CALIFORNIA STATE UNIVERSITY
FULLERTON™

COLLEGE OF ENGINEERING
AND COMPUTER SCIENCE

Advanced Software Process

Part III: The Defined Process

13. Defining the Software Process

Dave Garcia-Gomez
Faculty / Lecturer
Department of Computer Science

Course Roadmap

Part I: Software Process Maturity
- 1. A Software Maturity Framework
- 2. The Principles of Software Process Change
- 3. Software Process Assessment
- 4. The Initial Process

Part II: The Repeatable Process
- 5. Managing Software Organizations
- 6. The Project Plan
- 7. Software Configuration Management (Part I)
- 8. Software Quality Assurance

Part III: Defined Process
- 9. Software Standards
- 10. Software Inspections
- 11. Software Testing
- 12. Software Configuration Management (continued)
- 13. Defining the Software Process
- 14. The Software Engineering Process Group

Part IV: The Managed Process
- 15. Data Gathering and Analysis
- 16. Managing Software Quality

Part V: The Optimizing Process
- 17. Defect Prevention
- 18. Automating the Software Process
- *19. Contracting for Software*
- *20. Conclusion*

Defining the Software Process

- Process Standards
- Definitions
- Levels of Software Process Models
- Prescriptive and Descriptive Uses of Models
- A Software Process Architecture
- Critical Software Process Issues
- A preliminary Process Architecture
- Larger Process Models
- Detailed Process Models
- Entity Process Models
- Process Model Views
- Establishing and Using a Process Definition
- Basic Process Guidelines

Defining the Software Process

- Software development can be exceedingly complex and there are often many alternative ways to perform the various tasks.
- A defined process can help guide the SW professionals through these choices in an orderly way.
- With an established process definition they can better understand
  - what they should do,
  - what they can expect from their co-workers, and
  - what they are expected to provide in return.

Defining the Software Process

- Factors to be considered:
  - Since software projects have differences, their SE processes must have difference as well.
  - In the absence of a universal SE process, organizations and projects must define processes that meet their own unique needs.
  - The process used for a given project must consider the experience level of the members, current product status and the available tools and facilities.

Process Standards

- While the need for project-unique process definitions is clear, there are also compelling reasons for standardization:
  - Process standardization helps to reduce the problems of training, review, and tool support.
  - With standard methods, each project's experiences can contribute to overall process improvement.
  - Process standards provide the basis for process and quality measurements.
  - Since process definitions take time and effort to produce, it is impractical to produce new ones for each project.

Process Standards

- The conflicting needs for customization and standardization can often be resolved by establishing a process architecture, which consists of a standard set of unit or kernel process steps with rules for describing and relating them.
- Customization is then achieved through appropriate interconnections of these standard elements into tailored process models.

Definitions

- Software engineering
  - The disciplined application of engineering, scientific, and mathematical principles, methods, and tools to the economical production of quality software
- Software process
  - The set of activities, methods, and practices that are used in the production and evolution of software
- Software engineering process
  - The total set of SE activities needed to transform a user's requirements into software

Definitions

- Software process architecture
  - A framework within which project-specific software processes are defined
- Software process model
  - One specific embodiment of a software process architecture

Levels of Software Process Models

- Software process models can be defined at any of three levels.
  - The U, or Universal, process model provides a high-level overview.
  - The W, or Worldly, process model is the working level that is familiar to most programmers and managers.
  - The A, or Atomic, process model provides more detailed refinements.

