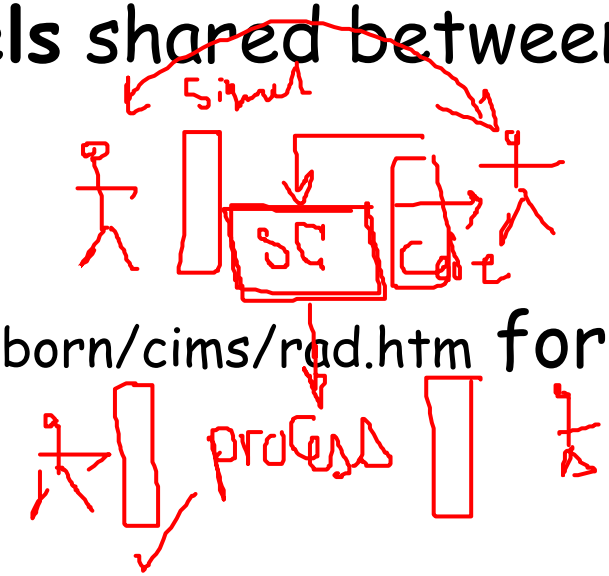
  Plan next version

  }

Additional Models/Acronyms

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

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

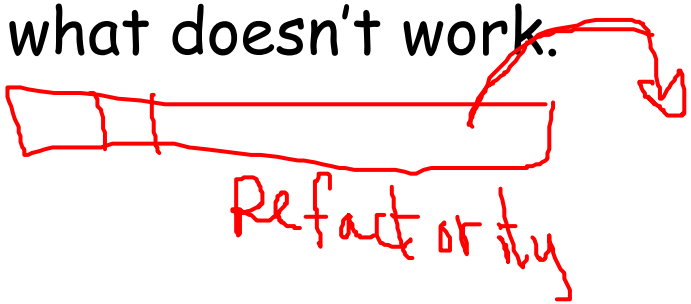


- See http://faculty.babson.edu/osborn/cims/rad.htm for additional points.

Extreme Programming (XP)

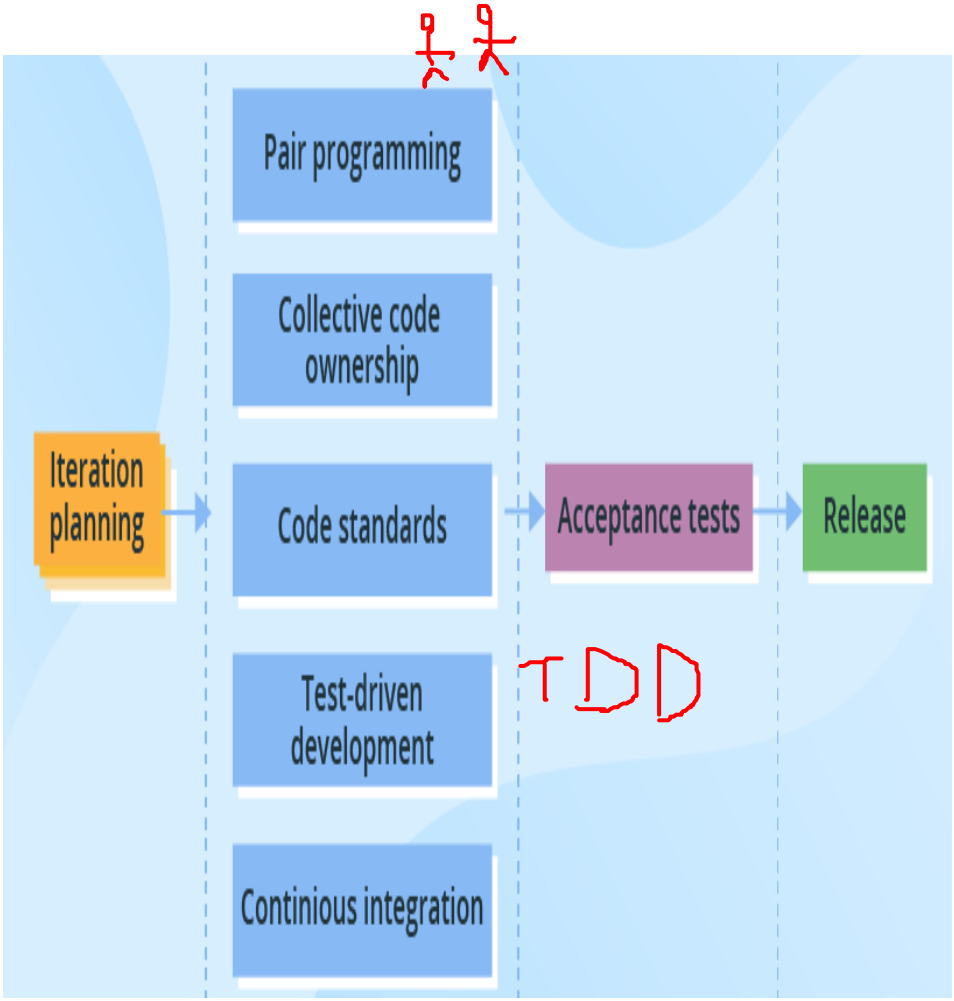
(cf. http://www.extremeprogramming.org/rules.html)

- 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.



