F - 3: Process Architecture

Software development and management processes should function coherently together to with well understood and simple boundaries; taking their cue from good software design. A software process architecture aids this by providing a context and scope for the processes.

What it is for:	To provide a stable framework and context for changing software development processes
When to introduce	Before undertaking widespread process definition work
When to use	When reviewing the business case for software processes, or exploring or revising software processes
When not to use	N/A

A good process architecture will reflect the business needs of the software development organization, provides a degree of stability for software processes, and an understandable structure that shows the relationships between the working practices.

It acts as a navigation guide and high level view of working practices which can be very useful for staff induction and training purposes. By providing a shared mental model of the working practices it helps technical staff work more effectively together. It allows process engineers to quickly revise, add or remove elements as working practices develop. It also provides the means for process engineers to analyse and assess software processes, and their relationships, identifying any omissions or inefficiencies, and opportunities for innovation.

A software process architecture is analogous to a software architecture, when in place, but its development is usually different. Software architectures are usually produced as part of the design activity before the code is produced. It is unusual for a software process architecture to be developed prior to the use of working practices. Typically the order of development is almost the exact reverse; working practices of some sort will generally be in place - managers and staff will be doing useful work prior to any process improvement or definition work[1].

[1] (For software the development process is typically *design, code, implement*; for working practices it is usually *practice in place, practice defined, documented and consolidated, practices organized*. An advantage of this reverse order of development is that the flexibility and necessary fuzziness and flexibility can be recognized, understood and built into the architecture from the beginning.)

Sophisticated, process aware organizations may design a process architecture to be populated by yet to be acquired or designed processes, but care should be taken to ensure it actually reflects business realities, and has the properties required to support effective processes and working practices, rather than just symbolizing a process or lifecycle model. In addition care is required to ensure that the architecture is acceptable to those affected by it. Top down architectural development and process definition can be perceived as 'overwriting' and discrediting existing, familiar and well liked working practices, and, by implication, professional competence. This is very undesirable. It is better to capture existing practice, so far as it can be described in an architecture, and then develop this collaboratively and progressively.

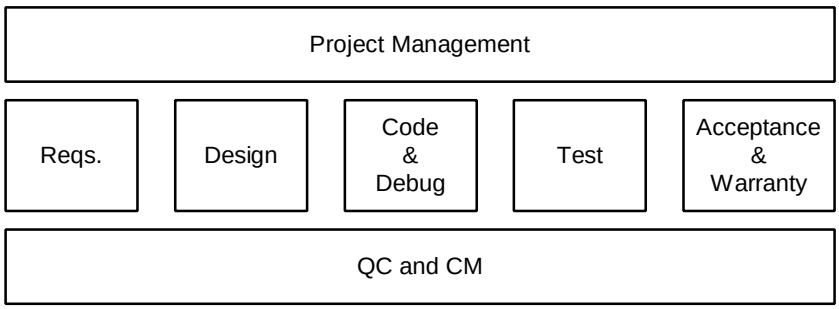
There are three stages to the evolution of software process architectures. Or, more accurately there are two precursors to the true process architecture. The three stages of architecture development are:

- Preliminary (Symbolic) Architecture;
- Intermediate (Lifecycle) Architecture;
- True Software Process Architecture.

These are described below:

Preliminary (Symbolic) Architecture:

The preliminary software process architecture is the symbolic representation. This is usually a diagram with a number of named nodes, often arranges in a pattern that implies the relationships between the nodes. The nodes themselves represent areas of software development working practice, and often management practice too. An example of this preliminary architecture is the 'sandwich model' shown here.