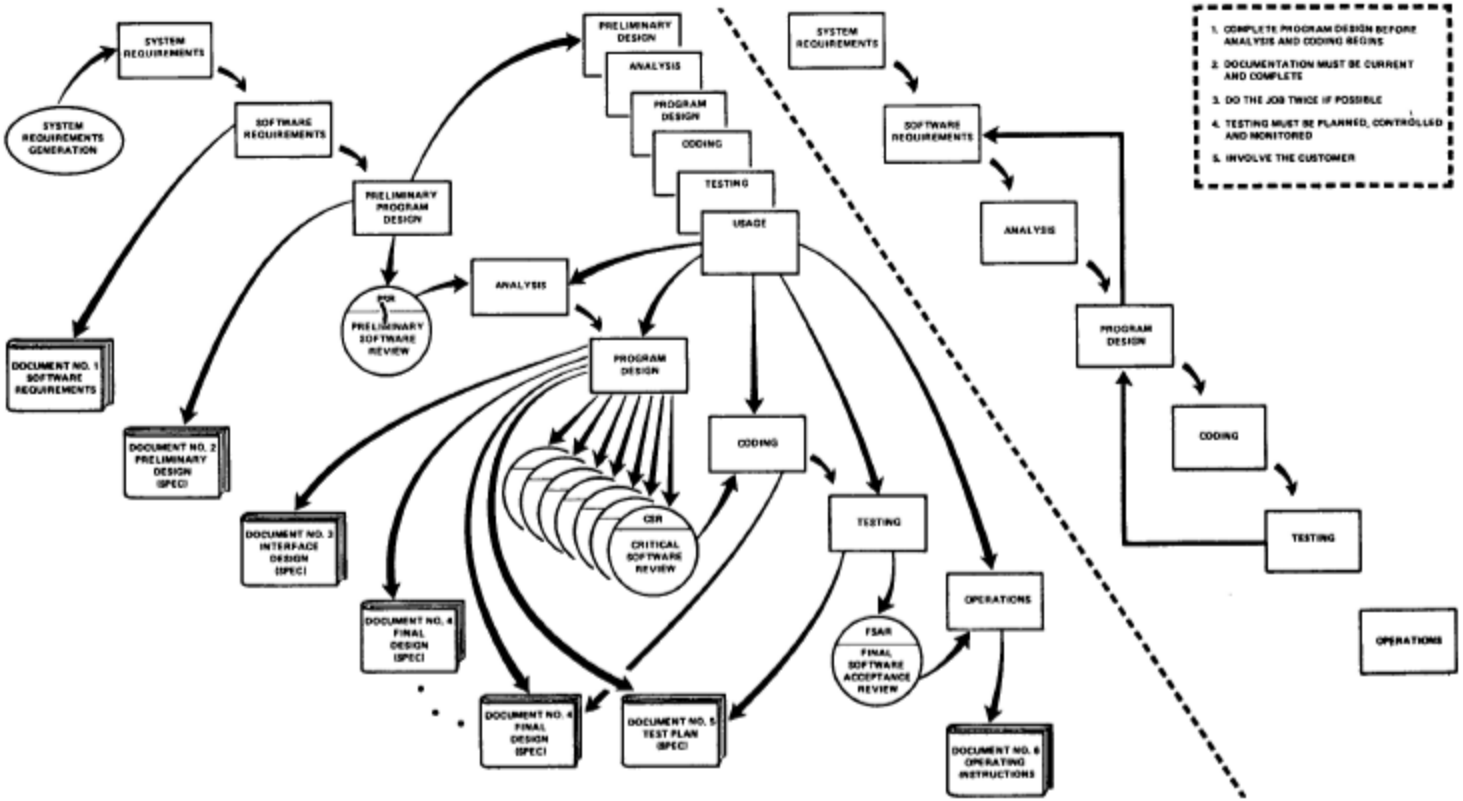
Levels of Software Process Models

- The Waterfall Model [Royce 1970] is still the best known and most widely used overview framework for the software development process.

