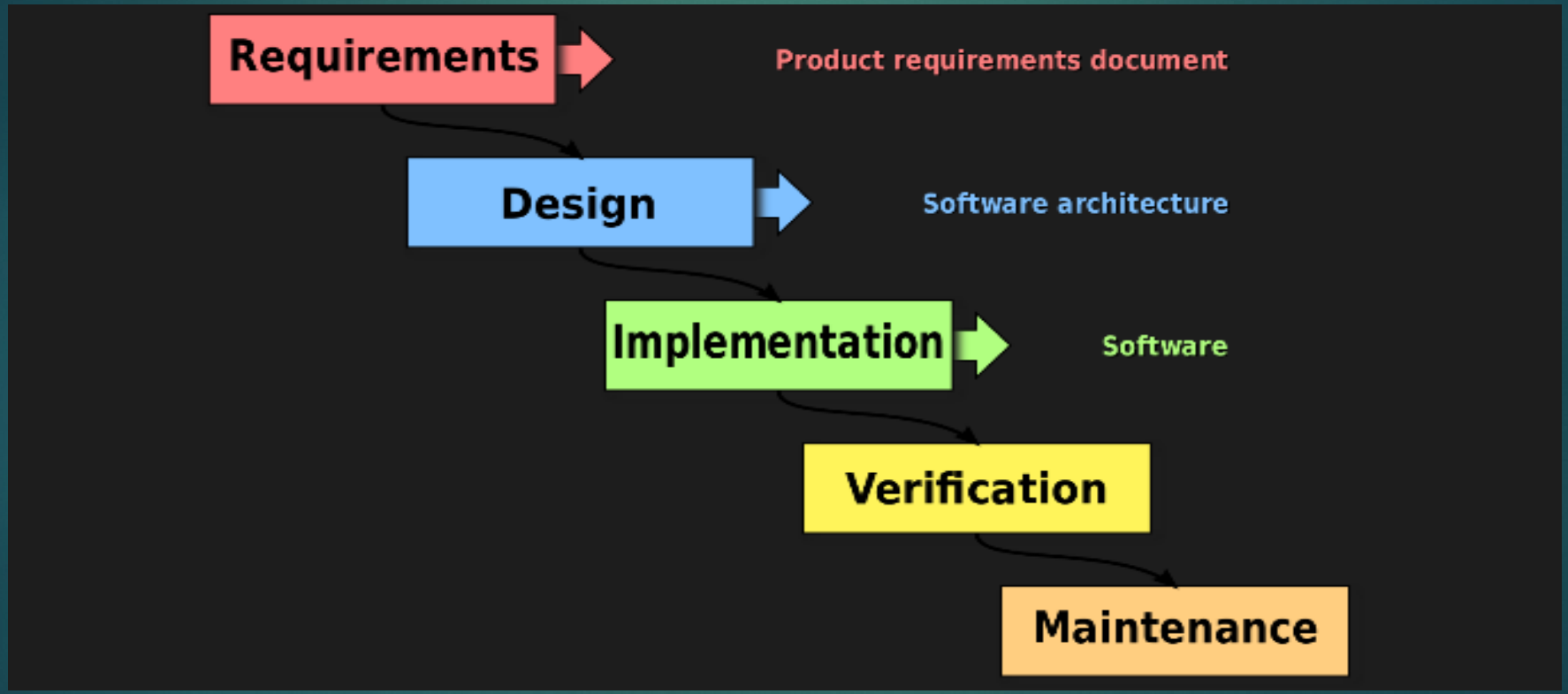
SW Lifecycle Models Some Examples

- Waterfall model
- Boehm's Risk Spiral model
- Prototyping Paradigm
- Evolutionary (Incremental) Development
- Extreme Programming
- Agile Programming
- Synchronize-and-stabilize

