The Waterfall Model – What It Is Not and Has Never Been

Dr. Peter Hantos
The Aerospace Corporation

14th Annual NDIA Systems Engineering Conference, San Diego, California
October 24-27, 2011

Acknowledgements

- This work would not have been possible without the following:
  - *Reviewers*
    - Asya Campbell
    - Suellen Eslinger
    - Zane Faught
    - Leslie Holloway
  - *Funding Source*
    - The Aerospace Corporation’s Sustained Experimentation & Research Program

Outline

- Motivation
- Parsing the Title
- The Canonical Waterfall Model
- Model Characterization
- What Does the Waterfall Life Cycle Really Look Like?
- Clarifying the Model's Intent
- "Hidden" Concurrency in the Waterfall
- Incremental Integration and Pair-wise Testing
- Scaling
- Conclusion
- Acronyms
- References

Motivation

- Your first reaction might be “Why are we talking about the waterfall? I thought that the Waterfall was dead”
  - *Indeed, since the Waterfall model was published, the literature has been full with critiques and proposals for alternative processes, most recently Agile Development*
- However, a substantial percentage of waterfall project failures can be attributed to lack of understanding of some fundamental issues, issues that are also present when modern methodologies are used
  - *Ignoring such issues will lead to project failure, regardless of the methodology that is used*
- Also, the waterfall (or, a “mini” waterfall) is still an essential building block of all the more complex life cycle models, such as incremental, evolutionary, spiral, or iterative incremental development (IID)