Levels of Software Process Models

- The Waterfall Process Model [Figure 13.1]
  - System requirements
  - Software requirements
  - Analysis
  - Program design
  - Coding
  - Testing

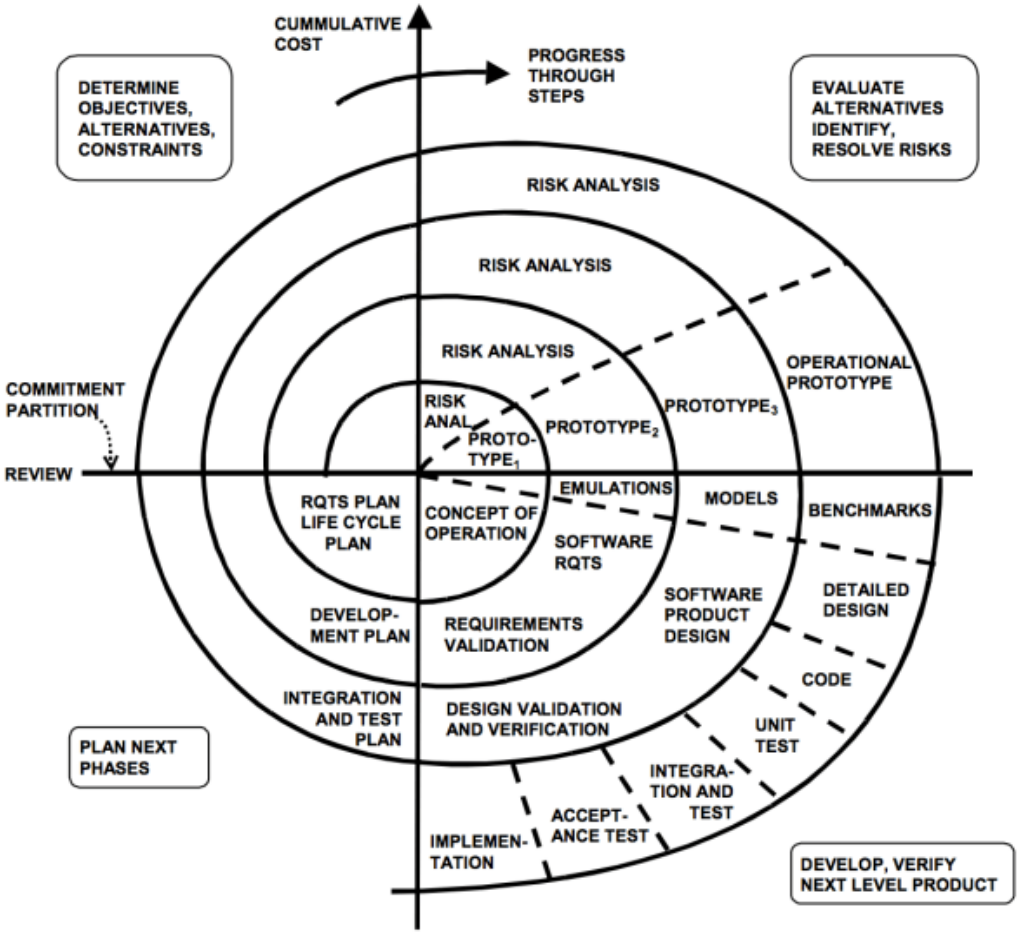
Levels of Software Process Models



Levels of Software Process Models

- Spiral Model [Boehm 1988]
- Spiral Model of the Software Process [Figure 13.2]

Levels of Software Process Models



U Process Models

- Unfortunately, the real world of software development does not conform neatly to either of these models.
- While they represent the general work flow and provide overview understanding, they are not easily decomposed into the progressively finer levels of detail that are needed to guide the work of the software professionals.

U Process Models

- Views
  - Task-oriented view
  - Functional view
  - Behavioral view
  - Structural view
  - Conceptual view
  - Entity view

A Process Models

- At the opposite extreme from U-level models, Atomic- (A-) level process models can be enormously detailed.
- They are needed by anyone who attempts to automate a specific process activity or use a standardized method or procedure to guide execution of a task.

A Process Models

- Precise data definition, algorithmic specification, information flows, and user procedures are essential at this level.
- The amount of detail to be included in such models must be determined by their use.
- Atomic process definitions are often embodied in process standards and conventions.

W Process Models

- The Worldly (W) process level is of most direct interest to practicing SE.
  - It guides the sequence of their working tasks and defines task prerequisites and results.
  - When reduced to operational form, these models generally look like procedure.
  - They specify who does what when.
  - Where appropriate, they reference the A level that specifies standard task definitions or tool usage.
  - For each task, W models define the anticipated results, the appropriate measures, and the key checkpoints.

