L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

An Introduction

Burkhart Wolff
wolff@lri.fr

What is Software Engineering ?

- Methods, techniques and tools for
  - design: requirement analysis, models, specifications
  - development: programmation, integration
  - validation: prototypes, testing
  - verification: formal proof of required properties
  - maintenance: reusability, improvements

- A slightly longer answer:

What is Software Engineering ?

- ... slightly longer answer:

  The discipline of software engineering was created to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification. [Wikipedia [en]]

- Or much shorter:

  SE addresses the problems of

  « Development in the Large »

  ... so for teams with 100 or 1000 of developers, and budgets of sometimes billions of dollars.

What is Software Engineering ?

- … lets consider this more closely:

The discipline of software engineering was created to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification. [Wikipedia [en]]

What is Software Engineering ?

- ... lets consider this more closely:

The discipline of software engineering was created to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification. [Wikipedia [en]]

What is Software Engineering ?

- poor quality
- projects exceeding
  time and budget
- software is built systematically,
- ... within specification.

What is Software Engineering ?

	Covered in this course:
poor quality	✔
projects exceeding time and budget	-
software is built systematically,	✔
... within specification.	✔✔

What is Software Engineering ?

Covered in this course:

- poor quality ✔
- projects exceeding time and budget -
- software is built systematically, ✔
- ... within specification. ✔✔

for something like *constructive cost models* (COCOMO) you need experience with a concrete process !

Poor Quality: Some empirical data ...

- Size of Software ?
  - Peugeot 607 : 2 Mb embedded software
  - Windows 90: 10 Mb. LOC source, Win2000: 30 Mb.
  - Kernel Hyper V: 50000 LOC. (Highly complex, concurrent C)
  - Noyau RedHat 7.1 (2002) : ~2.4 M. LOC, XWindow ~1.8, Mozilla ~2.1 M.
  - Space Shuttle (and its environment) : ~50 MLOC

- Development Cost ?
  - Percentage of «Coding» ? 15 - 20 %

    Trend: Code is more and more generated (CASE Tools)
  - Proportion of Validation et Verification ? ~20% / ~20%

Poor Quality: Verification Costs

- costs ? 35 - 50 % of the global effort ?
- all “real” (large) software has remaining bugs …
- The cost of bug ?
  - the cost to reveal and fix it …
    or:
    the cost of a legal battle it may cause...
    or the potential damage to the image
    (difficult to evaluate, but veeeery real)
    or costs as a result to come later on the market
  - on the other side – you can't test infinitely, and verification is again 10 times more costly than thoroughly testing !

Why is it important to get software right?

? ? ?

Why is it important to get software right?

- Since information technology becomes more and more pervasive, the risks become more important
  - Reliability, Safety and Security becomes more critical :
    - transport systems (Cars, Métros, TGV), aviation controls, aerospace, ...
    - critical industrial processes, nuclear power plants, weapons, …
    - medical technologies: tele-surgery, radiation control…
    - critical telecommunication infrastructures and networks,
    - electronic commerce

Why is it important to get software right?

- Since information technology becomes more and more pervasive, the risks become more important
  - Reliability, Safety and Security becomes more critical :
    - transport systems (Cars, Métros, TGV), aviation controls, aerospace, ...
    - critical industrial processes, nuclear power plants, weapons, …
    - medical technologies: tele-surgery, radiation control…
    - critical telecommunication infrastructures and networks,
    - electronic commerce

For all of them exist certification processes, legal requirements, etcpp.

Why is it important to get software right?

- Since information technology becomes more and more pervasive, the risks become more important
  - Reliability, Safety and Security becomes more critical :
    - transport systems (Cars, Métros, TGV), aviation controls, aerospace, ...
    - critical industrial processes, nuclear power plants, weapons, …
    - medical technologies: tele-surgery, radiation control…
    - critical telecommunication infrastructures and networks,
    - electronic commerce

This should be the most important reason, but actually, it isn't.

Why is it important to get software right?

- The more likely reason is:

  it is so expensive if you don't !!! (It's the economy, stupid !)

  50 % of the overall costs were spent for test and verification in large software projects ... So, if the development of MS Vista cost 8 billion $ ...

- Another reason is:

  We want to build more complex systems, and validation and verification techniques are a limiting factor!

  Systems without Advanced SE can't simply be no longer built!

What are the Sub-disciplines of SE

- Well - again common knowledge [thanks wikipedia!]
  - *[1] Requirements engineering: The elicitation, analysis, specification, and validation of requirements for software.*
  - *[2] Software design: The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.*
  - *[3] Software construction: The detailed creation of working, meaningful software through a combination of coding, verification, testing and debugging.*
  - ...