Extreme Programming (XP)

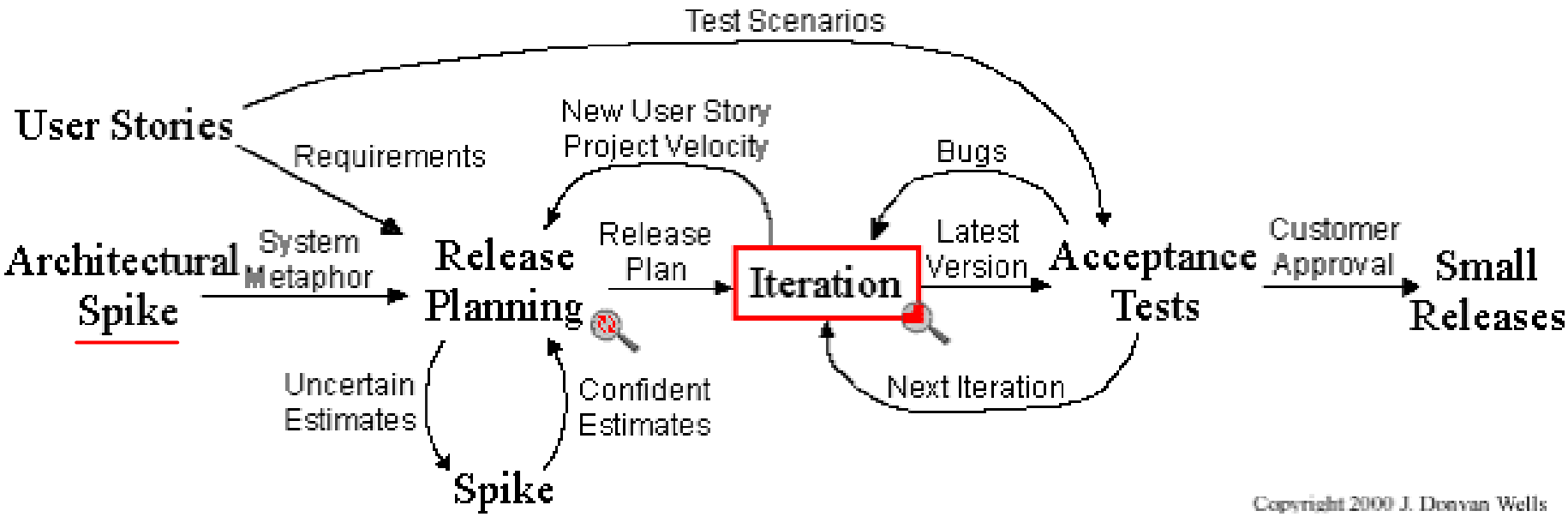
- Each release is preceded by a **release planning meeting**.
- Each day begins with a **stand-up meeting** to share problems and concerns.
- CRC cards are used for **design**. [XP and CRC were created by the same person, Kent Beck.]
- **Spike solutions** are done to assess risks.
- The **customer**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



Extreme Programming (XP)