Examples of Process Models

- The three levels of process models can be viewed as embodied in
  - Policies at the U level
  - Procedures at the W level
  - Standards or tools at the A level

Examples of Process Models

- At the U level, policies establish a high-level framework and set of principles that guide the overall behavior of organizations.
- They are particularly helpful in unanticipated circumstances where no precedents have been established.
- Examples
  - All work will be subjected to an inspection before it is incorporated in a baseline.

Examples of Process Models

- At the W level, procedures are established to implement the policies.
- This W-level process model refers to any available Atomic-level standards that define precisely how tasks are to be performed.
- Examples
  - A procedure might define the points at which QA reviews are to be conducted and how the resulting issues are to be handled.

Examples of Process Models

- Atomic level standards then serve as the basis for directing the work and for the SQA review.
  - For example, a code inspection standard would specify what code is to be reviewed, when, the methods to be used, the reports to be produced, and the acceptable performance limits.

Prescriptive and Descriptive Uses of Models

- Process models can be used either to describe what is done or to characterize what is supposed to be done.
  - In a descriptive case, models can provide useful information about the process and its behavior.
  - This book uses process models in a prescriptive sense.
    - The approach is to define how the process should be conducted and suggest where appropriate policies, procedures, and standards could help guide the work.

A Software Process Architecture

- Since most organizations have at least some policies, procedures, and standards, they are also generally following some intuitive U-, W-, and A-level models both prescriptively and descriptively.

A Software Process Architecture

- To be fully effective, the process models should be explicit and should relate to each other.
- The problem of building process models, in fact, is much like that of building software systems.
- An architectural framework is needed to define the basic elements, how they relate, and how they are decomposed into greater detail.

Critical Software Process Issues

- The reason for defining the software process is to improve the way the work is done.
- By thinking about the process in an orderly way, it is possible to anticipate problems and to devise ways to either prevent or to resolve them.
- Some of the major software process issues concern
  - Quality
  - Product technology
  - Requirements instability
  - Complexity

Critical Software Process Issues

- Requirements instability
  - Unknown requirements
  - Unstable requirements
  - Misunderstood requirements

A Preliminary Process Architecture

- Organizations that face the issues of quality, product technology, requirements stability, and/or complexity need to define ways to address them.
- A process architecture permits them to represent and manipulate the process at the U level and then selectively to refine it to the W and A levels.
- This, or course, needs an overall architectural framework and a set of definitions.

A Preliminary Process Architecture

- Basic Unit Cell [Figure 13.3]
  - The basic elements of the process architecture – unit cells
  - Each cell is defined to accomplish a specified task and is uniquely identified.
  - Each cell also has required entry conditions specified for task initiation that include the inputs (one or more with their sources).
  - The task standards, procedures, methods, responsibilities, and required measurements are also defined.
  - The exit conditions define the results produced, their level of validation, and any other post-task conditions.
  - Cell feedback refers to any data provided to or received from other stages in the process.

A Preliminary Process Architecture

- The Single-Cell Development Process [Figure 13.4]
  - Shows a full development cycle in one cell
  - Shows what kind of information is required for every process cell

A Preliminary Process Architecture

- The U-Level Development Process [Figure 13.5]
  - More refined U-level model of the development process
  - The development cycle is broken into the basic cells of the Waterfall model.
- U-Level Development Process Cell Specifications [Table 13.1]

Standard Process Elements

- When looked at from the highest, or U, level, software processes tend to look much the same.
- This is because they are described in broad generic terms like design, implementation, or test.
- When these activities are broken into more detail, however, significant differences show up.

Standard Process Elements

- With all variation at the W-level, however, many software activities can be relatively standardized across different projects.
- It is thus possible to establish some basic process cells that can be interconnected in different ways to meet project-unique needs.
- The detailed structure of these standard cells are then further defined by A-level models as needed.