What are the Sub-disciplines of SE

- Well - again common knowledge [thanks wikipedia!]
  - . . .
  - *[4] Software verification: The verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.*

    [I add here: This may include test-generation, formal proof and model-checking activities ...]
  - *[5] Software maintenance: The totality of activities required to provide cost-effective support to software.*
  - ...

What are the Sub-disciplines of SE

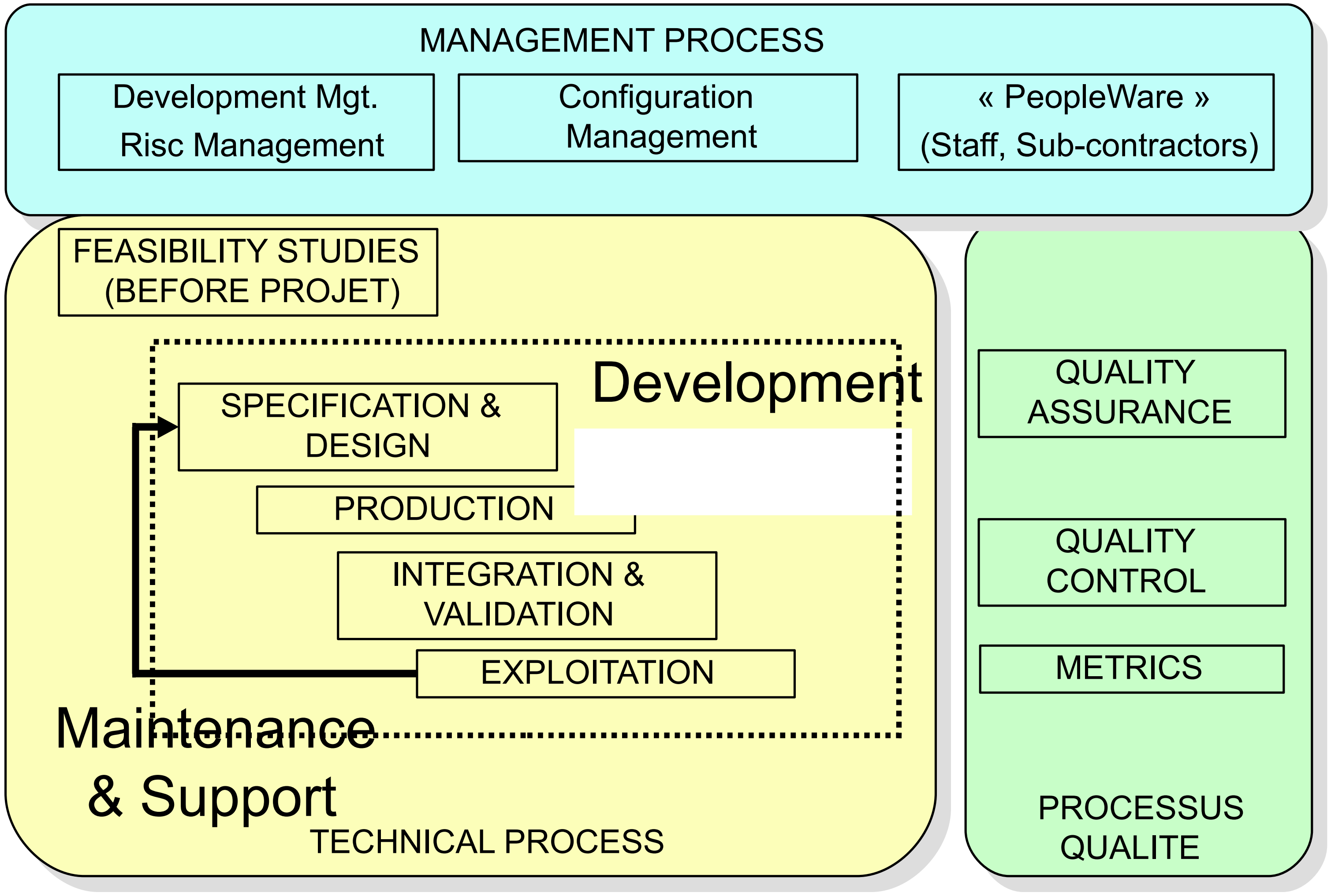
- Well - again common knowledge [thanks wikipedia!]
  - . . .
  - *[6] Software configuration management: The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle.*
  - *[7] Software engineering management: The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.*

    [Again: this is not what we do in this course: it requires more experience and a concrete process to do this ...]
  - . . .