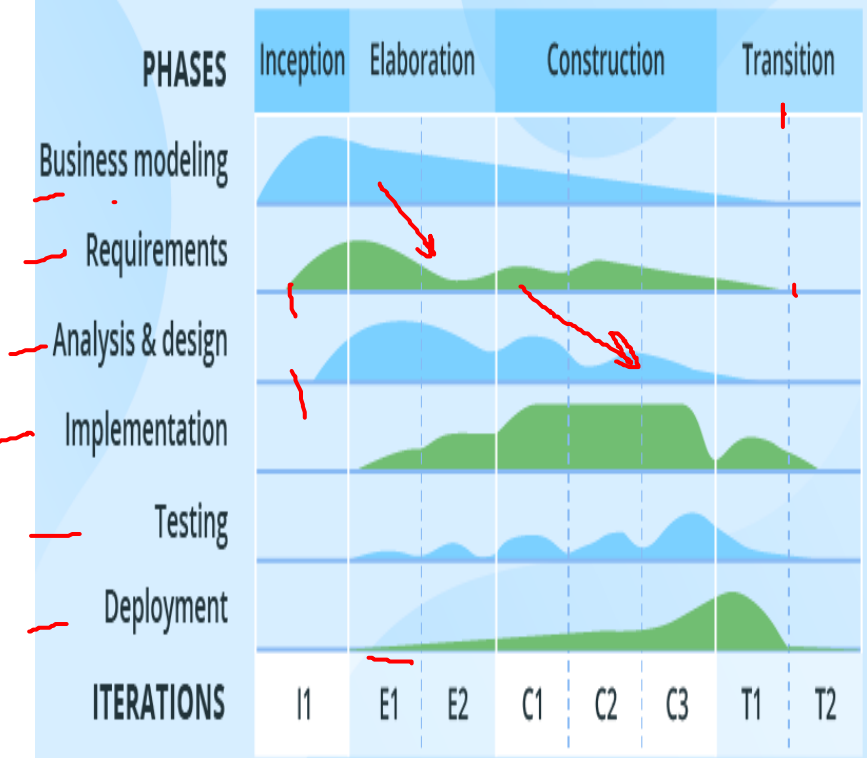
- All code must pass **unit tests**, which are coded before the code being tested (**test-driven design**).
- **Refactoring** is done constantly.
- **Integration** is done by one pair.
- Integration is done frequently.
- Optimization is done **last**.
- **Acceptance tests** are run often.