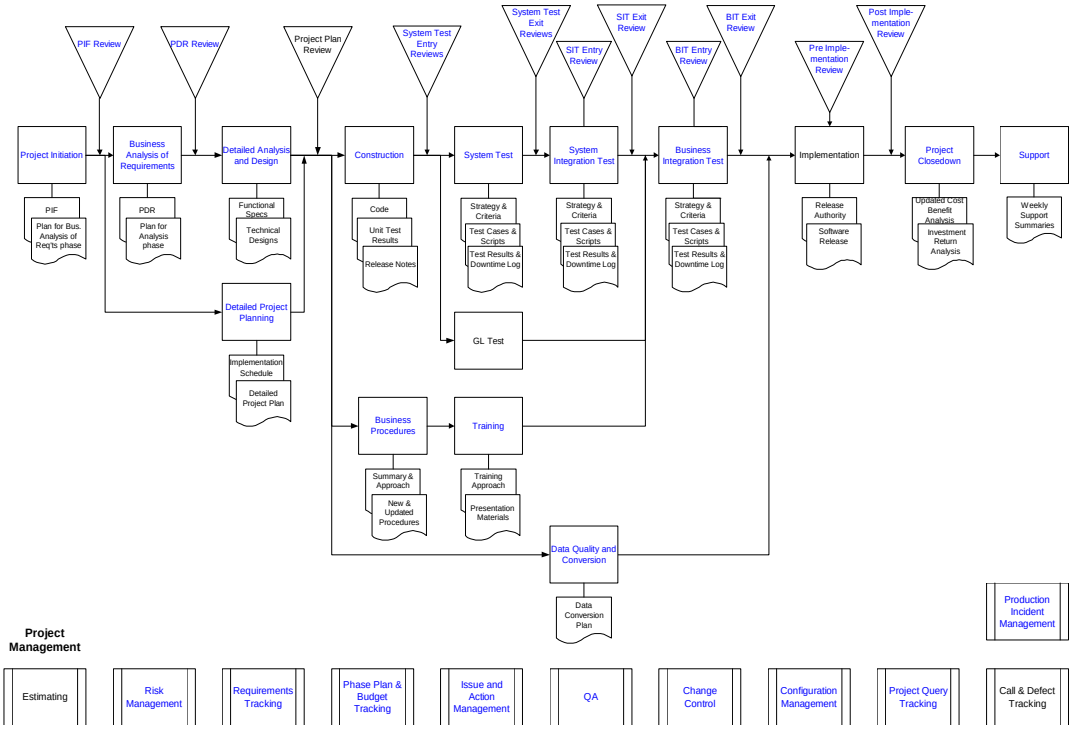
This example shows a diagram of seven nodes; a sequence of five technical activities overseen by a project management activity and supported by a QC and CM activity These relationships are implied by the positioning of the nodes, but specifics are absent.

The value of the preliminary architecture is the scoping and naming of working practices as a basis for shared understanding, or at least agreement, and as the foundation for further development. It is unlikely that the picture reveals much about the way work is performed that was not already known but does categorize activities and provide an initial shared view and focus for understanding working practice and a basis for discussion and debate.

Much of the value of these preliminary models depends on the manner in which they are developed or introduced. Collaborative working to produce a symbolic representation; that assigns names to groups of activities, reflects business priorities and, without being too dogmatic, introduces relationships between the groups of working practices can set the scene for further development of working practices. (These symbolic architectures, because they can be visually attractive - exhibiting symmetries and pattern - and involve categorization and grouping of working practices can provide the basis for a useful home page for a nascent process group with links leading to more detailed description or examples of working practices.)

Intermediate (Lifecycle) Architecture:

The intermediate architecture can evolve from the preliminary. The intermediate architecture is a more detailed picture with activities, and sometimes decision points, reviews and deliverables shown as well. This lifecycle type diagram usually has sequencing of activities implied by horizontal or vertical positioning of activities and with arcs connecting the nodes. Typically the meaning of the arcs is not specified. An example is shown below.



These activities may also be grouped by a sequence of lifecycle stages or phases.
The intermediate architecture has the same value as the preliminary as a vehicle for a shared overview of working and management practices. The additional detail is useful for showing more about activities and identifying work products and documents and when they are produced, but some of the limitations begin to show themselves too. The increased detail introduces increased complexity. The patterns presented by the symbolic representation become blurred or lost under the increased complexity. The complexity itself can become problematic. The implied relationships of the symbolic representation have become explicit and their appropriateness, and the tailorability and flexibility of the model for diverse projects is open to debate. A warning sign is the emergence of a plethora of variants to deal with different situations or a diagram cluttered with ‘exception handling’ options. Most importantly as the diagram attempts to describe the technical activities some activities are found to have no clearly prescribed place. This is illustrated in the example above. Estimation, change control, risk management do not have a location but ‘float’ on the margins of the diagram – because of the timing or sequencing orientation.