What are the Sub-disciplines of SE

- Well - again common knowledge [thanks wikipedia!]
  - . . .
  - *[8] Software engineering process: The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.*
  - *[9] Software engineering tools and methods: The computer-based tools that are intended to assist the software life cycle processes (see Computer-aided software engineering) and the methods which impose structure on the software engineering activity with the goal of making the activity systematic and ultimately more likely to be successful.*
  - *[10] Software quality management: The degree to which a set of inherent characteristics fulfills requirements.*

One way to view Software Engineering Project



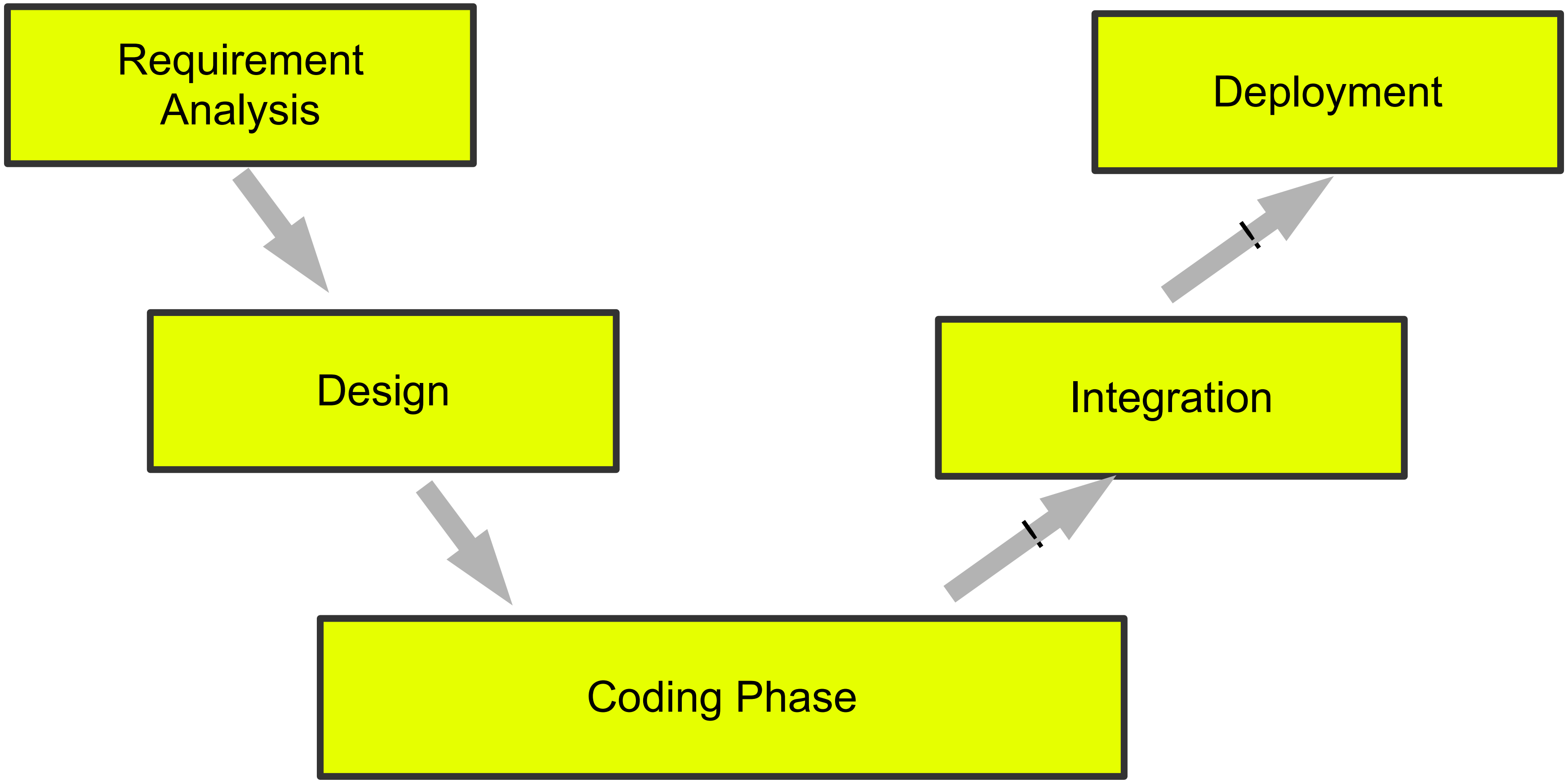
How can software be «built systematically»?