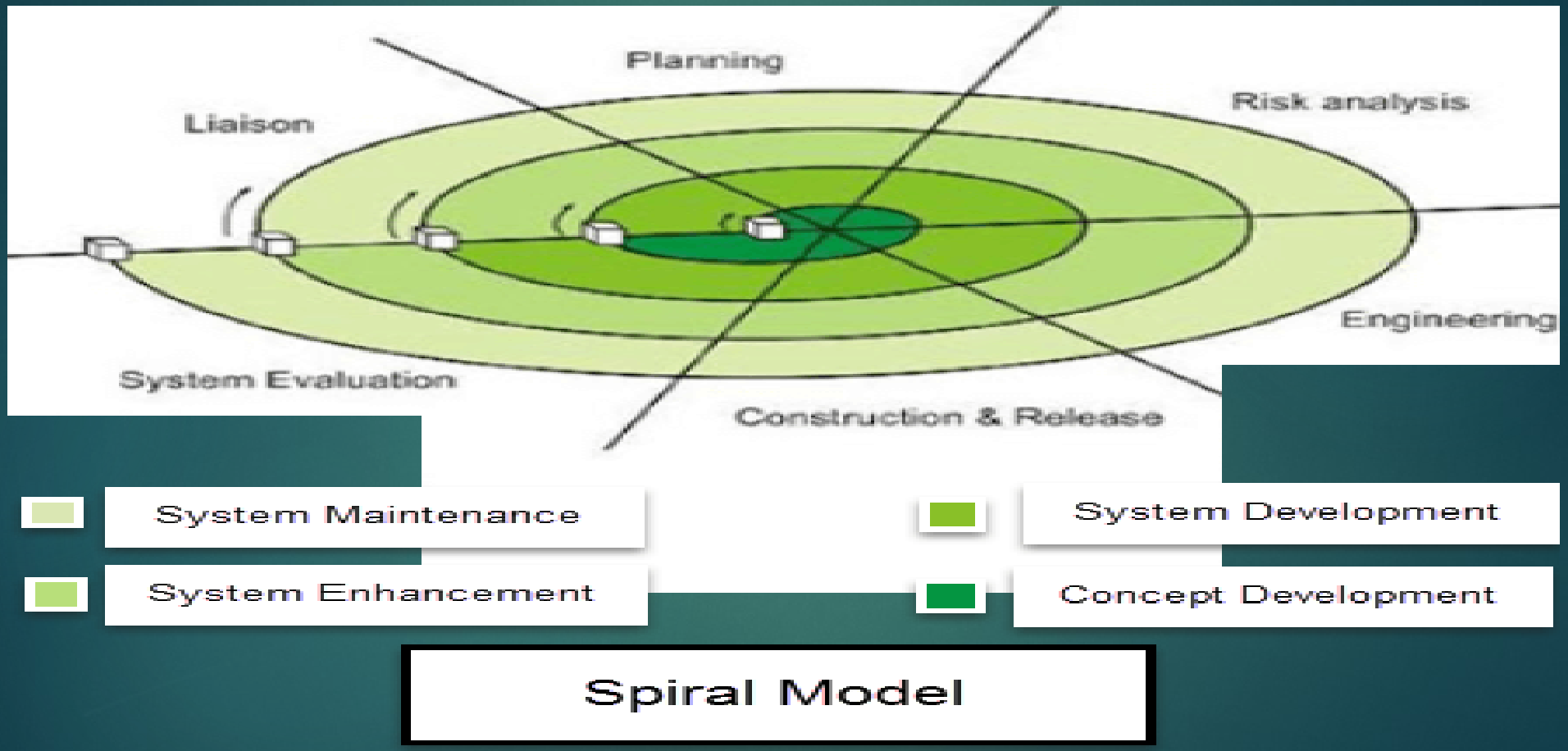
Waterfall model



Waterfall Model Issues

- To be effective there must be a review step at each phase to check for completion.
- In theory we should not proceed down the waterfall until the phase is "complete".
- **Problem**: real projects don't work this way – "change is the only constant"

Boehm’s Risk Spiral model



Boehm’s Spiral Model

- Essence: process contains several cycles, starting from the centre; at each iteration the tasks involve more detailed knowledge and design tasks.
- Focus on addressing risks incrementally, in order of priority
- Distance from origin represents cost accumulated by the project
- Angle from the horizontal represents the type of activity (e.g. risk, risk analysis, testing)

Spiral Model: Key Assumptions

- Assume sufficient flexibility to adapt and change after each analysis spiral
- Sensitive to having a thorough risk assessment at each cycle.
- Requires confidence in the project managers to carry it off, and to maintain the confidence of both the development team and the client!

A Requirements Capture Dilemma

- Where does these leave both the customer and the developer?
- They both think they know what is required but either do not know how to do it or cannot be sure of what is really required.

This might be described as the software engineer's dilemma!

We don't know what we want because we have never had one before ...
... but IKIWISI !
We don't know how to build one because we have never written one before.
CUSTOMER
SOFTWARE DEVELOPER
IKIWISI = I'll Know It When I See It

Prototyping (a solution to IKIWISI)

- Developing some high level strategies, and some simple frameworks for testing out the concepts that the customer wants and the developer can implement.
- From here both are able to respond with suggestions for improvements and changes. The cycle may then be able to be repeated, each time progressing closer to the required solution.
- This iterative process is called prototyping.

Disadvantages of the Prototyping Model

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- Poor documentation because the requirements of the customers are changing.
- It is very difficult for software developers to accommodate all the changes demanded by the clients.
- The client may lose interest in the final product when he or she is not happy with the initial prototype.