It is rare that an intermediate architecture is presented using a formal notation so quality control by formal technical review is difficult. This limitation, coupled with the complexity can lead to unhelpful or even incorrect architectures.

When these limitations show themselves it is time to develop a true software process architecture.

True Software Process Architecture:

A true software process architecture is a technical artefact that can be used by process engineers for a variety of process engineering tasks – process design, diagnostics and improvement. It is used to show how software development processes work (or should work, or will work) together as a coherent set to meet the needs of the software development organization. Without this processes and working practices, developed and assembled ad hoc may well not function together well. Without the architectural perspective deficiencies in working practices, and opportunities for innovation may be missed.

The true software process architecture uses a formal notation[2] that allows the quality of the architecture to be evaluated (quality controlled) and permits assessments of architectural completeness and scope.

As a technical artefact it may lose some of the benefits of the preliminary and intermediate architectures but has far greater value in revealing ways of working and providing a powerful tool for managing change. A true architecture, accurately reflecting the relationships between software development processes used, or to be put in place, may be complex, idiosyncratic and difficult to readily understand. Consequently it may have limited value as a learning or communications aid for general use. Simplified diagrams, perhaps reflecting a lifecycle or procedural organization may be derived or abstracted from the true process architecture to fulfil these secondary needs, but it is not wise to degrade the process architecture to serve as both technical definition and as communication aid any more than it is sensible to compromise the technical documentation for a complex system or piece of equipment it order to make it serve as the user manual too. Distinguish between, and value, the two functions.

The software process architecture is, not surprisingly, process oriented. It shows how processes interact with each other. It is not the same as a lifecycle, which organizes activities in a sequence of chunks to aid the management[3] of software development, neither does it

[2] If it is unacceptable or difficult to use a formal notation then use other familiar notations but ensure that some standards or rules for the notation are used. – consistent (and therefore constrained) typing and naming of nodes and (often omitted) consistent naming of the arcs).

[3] Lifecycle models are management, not technical, tools. Their defining characteristic is the 'chunking' of work into a sequence of manageable pieces. Criteria for what goes in the chunks vary widely, depending on circumstances, but the presence of chunks is universal.

reflect organization departments, boundaries or functions - which implies that a software process architecture may cross organizational boundaries, and can be extended to become an organization process architecture.

The primary components of the architecture are those processes that are important for the development and support of software, that make sense to software developers and managers, have value to them, and are relatively independent of the means by which they are implemented, by specific practices or tools, and the interactions between them.

The development of the process architecture is similar to the modelling step used in process definition (See C – 2):

a) Determine the scope of the architecture. This may be all the working practices and activities within an organization needed to develop and support software. It may be limited to particular aspects of the development support activity - especially if the architecture is undertaken as the basis for process exploration. Record the agreed scope.

b) Brainstorm and list concepts related to the process areas in scope.

c) Identify the entities involved in the process areas. These may include:

- people (roles and responsibilities)
- interactions
- triggers
- activities
- documents/work products
- libraries/repositories
- environment
- tools
- etc….

d) Map concepts onto entities to determine coverage. That is, list concepts against each of the entities and determine how well balanced the distribution is. The objective of steps a, b and c is to determine the understanding and coverage of the architecture by early ideas.

e) Develop a context diagram (level 0) for the architecture – define the boundaries – what's in, what's out. Then identify the:

- requirements or needs
- outcomes
- controls or constraints.