- Organize a development into formally described process !
  - ... with identified phases, (which correspond partly to the aforementioned "SE disciplines")
  - ... staff (and organization and cost-plans)
  - ... defined deliverables (i.e. documents, codes, ...)
  - ... procedures (and tools !) to validate the quality of the deliverables(reviews, static checks)
  - ... procedures to version and configurate deliverables (in particular code)
  - Compare: THE SWEBOOK
    IEEE Computer Society an international standard ISO/IEC TR 19759:2005

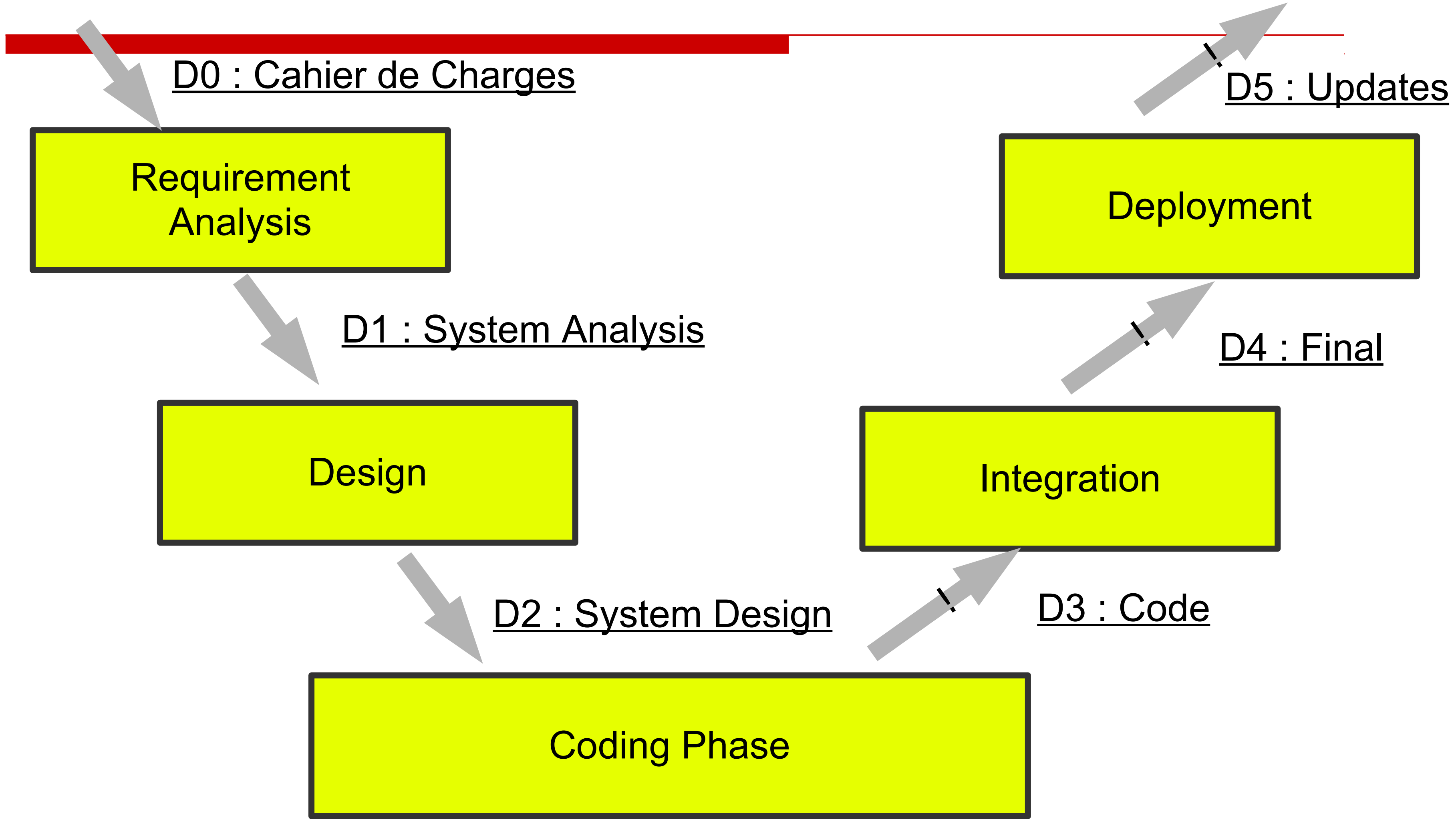
How can software be «built systematically»?