Parsing the Title-1: “Waterfall” is a metaphor

Glade Creek Spring Waterfalls

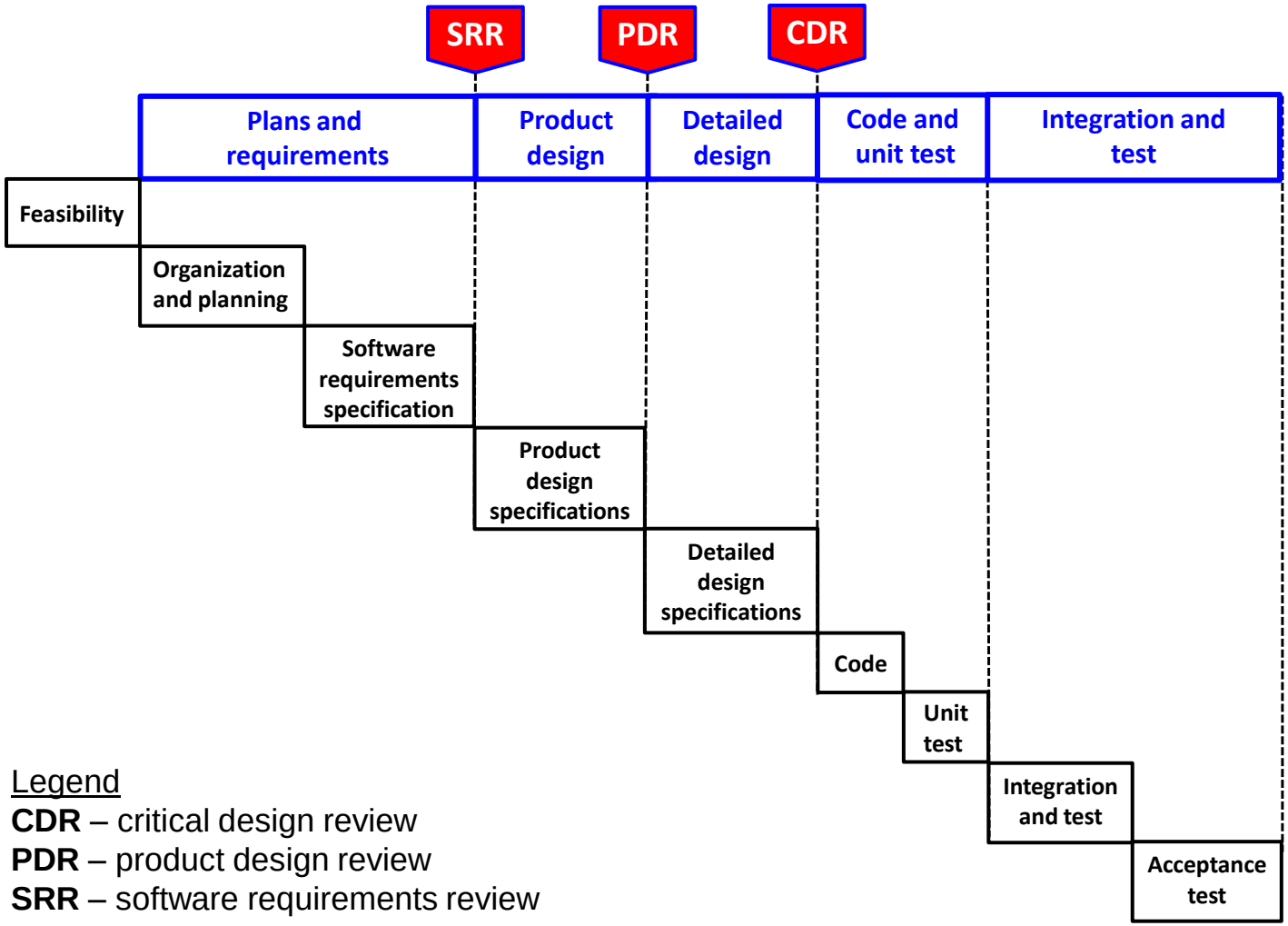
Parsing the Title-2: Model

- Definition of a model*
  - A model is always a reproduction of an original system
    - However, a model is an abstraction and does not reproduce all attributes of the original system
  - Models serve a certain purpose and they are to be used in a certain context
- What we are going to do
  - Start with the prevailing description of the Waterfall Model
  - Highlight the hidden abstractions of the model
  - Develop successively more complex descriptions to facilitate the understanding of underlying issues