Implementation Cells

- In defining a standard set of software process cells, we start with some relatively detailed software tasks.
  - Basic Implementation Cell C_0 [Figure 13.6]
  - Quick Kernel K_q [Figure 13.7]
  - Inspection Operator [Figure 13.8]

The Unit Implementation Kernel

- Implementation Cell C_1 [Figure 13.9]
- Unit Implementation Kernel K_1 [Figure 13.10]

The Cell Specification

- Once the general flow of a process is known, it is important to define each process cell.
- ETVX paradigm
  - Entry
  - Task
  - Verification
  - Exit
- ETX specification
  - A modified version
  - The explicit characterization of the Entry, Task, and Exit criteria for each process action

The Cell Specification

- ETX Specification for the Implementation Cell C_1 [Table 13.2]
- ETX Specifications for the Unit Implementation Kernel K_1 [Table 13.3]

The Cell Specification

- These process specifications for each project should include explicit responsibilities for task performance and should refer to the applicable standards and procedures.
- A defined process can provide software engineers guidance as well as setting the standards for management review and SQA audit.

Larger Process Models

- Once some basic process cells have been defined, it is possible to construct larger process models.
- This is done by interconnecting these basic cells in various ways.
- The idea is consciously to design the development or maintenance process to address the anticipated issues and problems.
  - Quality, product technology, unknown requirements, unstable requirements, misunderstood requirements, complexity

Product Technology Unknowns

- In advanced software systems there are often significant technical unknowns.
- Hardware engineers have long known that initial implementations of novel products rarely work as expected and are never directly shippable.
  - They thus build breadboards to test their technical concepts and experiment with alternative approaches.
- With software it is also appropriate to breadboard critical, complex, or unusually demanding functions.

Product Technology Unknowns

- Experimental Kernel K_e [Figure 13.11]
- ETX Specifications for the Unit Implementation Kernel K_e [Table 13.4]

The Problem of Complexity

- The process kernels described thus far have focused on small tasks that could be performed by one or two programmers.
- While these kernels provide useful guidance for small tasks, their real value is the insight they can provide for larger projects.
- Here multiple modules are typically involved, and the work of many professionals must be coordinated.

The Problem of Complexity

- Integration Cycle [Figure 13.12]
- Integration Cell [Figure 13.13]
- Build Cell [Figure 13.14]

Requirements Instability

- To this point, it has been assumed that the software requirements were known, stable, and understood.
- Since this is rarely the case, compensating process provisions are needed.
- The appropriate provisions depend on which of the three basic types of requirements instability is involved.

Requirements Instability

- KSU'
  - The requirements are known and stable, but the implementers do not understand them sufficiently well to produce an adequate implementation (not U, or U').
- KS'U
  - The requirements are known and understood, but they are not stable (S').
- K'SU
  - The requirements are stable and understood, but they are not fully known (K').

Prototype Process Models

- Experimental Kernel K3 (KSU') Requirements Not Understood [Figure 13.11]
- Requirements Unstable (KS'U) [Figure 13.15]
- Requirements Unknown (K'SU) [Figure 13.16]

Prototype Process Models

- Prototype programs can be built to learn the potential customers' reactions.
- These prototypes demonstrate one or more facets of system behavior for test with the intended users.
- They can then be used to try to reduce the requirements uncertainties

The Use of Prototyping

- Each of the prototyping methods could be used at any of the process levels.
- In a large system some modules may require some form of prototyping, while others could be developed directly.
- With external interface, for example, the involved modules should often be prototyped and tested with the end users.

Prototyping Issues

- Evidence shows that product development through prototyping is substantially faster and less expensive than traditional methods.
  - Even if it is only intended as a quick experiment, the prototype's objectives should be clearly established before starting to build it.
  - Define the prototype process in advance.

Detailed Process Models

- All the process models in the world will not help to produce programs unless they can be reduced to the level of programming.
- This is the Atomic process level.
- With the A-level process, the detail used in task definition should be appropriate to the knowledge and skill of the professionals.