- An Example Process-Model: The V-Model.
  - V-Modell XT [German Administration 2005]
    http://de.wikipedia.org/wiki/V-Modell_(Entwicklungsstandard)#cite_note-6
  - ... phases : Requirement, Architectural Design, Design, Code, Tests, passed deployments, ...
  - ... defined deliverables (i.e. documents, codes, ...) templates that have to be "taylored"
  - ... procedures: (and tools !) : an xml editor, a version management and an access control management for the 35 (!) different roles in the process

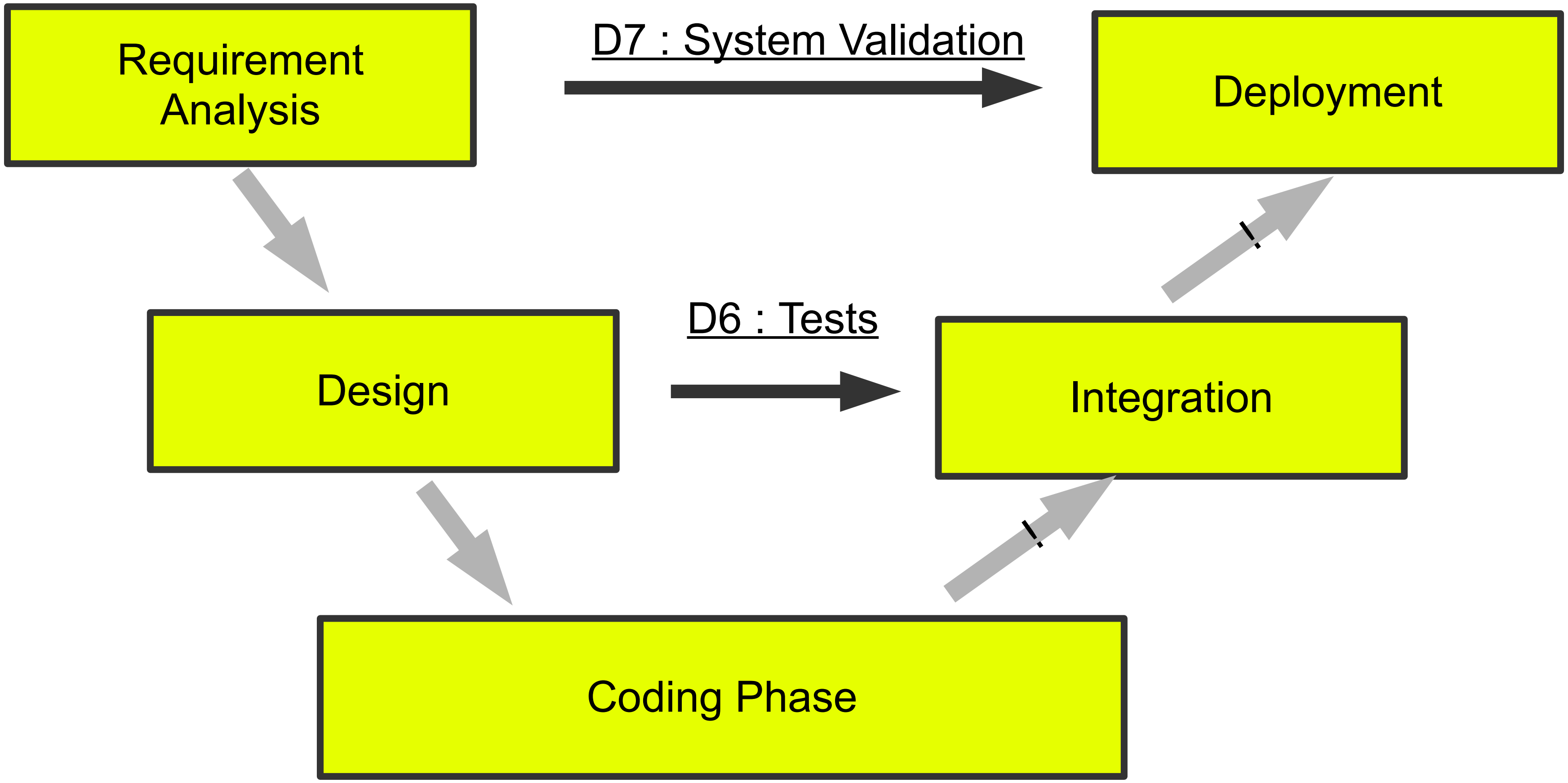
How can software be «built systematically»?



How can software be «built systematically»?

