BUILDING AND USING A WORK BREAKDOWN STRUCTURE

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com
+1-(520)-544-2881

Real-time systems development is both technically challenging and challenging to manage too. The key to project management is the work breakdown structure, which can be remarkably complex even in a relatively small real-time system.

This paper shows how to:

- divide a system into subject matters
- divide those subject matters into manageable chunks
- organize the chunks into a coherent structure, which can be understood by techie and manager alike
- identify dependencies between the chunks, which can then be used for scheduling
- manage the project using the work breakdown structure
- build and execute a test and integration plan

1. THE STRUCTURE OF THE WORK

The structure of the work should match the technical content of the project. Hence, the first step in building a work breakdown structure is to understand the subject matters that comprise the system.

Subject Matters. A system comprises several subject matters. A system for managing a chemical plant, for example, has at least the following subject matters:

- the chemical plant application itself
- the application user interface (AUI), which lays out the rules and concepts required to support the application, such as menu layouts, and 'live' screens.
- the graphical user interface (GUI), which comprises a set of general graphical facilities, such as icons, windows and scroll bars
- process input/output (PIO), which controls the special purpose hardware
- alarms facility
- software architecture, which proclaims and enforces the system-wide rules for the organization of data, control, structures and time
- the operating system, and
- the programming language

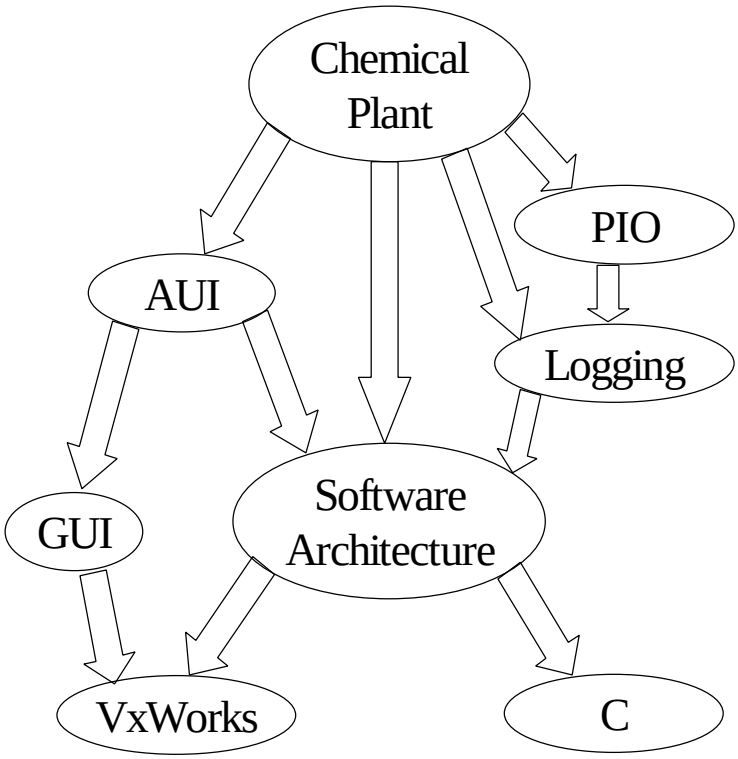


Figure 1: A domain chart for the chemical plant system

Each of these subject matters represents a body of knowledge that can be understood by a process of analysis. We call each subject matter a *problem domain*, or *domain* for short. We may diagram the domains in a system as shown in Figure 1.

Work To Do. A method prescribes what to do, and the outputs from each step in the method. Thus, if you using Booch (with or without UML), you will:

- Identify classes to produce a class diagram;
- Identify the semantics of these classes and operations;.
- Identify the relationships among the classes
- Implement these classes and objects

While if you are using Shlaer-Mellor, you will:

- Identify classes, and produce a class diagram;
- Identify their behavior, to produce a state chart for each class;
- Identify the functions required to produce the behavior, to build action language for each stage in the object's lifecycle.
- Publish the interfaces to clients

Each prescribed output, which is the result of doing the work involved in the step, is an "artifact," an internal or external deliverable.

We can therefore represent the work to be done, as well as the associated artifacts, as a matrix with domain names across the top and the names of the as row headings.

	Chemical Plant		AUI	Software Arch.		PIO	Logging
Class Diagram							
State Chart							
Actions							
….							

	GUI	OS	Language
Identify criteria			
Identify candidates			
Score candidates			
Purchase			

Figure 2: Preliminary project matrices for chemical plant application

Realized Subject Matters. Some subject matters already exist as code. These subject matters do not require modeling, though there is a method to incorporating them into the project. For example, you may decide on a systematic approach to software acquisition comprising the following steps:

- Identify selection criteria;
- Identify candidate suppliers;
- Score candidate products;
- Purchase product;

Preliminary Project Matrices. We may represent the work to be done as a set of matrices, with domain names across the top and the names of the steps as row headings. Because the approach taken to building subject matters from scratch is different from that used for software acquisition, there is a different matrix. In general, there will be as many matrices as there are different approaches to the work. We show the preliminary matrices in Figure 2.