All models are wrong. Some models are useful.
~~~ George Box

* [Stachowiak 1973]

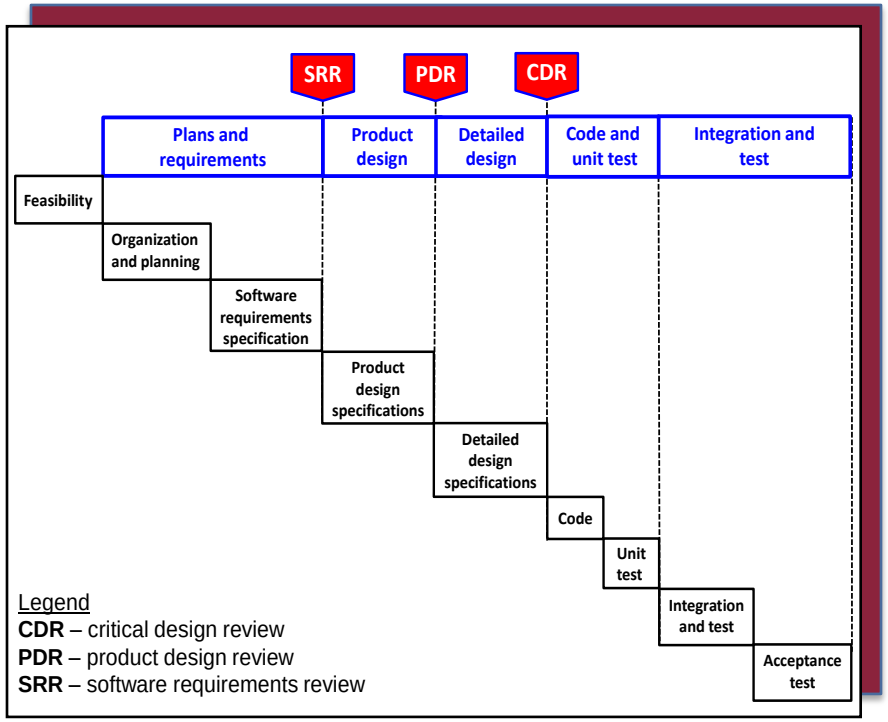
The Canonical Waterfall Model*



Legend
CDR – critical design review
PDR – product design review
SRR – software requirements review

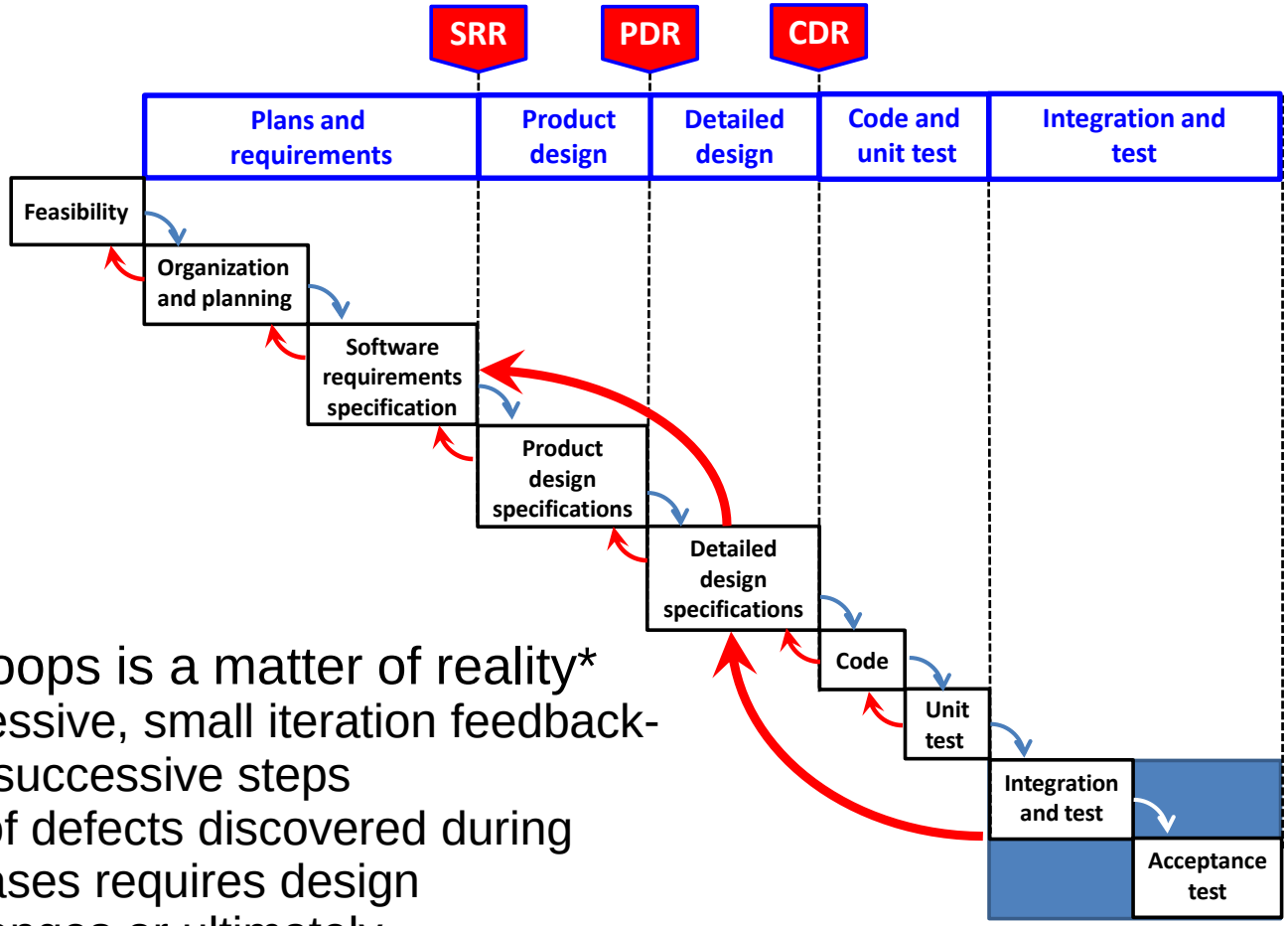
* [Boehm 1981]

Model Characterization



- Scaling: **No**
  - Seems to depict a macro view of software development
- Depiction of feedback loops: **No**
- Depiction of concurrency: **No**
- Presence of other technical disciplines: **No**
  - The “product” is software only

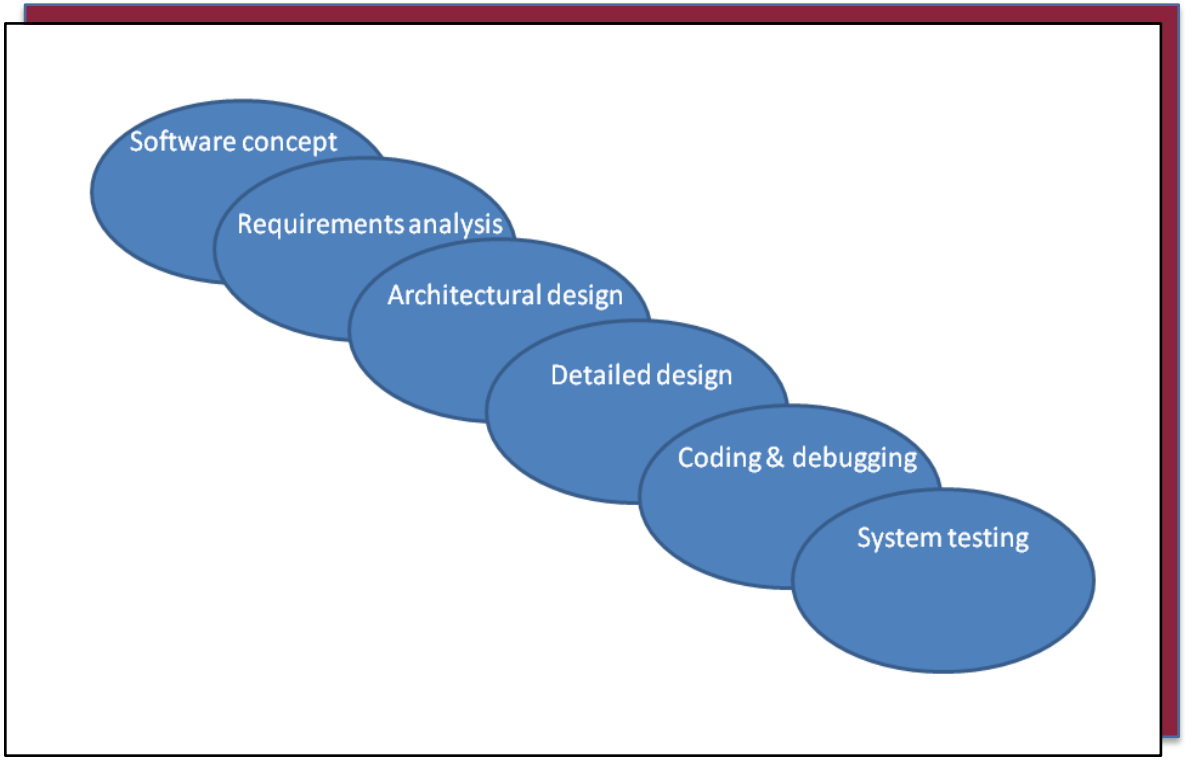
Hidden Feedback Loops



- Having feedback loops is a matter of reality*
  - There are successive, small iteration feedback-loops between successive steps
  - The correction of defects discovered during the final test phases requires design specification changes or ultimately, requirements specification changes
  - The basic interpretation of the metaphor (one-directional flow) needs to be challenged

* Based on [Royce 1970]

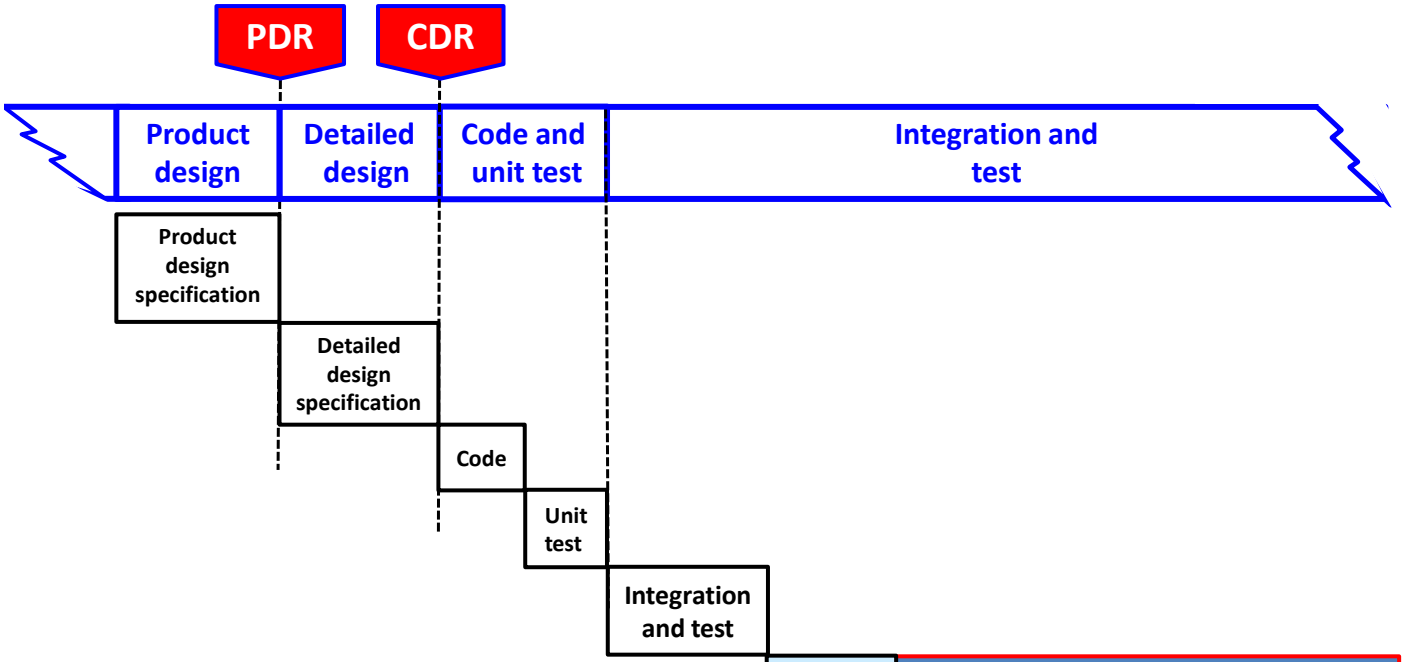
Phase Overlap Added – the Sashimi Model*