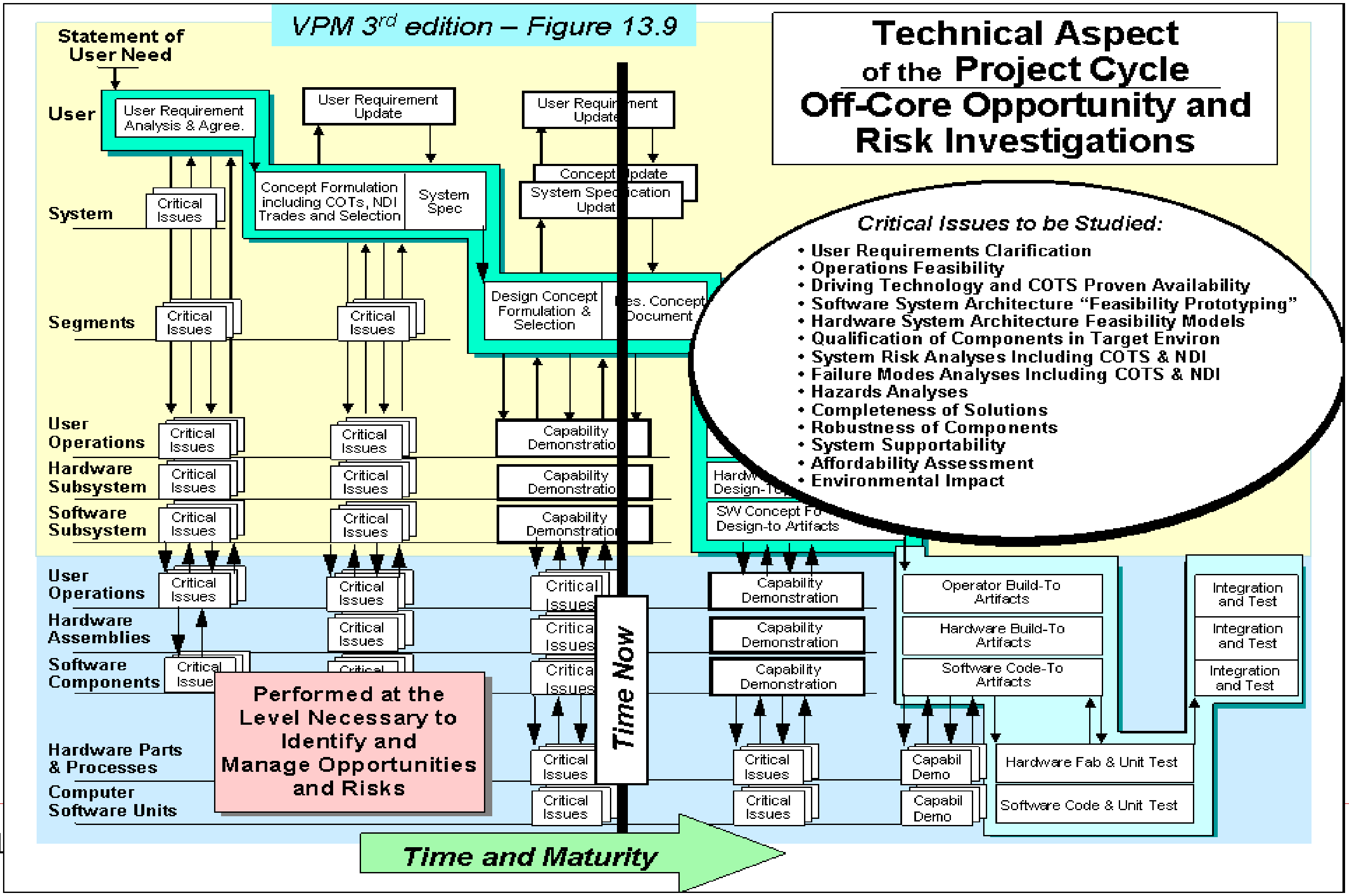


How can software be «built systematically»?



How can software be «built systematically»?

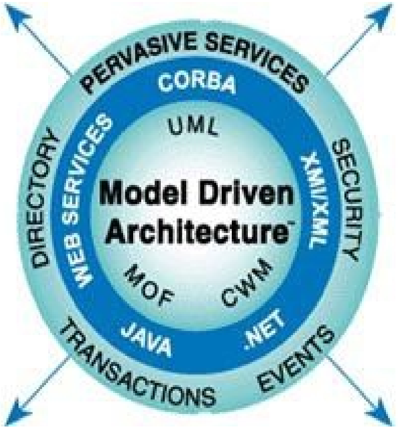
- A Variant: The VPM3-Model (Daimler)