2. SIZING THE BOXES

Each box on the matrix represents both an artifact and the work required to produce it. These boxes must be of "reasonable" size, so that two to four people can do the work in two to four weeks.

Subsystems. Subject matters derive from the underlying technical content of the work, and as a consequence, they are of arbitrary size. Some subject matters, such as the Chemical Plant, might contain several thousand conceptual entities, while others, such as Alarms may be extremely small—only a dozen conceptual entities. To make the work manageable, we must break large subject matters into smaller ones, called subsystems.

Note that we do not agglomerate small subject matters into larger ones. If Alarms is very small, so be it.

Class Blitz. To reach an estimate of the size of a subsystem, think of all the concepts you can that belong in a particular domain. Frequently, you'll find that the concepts come in waves. You come up with a concept or two, then a flurry of related concepts present themselves. Then the wave slows…until a new wave appears.

This activity is best carried out in a group, with an appointed scribe. Do not be too critical during the "class blitz." Good ideas come from many sources. However, once the blitz is complete, do be sure that the proposed classes really do belong in the domain. Domain boundaries need to be crisp, and it's struggle to maintain them.

Estimates. The amount of effort required to complete a cluster depends primarily on the method. The work involved building a lifecycle model, for instance, depends on what the inputs to the work are, and the level of detail required for the model to be acceptable for the next steps. The best estimates for the effort come from your project, so as the project proceeds, we must gather metrics to refine our early estimates.

In the absence of such metrics, use data gathered from previous projects.

In the absence of a previous project using the same method, use the metrics provided by the supplier of the method.

Refactor. At this point, you may wish to consider refactoring the steps of the selected method. If a step consistently seems to be too large, and the step requires several outputs, you could choose to divide the step into several steps. Conversely, if some steps are too small, you could agglomerate them, if and only if the steps are closely related and require roughly the same people skills for their construction.

3. MAPPING BETWEEN DOMAINS

Each domain may be understood completely separately from all others in respect of the semantics of that domain. An analyst can understand how the chemical plant works without knowing about user interfaces or operating systems. However, there is some work to be done to link the domains.

Work Between Domains. Consider a purchased process I/O domain that needs to be configured. According to the documentation of the purchased package, we must create a file that defines the mapping between the elements in our application and the physical signals that the PIO package manipulates. There are, therefore, three components, though we have accounted so far for only the first two of these:

- the application,
- the purchased package, and
- the mapping between the two

In general, whenever there is an arrow on the domain chart (Figure 1), developers must supply the mapping for each one, called a bridge. The mapping is asymmetric: one domain, called the *client*, makes use of another, called the *server*. Hence, there is a mapping from the application to PIO, to logging, to the architecture, and the AUI; from the AUI to the GUI and the architecture and so on.

The form of the mapping is always defined by the server domain. In the example above, the suppliers of the PIO package determined that a file was the appropriate way to configure the package. They could have chosen to build multiple files, to require the work to be done interactively, to do it graphically—almost anything. Whatever forms are chosen by the supplier, that form is independent of the particulars of the application.

There is, therefore, a method for the construction of the mappings. We may then add each "method" as a step in the matrices. Hence, for the application domain, there is a step to map to the PIO (the file in the example), to logging to define the events of interest, to the architecture to implement the logic and data, and to the AUI to define the screens.

Note this last. The AUI domain defines restrictions on the general package supplied by the GUI, with conceptual entities Live Window (a window with live data that cannot be occluded), Application Menu which adheres to certain conventions, and so on. However, there are, at this point, no actual screens defined. The mapping between the application and the AUI takes on this task. The step may then be divided into parts: define the screens and then fill in the configuration files.

Because the mappings required differ from domain to domain, you may choose between building several different matrices, each with different mapping steps, or showing everything on one matrix with inapplicable steps removed.

4. USING THE MATRICES

Estimation. Construct rough estimates of subsystem size by picking one, probably the one you intend to tackle first, as the reference subsystem, and then working out relative sizes for the rest. Similarly, estimate the relative sizes of the steps, using one step as the reference step of size one. We illustrate the results in Figure 3.

From an estimate of the size of the "reference box," we can propagate estimates across the entire system, providing a first cut at an estimate for the entire project.

Ordering the Work. Even though domains are semantically separate, there are some dependencies between domains. But, certain properties of the application, a need to control a value periodically say, may drive certain capabilities in the server, such as the need to set values based on time. Further, the mapping between the domains cannot be completed until the server is defined. During specification and modeling, then, a server domain cannot be absolutely guaranteed to be completed until all its clients are complete. Practically speaking, though, we often know enough to be able to go ahead anyway.

	Chemical Plant			
	Inventory	Transfers	Recipes	Control
Class Diagram				
State Chart				
Actions				
Map to Arch				
Screen Def'n				
Screen Files				
PIO Config.				
Logging Config.				