Evolutionary Paradigm (Gilb)

"Grow, don't build software." - Fred Brooks

Step 1: Prototype

- produce a useable but small part of the system
- solidifies user requirements
- sketch of system design

Step 2: Expansion

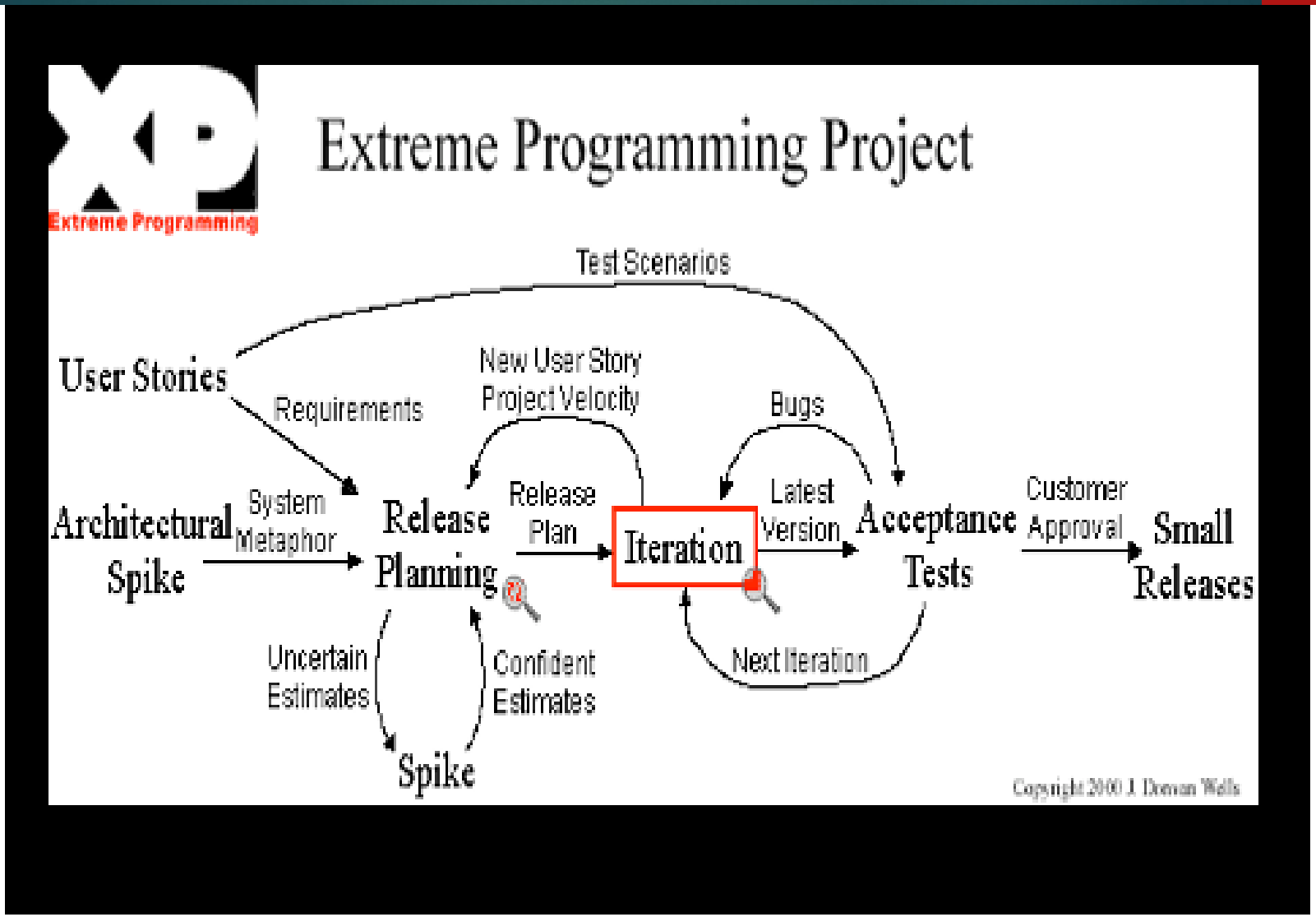
- add functionality
- determine "hot-spots" .

Step 3: Consolidation

- correct design defects
- introduce new abstractions

Extreme Programming

- Extreme Programming (XP) stresses customer satisfaction and emphasizes team work.
- XP principles: communication, simplicity, feedback, and courage.
- Feedback = testing SW from day one; deliver to customers as early as possible; implement changes as suggested.
- With this foundation XP programmers are able to respond courageously to changing requirements and technology.


Extreme Programming Project
Extreme Programming
Test Scenarios
User Stories
Requirements
New User Story
Project Velocity
Bugs
Architectural
Spike
System
Metaphor
Release
Planning
Release
Plan
Iteration
Latest
Version
Acceptance
Tests
Customer
Approval
Small
Releases
Uncertain
Estimates
Confident
Estimates
Next Iteration
Spike
Copyright 2000 J. Donan Wells

Agile Programming

Agile software development refers to software development methodologies centered round the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

Underlying Assumptions for Agile SW Development

- “Different projects need different processes or methodologies”
- “Focusing on skills, communication and community allows the project to be more effective and more agile than focusing on process”

Synchronize-and-stabilize

- Teams work in parallel on individual application modules
- Teams frequently synchronize their code with that of other teams
- Debug (stabilize) code regularly throughout the development process
- Flexible since allows for changes at any point
- Successfully used by Microsoft and Netscape