When developing the context diagram the input and outputs may also be indicated but this is far less important at this stage and at this high level. It is more important that the requirements and outcomes are identified that show the business rationale. Inputs and outputs will be tend to be more important in showing the relationships of elements within the architecture, along with other interactions, and inputs and outputs to and from the architecture will reveal themselves.

f) Decompose (elaborate) the context diagram to level 1[4]. To begin with it is sometimes easier to begin by diagramming information flows and then develop understanding of the other interactions – uses, creates, triggers, manages etc. This is an important activity worth spending time on and taking care with. Take care to reason about the entities being diagrammed – their lifetimes and relationships. Expect to redraft the diagrams several times. It is not unusual to wish that the diagram had extra dimensions. When this happens see if processes can be categorized, e.g. utility processes, line processes, management processes. Similarly if some processes recur a category of processes is emerging. It may help to organize and reorganize the architectural layout (placing of nodes wrt to each other) to aid in understanding (I like IDEF0 where the notation's constraints tend to impose a layout that reveals more about the relationships between process areas in a way not otherwise revealed by the notation.)

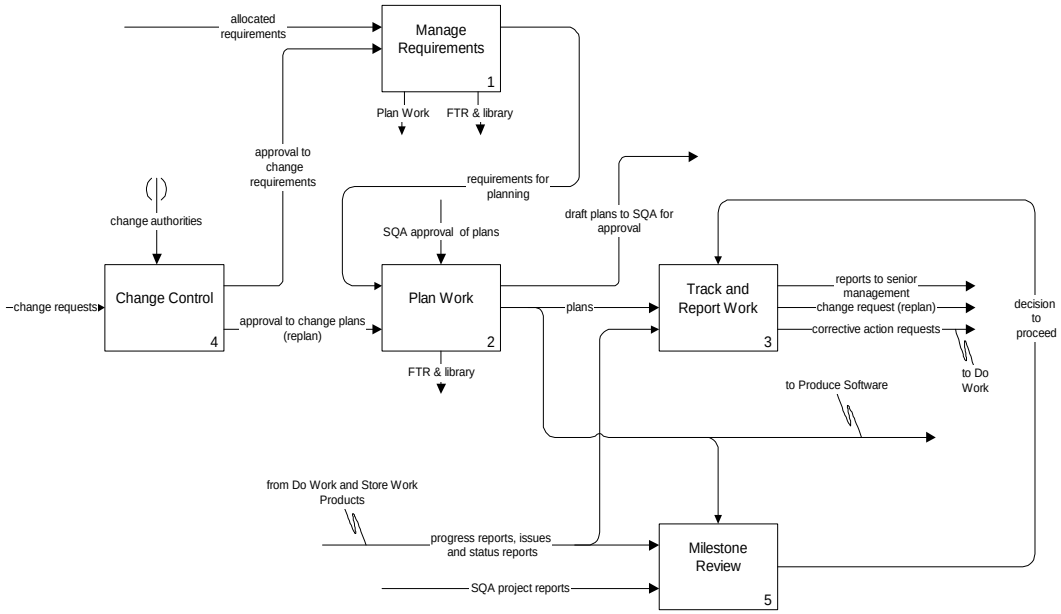
At the end of this activity well understood process areas with well-defined, relationships, often only tacitly recognised before, if at all, will be captured.

f) *Attempt* to decompose level 1 diagrams to level 2. It is unlikely that useable level 2 diagrams can be produced at this stage but it is useful to attempt to produce them to test understanding of level 1 diagrams. It is often of value to draft a commentary or narrative describing a walkthrough the level 1 diagram. This tests understanding and can reveal gaps or redundancies.

[4] It is work considering what types of interactions are of interest when drafting the level 1 diagram. Nodes will be processes, but the relationships between them, indicated by arcs, and to a lesser extent by relative position, can be of many types – what types are of interest? Revisiting steps b and c may be helpful here. A knowledge of process modelling notations may be of value too.

g) Formally review (QC) the context diagram and level 1 diagrams against any organizational policy or process requirements. Also review for completeness and consistency.

An element of a software process architecture is shown below. This example uses IDEF0:



With an architecture complete and reviewed it should of course be managed as a valuable organizational asset.