VPM 3rd edition – Figure 13.9
Technical Aspect of the Project Cycle
Off-Core Opportunity and Risk Investigations
Statement of User Need
User
System
Segments
User Operations
Hardware Subsystem
Software Subsystem
User Operations
Hardware Assemblies
Software Components
Hardware Parts & Processes
Computer Software Units
User Requirement Analysis & Agree.
User Requirement Update
User Requirement Update
Concept Formulation including COTs, NDI Trades and Selection
System Spec
Concept Update
System Specification Update
Design Concept Formulation & Selection
Des. Concept Document
Critical Issues
Capability Demonstration
Hardware Design-To
SW Concept For Design-to Artifacts
Operator Build-To Artifacts
Hardware Build-To Artifacts
Software Code-To Artifacts
Integration and Test
Capabil Demo
Hardware Fab & Unit Test
Software Code & Unit Test
Performed at the Level Necessary to Identify and Manage Opportunities and Risks
Time Now
Time and Maturity

Critical Issues to be Studied:
- User Requirements Clarification
- Operations Feasibility
- Driving Technology and COTS Proven Availability
- Software System Architecture "Feasibility Prototyping"
- Hardware System Architecture Feasibility Models
- Qualification of Components in Target Environ
- System Risk Analyses Including COTS & NDI
- Failure Modes Analyses Including COTS & NDI
- Hazards Analyses
- Completeness of Solutions
- Robustness of Components
- System Supportability
- Affordability Assessment
- Environmental Impact

How can software be «built systematically»?

- Another Variant /Alternative
  IBM Rational Unified Process
  (RUP)



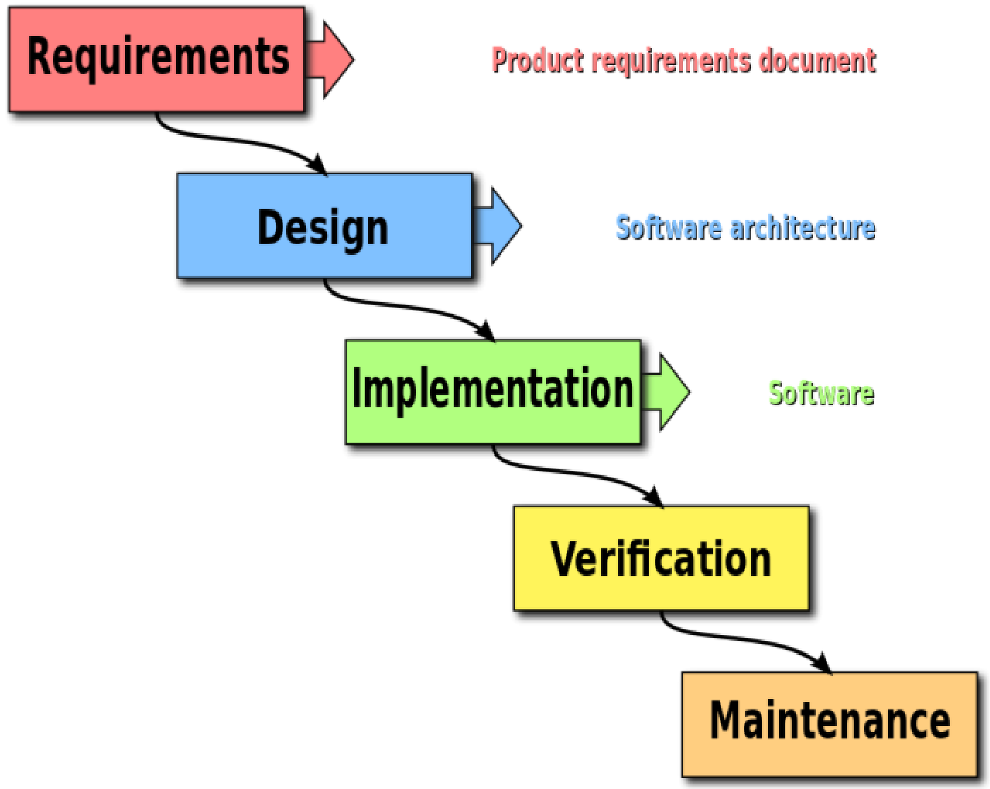
- Idea : Using UML and OCL integrated into the Deliverables (documents)
- Idea : Allows for semi-formal editing, more precise notation and therefore better communition
- Analysis, Design and Code Documents CONTAIN standardized diagrammatic specification elements (the "model") which can be automatically validated
- Code and Tests can partially be generated from design models (Model-Driven Engineering (MDA))

Other Variants of Dev. Process Models

Waterfall Model

[Benington 56, Royce 70]
Royce presented this model as an example of a flawed, non-working models.

Category: Bad example.