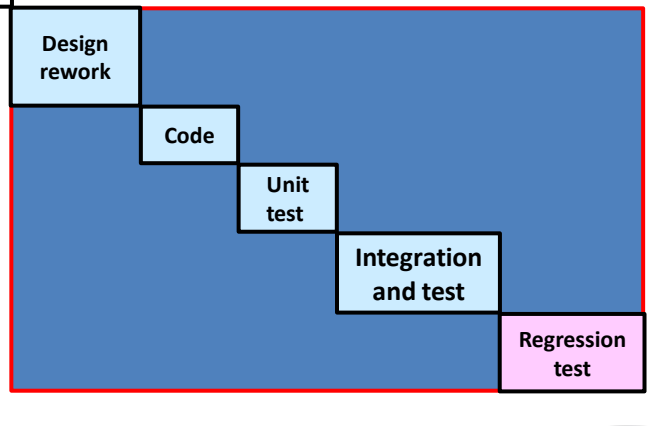
- Overlap is suggested between phases
  - This overlap can reduce belated problem discovery
- However, in most cases this is not really concurrency but iteration
  - Correcting design while coding is essentially iteration
  - It was shown earlier that to deal with defects such small-scale iteration is inherently present in the Waterfall model

* Source: [McConnell 1996]. Note his slightly more up-to-date terminology.

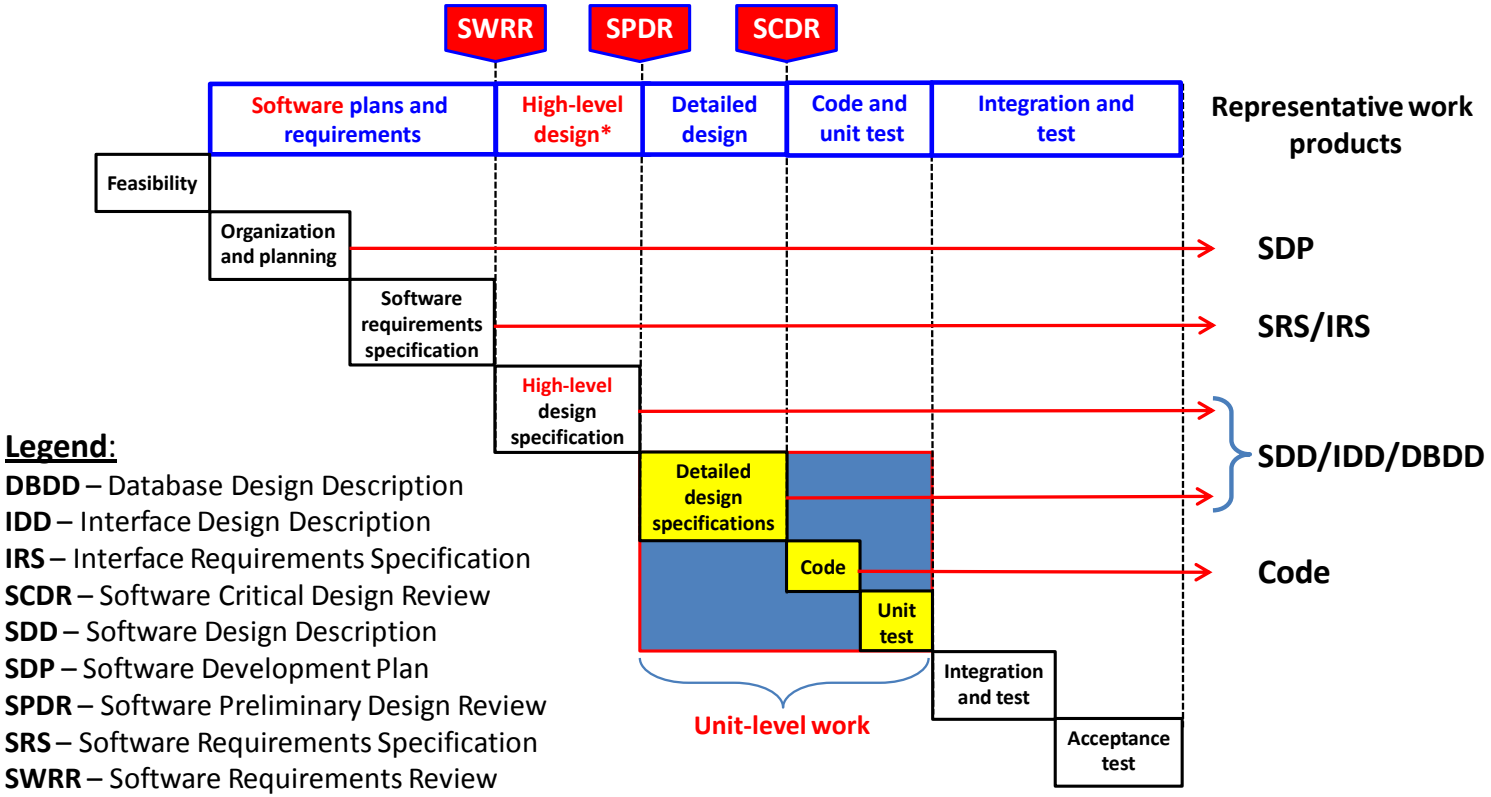
What Does the Waterfall Life Cycle Really Look Like?



- Feedback loops' impact on the life cycle
  - Activities don't neatly map into the life cycle phases anymore
  - Diagram does not even show the impact of failed integration and regression tests
  - Length of the "Integration and test" life cycle phase is becoming more uncertain



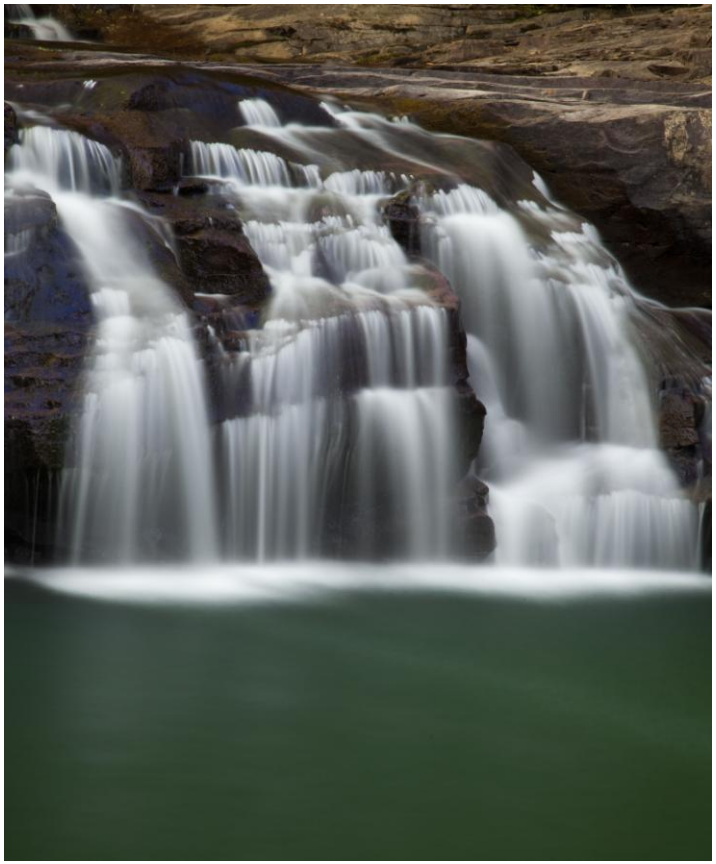
Selected Name Changes to Clarify the Model's Intent

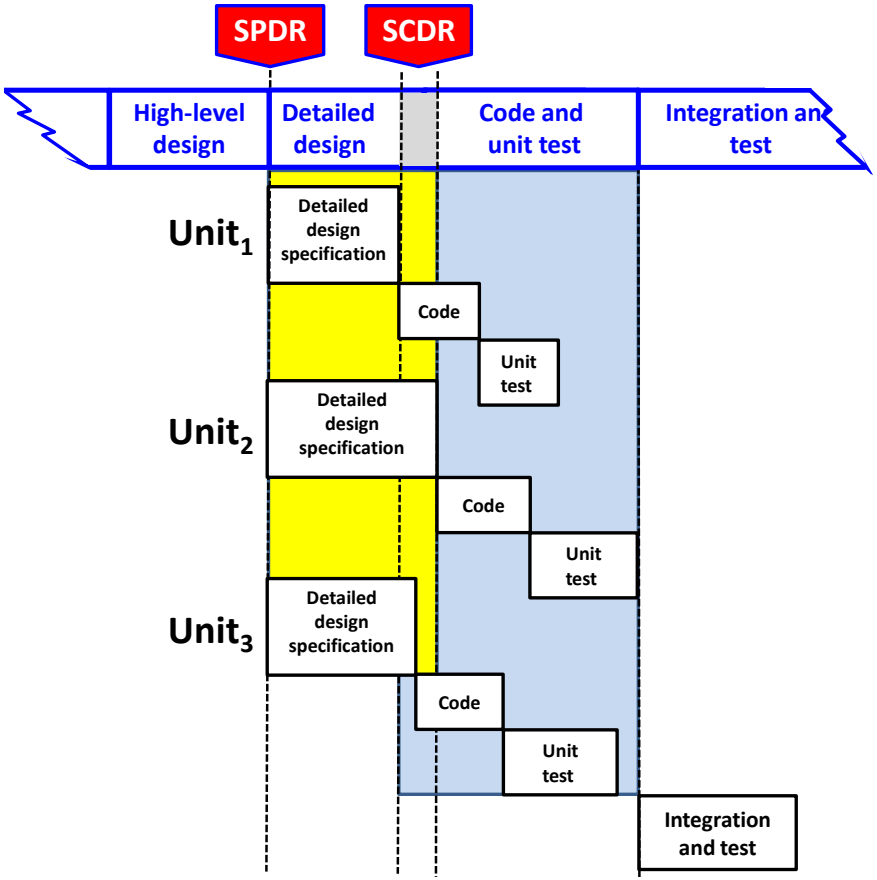


- Some of the terminology in the original model is outdated or confusing
  - Changes reflect that in this version of the model software is the product