Extreme Programming Project



The rational unified process

- 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

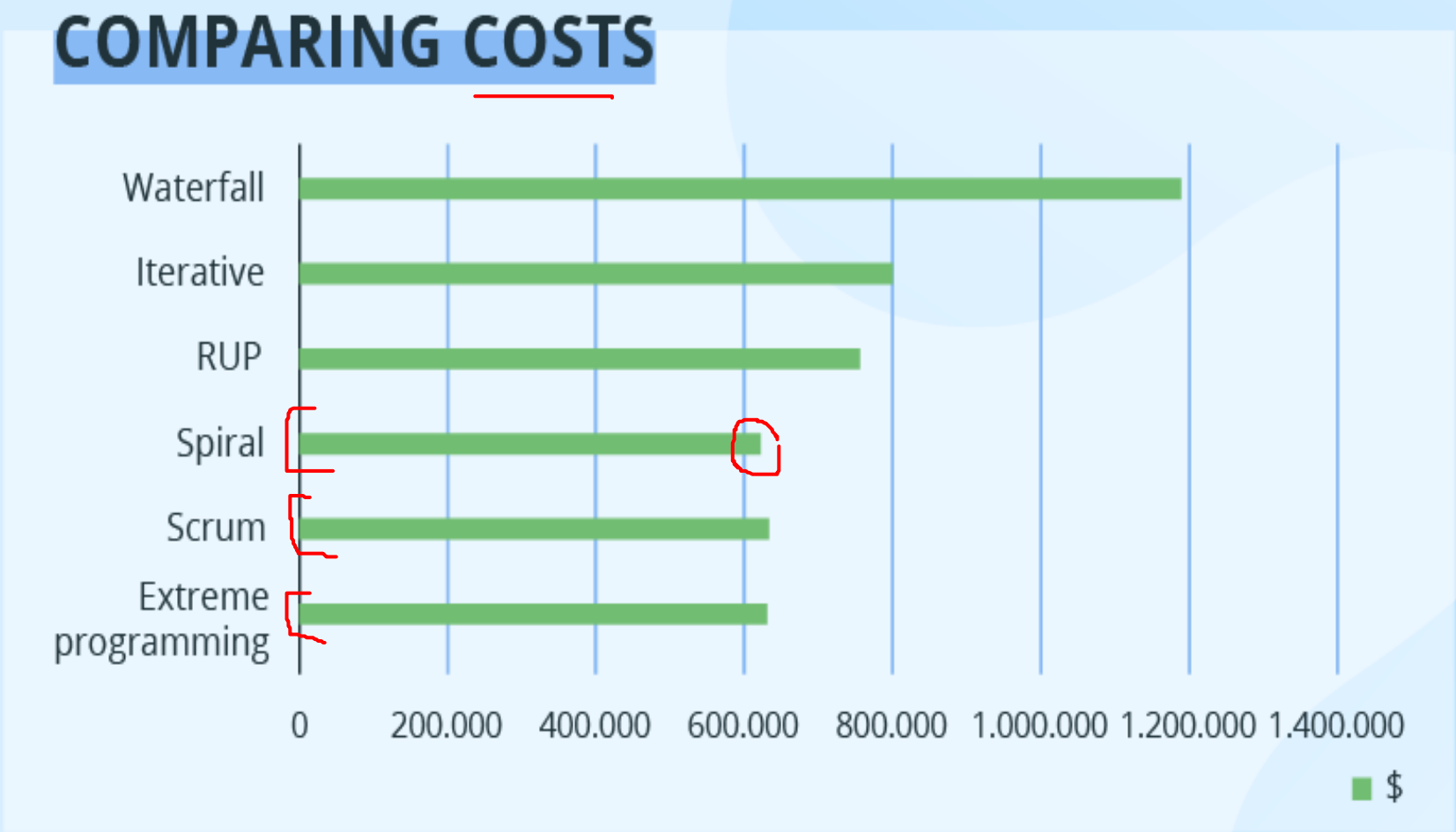


Large and high-risk projects, especially, use-case based development and fast development of high-quality software.

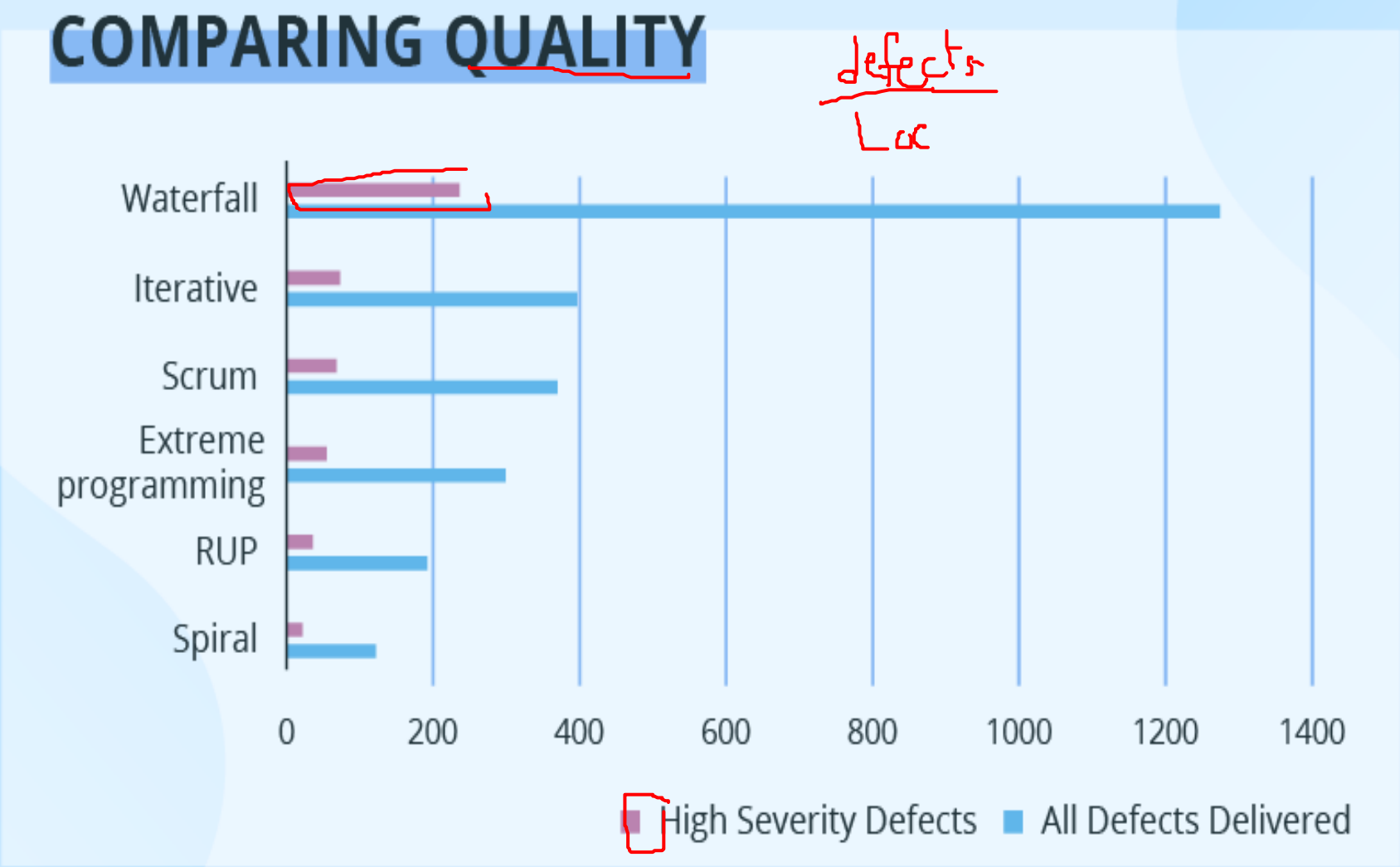
Kanban

- 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



Comparing all models