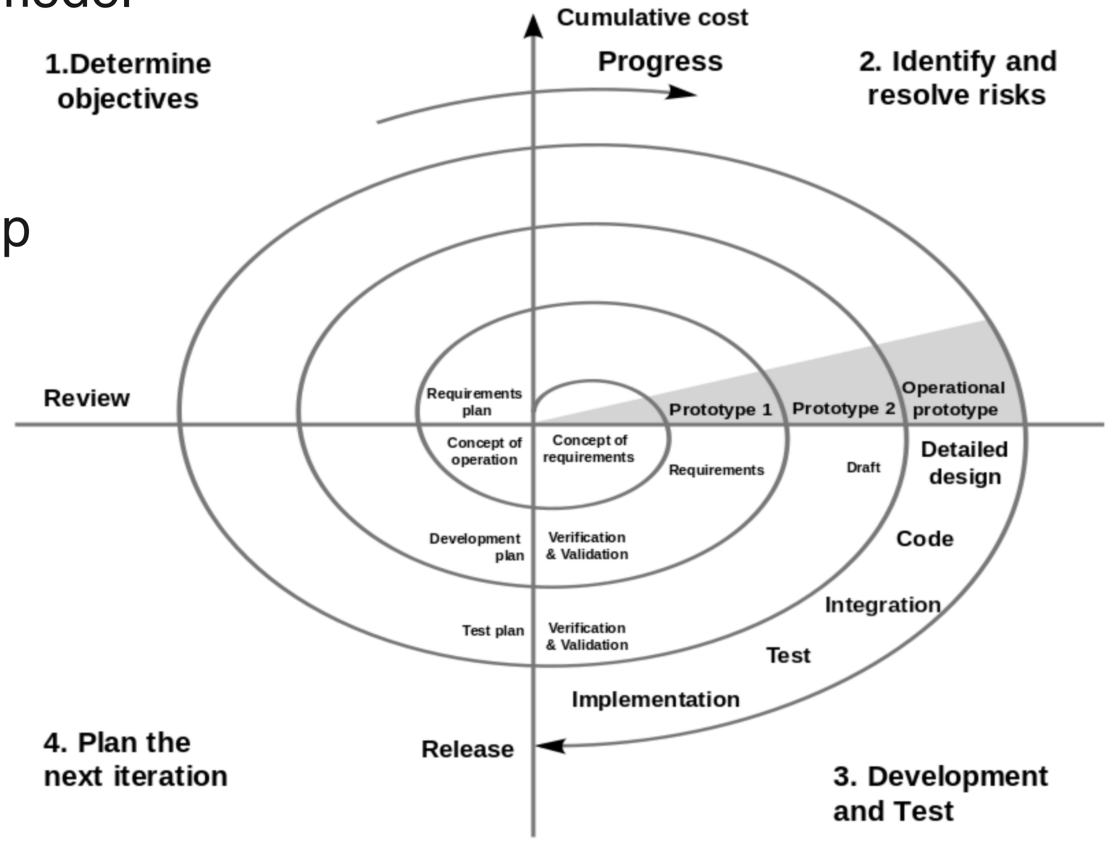


Other Variants of Dev. Process Models

- Spiral Model [Barry Boehm 88]

combines some key aspect of the waterfall model and rapid prototyping methodologies, in an effort to combine advantages of top-down and bottom-up concepts.

Today mostly a conceptual reference; ideas are retaken in « agile development »

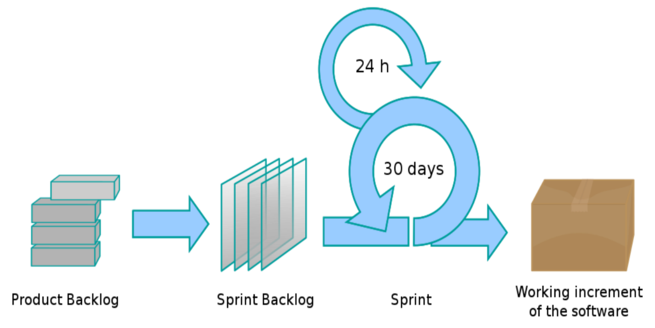


Other Variants of Dev. Process Models

- Agile Software Development
  [Beck et al 2001, V2: 2010]
  Advocates:
  - adaptive planning,
  - evolutionary, incremental development,
  - early delivery,
  - continuous improvement,
  - and it encourages rapid and flexible response to change

  Particular variants are called « Extreme Programming » (with an emphasis on early, handwritten tests)

  SCRUM (with emphasis on social organization and cont. team-reviews



Why UML in a SE Course ?

- It is clearly not perfect
  - very syntactic, very diagrammatic flavor
  - diagrams do not necessarily scale up
  - not everyone in industry uses it (large companies typically have there own development process, reflecting their own « culture »)

- BUT: Many in industry use it,
  - or use similar things (SysML), and most practitioners in industry would understand UML
  - we use it in requirements analysis, design, and for test generation techniques.