  - New phase and review names now clearly show that they relate to software
  - These clarifications are helpful if we need to place the model in an acquisition or a software-intensive system development context

* Note that in modern terminology "high-level design" is replaced with "architectural design" and the work product is Software Architecture Description (SAD)

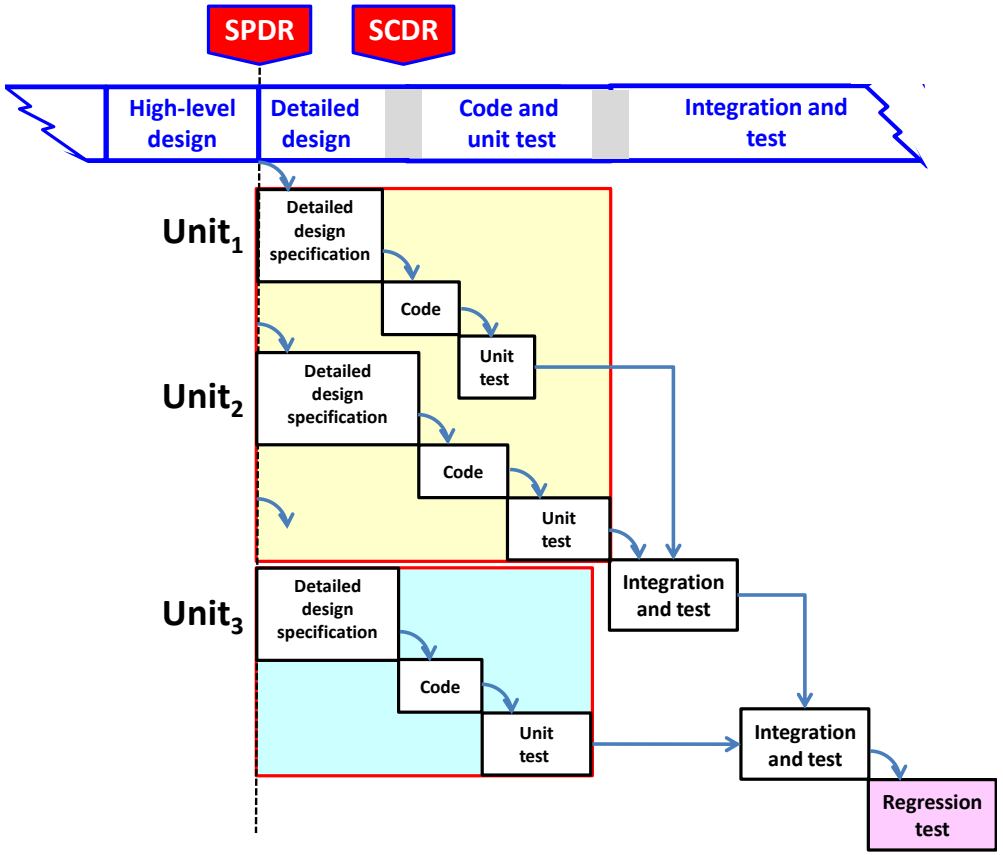
“Hidden” Concurrency in the Waterfall Model





- Unit-level work represents independent and concurrent processes
  - This is where the waterfall is a truly fitting metaphor
  - The streams don’t meet before hitting the pool at the bottom
- Note that SCDR positioning and content are ambiguous due to the fact that phase boundaries are now blurred

Incremental Integration and Pair-wise* Testing



- This strategy can reduce the problems stemming from belated problem discovery
  - In terms of the metaphor, note the intermediate pools in the picture
  - As a side-effect, phase boundaries are further blurred and the life cycle aspects of the model are less-and-less valid

* “Pairs” should not be taken literally; multiple units can be integrated as well