	Services			
	Arch.	AUI	Logging	PIO
Class Diagram				
State Chart				
Actions				
Define Interface				
Map to Arch				
Logging Config.				
Screen Def'n				
Screen Files				
PIO Config.				

	GUI	OS	Language
Identify criteria			
Identify candidates			
Score candidates			
Purchase			

Figure 3: Matrices for Application, Services and Packages

Scheduling. These estimates, together with the ordering implied by the domains, provide the basis for project scheduling. Such fundamental ordering still leaves considerable choice in the order of the subsystems. There are many possible approaches here; one generally favored is to work the riskiest subsystems first. Risk factors include:

- the need for an example;
- subject-matter complexity;
- performance constraints; or
- method.

Express the schedule in a form appropriate to the various users—clients, other managers, program investigators etc.

Actuals. As the project proceeds, collect metrics on the amount of work completed. Experience indicates that passive metrics are easier to obtain and more reliable than estimates that require active participation. One relatively easy approach is to instrument your modeling and system configuration management tools to produce files that indicate the amount of work done. For example, a modeling tool can produce a report nightly that shows the names of all classes, attribute and operations modeled. By comparison with the previous report, we can determine the amount of work done and the amount of "churn."

Note that passive metrics are not as precise as active ones because, for a given person, it is not clear how many hours were actually worked, nor exactly what has been worked on. However, gross figures are available: vacation days, reporting to different projects when a person is shared, and so on. We certainly will have a notion of whether a team is working 40, 60 or more hours a week. Similarly, we know who is assigned to which tasks, even if the person also takes part in reviews, answering questions from new hires, unjamming the printer (again),and so on. This is life on a project; there are always distractions and other tasks that need to be done.

Should you require more data about a particular activity, drill down and gather data on that subject only. For example, you could ask that every person track time spent on reviewing other work for a certain period. Or put a stop clock by that #@%$ printer.

Artifact Management. The project matrix provides a strong organizing force on the project. Use the matrix to identify, track, and configure the artifacts. One scheme we have found helpful is to provide an artifact identifier of the form:

<project>.<matrix>.<column>.<row>.<x>

so we might label the collaboration diagram for the state chart of the AUI in the Services matrix on the Chemical project as:

Chemical.Services.AUI.StateChart.Collaboration

or some numeric or key code abbreviation such as cp.sv.2.2.3 (the third document in the second row in the second column.)

Assignments. Always assign teams with two, three or four people. In the context of programming, this is sometimes called "pair programming." Use the matrix to indicate which team is taking on each box.

Status. Use the matrix to show completion status, version number, end date, start date, etc. Some projects maintain the matrix on a wall in the main area for the project. Everyone knows where the project is.

Refactor. As the project proceeds, refactor the matrix based on what you learn. Often, new subsystems appear, especially subsystems for tooling.

Other Uses. Every project seems to find a new use for the matrix. The reason is that the matrix provides a single unifying theme that connects both management and the technical work in all its phases and forms.

5. Build and Execute a Test and Integration Plan

There is an old saw that 90% of the time goes into analysis, design, and code, while testing and integration takes the remaining 90% of the time. But this doesn't tell the worst of it: testing and integration takes place late in the project, when there is too little time to react.

Specification Time. One way to address this problem is to put testing up front in the project. Therefore, no box can be said to be complete until it has been reviewed. Indeed, each box can be cast with a pre- and post-condition set which defines what must be done before the work can begin, and what must be true when it is complete. Further, whenever one step is the input to another, the team that is to do the work on the subsequent step must explicitly accept (or reject) the work of the previous step.

This approach can significantly reduce the total number of errors during the construction phase, especially when coupled with executable specifications.

Construction Time. During the construction phase, the system can only be and integration tested bottom up. For example, the application may use the PIO domain to get and set values in the plant. Clearly, we cannot test and integrate the application completely until the PIO is tested and integrated. Further, we cannot completely test and integrate PIO until the architecture is complete, which in turn depends on the operating system and programming language. Indeed, the dependencies shown on the domain chart describe the *reverse* order for integration.

These dependencies must be used to order the later steps in the matrix.

Just as, practically speaking, one can proceed with specification in the absence of a complete definition of all the client domains, so too can we, during construction, test and integrate without complete implementation of the servers. For example, we could build a subset of the subsystems in the architecture (single tasking without optimization, fault recovery or other higher-order facilities) and only the input subsystem of the PIO. With just these small parts, we can offer the hardware engineers enough software to verify that the hardware works properly. Such considerations may affect the selection of subsystems during the initial construction of the matrix.

The overall plan then is cast as a part of the matrices.

6. SUMMARY

The project matrix provides a framework for managing a project that is accessible to managers and technical people alike. It provides a framework for estimation, scheduling, documentation, metrics, and status as well as testing and integration. The approach is independent of the method or notation that you choose.