Software Engineering Components

Product - result

Problem - requirements and specification

Process - organization and actions to get results

In developing a project which entails software engineering, I would organize the project in the order above.
Software evolves
  Takes on a life of its own
  New hardware, new systems, new needs, handle more complexity

Software experiences
  Used to be many “horror” stories but they have subsided as we are better able to perform software engineering effectively.
Software characteristics

⇒ Software doesn’t wear out. Failures to “design” flaws.

⇒ Software is mostly custom built.
   New technology enables frameworks, components, etc.

⇒ Software manufacture is almost trivial

⇒ Software systems - is there a crisis?
   Yes and no
Myths can be strawmen used by “salesmen”.

⇒ A general statement of the objectives is sufficient to begin writing programs

⇒ Situations where problem, solution, and experience is lacking. Apollo is a typical example.

⇒ Software creates voluminous and unnecessary documentation.
  ⇒ This can be true or false.
  ⇒ “Good” software engineering recognizes documentation is communication and “corporate” memory.
Software Problem Definition

Requirements and specifications

⇒ System definition
  A set or arrangement of elements organized to accomplish some predefined goals by processing information

⇒ System elements may be:
  ⇒ Software - computer programs, data structures, documentation, etc.
  ⇒ Hardware - electronic computational systems, sensors, network and communication elements, actuators, etc.
  ⇒ People - operators, users, etc.
  ⇒ Documentation - description information concerning the system and how to use it.

⇒ Elements are combined to transform information, accomplish useful objectives
It should specify behavior, constraints on implementation, any thoughts about life cycle (use), responses to events, etc. Like other documents in engineering process, it should enable evolution.
Requirement and Specification Techniques

⇒ Scenario Analysis
  ⇒ Viewpoint, Service operation and service actions
  ⇒ Scope of viewpoints
    ⇒ Data sources & sinks
    ⇒ Frameworks - organization & structure
    ⇒ Users of service
    ⇒ Environment
  ⇒ Scope of service
    ⇒ data flow, event, hierarchy - you illustrate with charts & diagrams
⇒ Remember to consider the all parts of system.
Divide and conquer

⇒ Multiple level functions and/or objects

Component Eng.

Software Eng.
Information Flow Models

Flow charts (PSpec)

This should not be new to any of you
State Transitions Flow Charts (CSpec)

- **Start Copying**
  - **Read Commands**
    - **Done**
      - **Empty tray**
        - **Diagnose Problems**
          - **Malfunction**
            - **Malfunction Fixed**
          - **Reloading Paper**
            - **Idle**
              - **This loop could go to make copies**
    - **Make Copies**
      - **Start Copying**
## Other Req. & Spec. Techniques

- **Pre- & post- conditions**
  - Operators - `<`, `==`, `not`, `or`, `for_all`, `there exists`, etc.
  - Predicates - a Boolean expression
- **Algebraic**

<table>
<thead>
<tr>
<th>Type:</th>
<th>name of spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>imports:</td>
<td>list of specification names</td>
</tr>
</tbody>
</table>

| Informal description of type and its operations |

| Operation signatures setting out the names, parameters, and results of the operations |

| Axioms defining the behavior over the type. They relate the operations used to construct the entities with the operations used to inspect its values. |
Structure Program Development Languages

⇒ PDL’s may be derived from actual programming languages
⇒ Use when:
  ⇒ sequence of steps like a program execution is needed
  ⇒ when specifying programatic operation
⇒ Difficulties
  ⇒ the PDL may not be rich enough linguistically
  ⇒ specifications at this detail may obscure a better design later
Objectives - Enable management and technical personnel to:

⇒ comprehend the problem and solution strategy

⇒ make reasonable estimates of resources needed

⇒ make the decisions to get and keep the project on track
  ⇒ risk analysis - chapter 6
  ⇒ setup initial scheduling and tracking - chapter 7
  ⇒ establish software configuration management - chapter 9
Goal - to get needs committed early

⇒ Human
   ⇒ Commitment, experience, environment, support, etc.
   ⇒ Estimation (Hindsight is 20/20 but costly)
      ⇒ Base on comparable projects, experience
      ⇒ Decompose and estimate subproject costs
      ⇒ Use empirical software estimation models
   ⇒ Make/buy decision
   ⇒ Relate to specs, designs, code, test data, etc.
⇒ Off-the-shelf components
⇒ Full-experience or partial-experience components
⇒ New components - by far the highest risk decision
Project Estimation

⇒ Based on the use of project metrics
⇒ 1. Subdivide project into “logical” units (LU) Function points, object points, code units with LOC estimates, etc.
⇒ 2. Weight the difficulty of each $S = (S_{opt} + 4S_m + S_{pct})/6$
⇒ Use Estimation model equations $E = A \times B(S(LU))$
⇒ Compare results to past project histories
⇒ The logical units can be anything which makes sense to the project and people
⇒ Makes estimates during project stages
  ⇒ COCOMO model of Boehm
⇒ Correlate estimates with finished projects - feedback
⇒ Estimate different things Cost, schedule, personnel needed (possibly by type), etc.
A very important part of project management planning

⇒ Provides for an intelligent progression of project subtasks
⇒ Provides insight into the inter-dependence of subtasks
⇒ Compartmentalization - same principle as subdivision
⇒ Relate to metrics and project planning (FP, OP, etc.)
⇒ Time & effort allocations - Determine the time & effort (resources) needed for each task
⇒ Milestones & Deliverables
⇒ Timeline chart or alternatively PERT and/or Critical Path charts can be used