Scaling

- As it was discussed, the model seems to be a macro-model
  - However, the mentioned shortfalls are amplified if the process is indeed executed on the macro level
- Many of the problems can be dealt with if Incremental/Evolutionary strategies are used on the macro-level (including acquisition,) and Waterfall is applied only on lower levels
  - For example, the following strategy hierarchy can be implemented in a space system
    - Acquisition – Once-through (Equivalent of the Waterfall) or Evolutionary
    - System – Incremental Development involving **segments**
    - Segment – Incremental Development involving **elements**
    - Elements – Incremental Development involving **subsystems**
    - Subsystems – Incremental Development involving **Software Items**
    - Software Items* – Waterfall Development involving **Software Units**
- The requirements volatility problem stemming from requirements ambiguity can be mitigated via stronger requirements engineering processes such as prototyping

* Larger software Items could be also incremental where each increment is a waterfall

Some Whimsical Remarks on Requirements

- The Waterfall Model assumes that the requirements can be determined with high-fidelity before actual development starts
  - The mantra in real estate is “Location, location, location”
  - The key to successful software development is “Requirements, requirements, requirements”
- Unfortunately, requirements volatility is a fact of life
  - New, agile methodologies are designed to cope with the influx of new requirements
    - **“…we have come to value responding to change over following a plan”***
  - However, I still suggest listening to Yogi Berra:
    - **"If you don't know where you are going, you will wind up somewhere else"**

* Source: [Agile 2001]

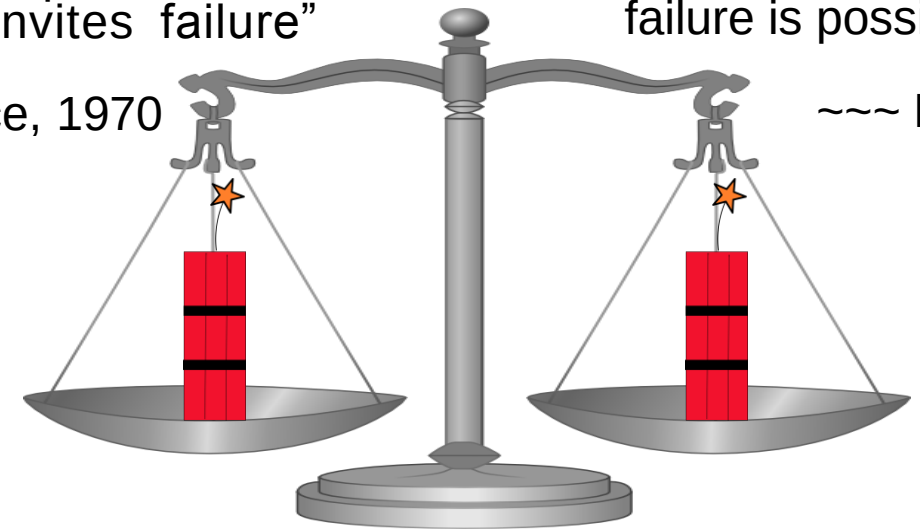
Conclusion - The Experts' Voice

"[Iterative development] projects are not easier to set up, to plan, or to control just because they are iterative. The project manager will actually have a more challenging task, especially during his or her first iterative project, and most certainly during the early iterations of that project, when risks are high and early failure is possible."