Detailed Process Models

- One example is the partial decomposition of the process for building regression test buckets.
- Regression Test Refinement [Figure 13.17]
  - The regression test process starts with the regression test plan.
- Regression Test Planning [Figure 13.18]
  - Regression test planning is further decomposed.

Entity Process Models

- The process models we have discussed so far are similar to state models of a software system.
- In simplest terms, the process is either in an idle state before the entry criteria for the first cell are met, or the process is in a state represented by one of the succeeding cells.
- After the final output from the last cell is produced, the model returns to the idle state, waiting for further requirements.

Entity Process Models

- As the implementation tasks become more complex, however, this simplistic picture may provide a less realistic representation of actual task behavior.
- One alternative is to consider basing process models on entities.
  - Here one deals with entities and the actions performed on them.
  - Each entity is a real object that exists and has an extended lifetime.

Entity Process Models

- Examples of entities
  - The deliverable code
  - The users' installation and operation manuals
  - The requirements documents
  - The design
  - The test cases and procedures

Entity Process Models

- Entity process models (EPMs) provide a useful additional representation of the software process because they are often more accurate than task-based models for complex and dynamic processes.
- It is clear that each entity cycles through a set of states during the software process.

Entity Process Models

- By focusing on these states and the actions required to cause state transitions, the process of producing an entity process model becomes relatively straightforward:
  - Identify the process entities and their states.
  - Define the triggers that cause the transitions between these states.
  - Complete the process model without resource constraints (unconstrained process model, UPM).
  - Impose the appropriate limitations to produce a final constrained process model (CPM).

Entity Process Models

- Entity Process Model [Humphrey and Kellner 1989]:
  - Watts S. Humphrey and Marc I. Kellner, Software Process Modeling: Principles of Entity Process Models, Technical Report, CMU/SEI-89-TR-002, Feb. 1989.
  - https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=10859

Process Model Views

- The three views of process models are the state view, the organizational view, and the control view.
- The state view is what we have discussed so far, with the states either representing various stages of the process (tasks) or stages of the product (entities).
  - Task-oriented
  - Entity-oriented

Process Model Views

- Organizational View
  - A view of the development process that defines the responsibilities for each activity
  - Organization View [Figure 13.19]
- Control view
  - Relates to measurement and control
  - Control and Measurement View [Figure 13.20]

Process Model Views

- The three views of process models are different views rather than alternatives.
- They each present an essential perspective of the process that must be understood, defined, and managed.
- If any view is not addressed, an important facet of software management will likely be overlooked.

Establishing and Using a Process Definition

- Each software organization should establish a process architecture and process models tailored to its particular needs.

Establishing and Using a Process Definition

- This tailoring is done as follows:
  - Define a standard process as a foundation for tailoring.
  - Establish the ETX specifications for the standard process model.
  - Make provisions to gather and track the resulting process measurements.
  - Establish checkpoints and standards for SQA review.
  - Incorporate specific measurement and reporting provisions.
  - Instruct the development personnel on the use and value of the process architecture, the standard process models, and when, why, and how they should be tailored.

Establishing and Using a Process Definition

- Since many projects will likely need their own unique process definitions, they should start with the standard process and then take the following steps:
  - Identify the unique project issues, problems, and success criteria.
  - Document the adjustments required to the standard process to produce a basic overall project process.
  - For each software system component, repeat these definitions.
  - Once each program module has been identified, consider the process definition for it as well and make any necessary adjustment.

Basic Process Guidelines

- Finally, some guidelines are needed for developing and using a process architecture and its process models:
  - Establish objectives for each project's process
  - Define the basic process architecture, make sure it meets the needs of the projects, and then enforce it as an overall process framework
  - Remember that each project, component, and module is unique and its process should be uniquely determined.
  - Establish process definition standards.
  - Change the process model dynamically as the problem change.
  - Require that all deviations for the standard process be documented, reviewed, and approved.

References

Humphrey, Watts S., *Managing the Software Process*, The SEI Series in Software Engineering, Addison-Wesley, 1989. (29th Printing, May 2003) (ISBN 0-201-18095-2)