~~~ Philippe Kruchten, 2000

I believe in this [waterfall] concept, but the implementation … is risky and invites failure"

~~~ Winston Royce, 1970



Concurrency, scaling, and scope management are equally difficult problems in all methodologies

To Get Back to the Mood after the **Q/A**…

It winds along the face of a cliff
This path which I long to explore,
And over it dashes a waterfall,
And the air is full of the roar
And the thunderous voice of waters which sweep
In a silver torrent over some steep.
It clears the path with a mighty bound
And tumbles below and away,
And the trees and the bushes which grow in the rocks
Are wet with its jeweled spray;

From “A Coloured Print by Shokei,” by Amy Lowell

Acronyms

CDR	Critical Design Review
DBDD	Database Design Description
IDD	Interface Design Description
IRS	Interface Requirements Specification
PDR	Product Design Review
SCDR	Software Critical Design Review
SAD	Software Architecture Description
SDD	Software Design Description
SDP	Software Development Plan
SPDR	Software Preliminary Design Review
SRR	Software Requirements Review
SRS	Software Requirements Specification
SWRR	Software Requirements Review

References

Agile 2001	Agile Alliance, Manifesto for Agile Software Development, 2001
Boehm 1981	Boehm, B., Software Engineering Economics, Prentice-Hall, 1981
McConnell 1996	McConnell, S., Rapid Development, Microsoft Press, 1996
Royce 1970	Royce, W. W., Managing the Development of Large Software Systems, Proc. IEEE WESCON, August 1970
Stachowiak 1973	Stachowiak, H., Allgemeine Modelltheorie (General Model Theory), Springer, Wien, New York, 1973

Use of Trademarks, Service Marks and Trade Names

Use of any trademarks in this material is not intended in any way to infringe on the rights of the trademark holder. All trademarks, service marks, and trade names are the property of their respective owners.

Glade Creek Spring Waterfalls image is courtesy of **ForestWander Nature Photography**:
http://www.forestwander.com/wp-content/original/2010_03/glade-creek-spring-waterfalls.jpg
(accessed August 24, 2011)

Clip Art used on Slide 17 is courtesy of **Open Clip Art Library**:
http://all-free-download.com/free-vector/vector-clip-art/settlement_law_justice_clip_art_9
(accessed May 11, 2011)

Poem passage on Slide 18 is courtesy of **themargins.net:**
Lowell, A., 1912. *A Dome of Many-Coloured Glass*.
http://themargins.net/anth/1910-1919/lowellcolouredprint.html
(accessed September 9, 2011). Originally published by Boston: Houghton Mifflin, 1912. Reprint, New York: AMS, 1981.