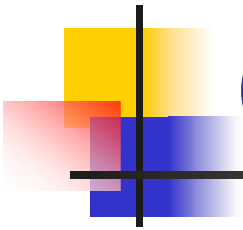
Software Process Modeling

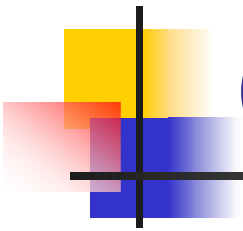
Sadaf Mustafiz
School of Computer Science
McGill University

Winter 2003



Outline (1)

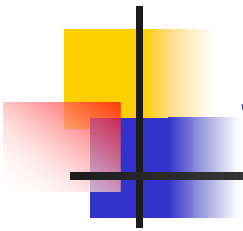
- The Need for a Defined Software Process
- Objectives of Software Process Modeling
- Current Software Process Models
  - Waterfall Model
  - Spiral Model
  - Incremental Development Model
  - Shortcomings of Current Models
  - Causes of Current Model Problems
- Process Modeling Considerations



Outline (2)

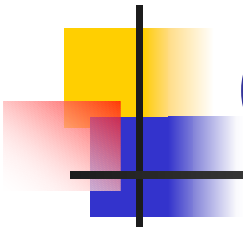
- Entity Process Model
- Differences of EPM and WM
- Stability of EPMs
- Entities and Software Process Entities
- Producing Software Process Models
- Example EPM
- Scheduling Considerations

The Need for a Defined Software Process



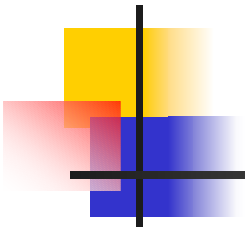
- The software process is the technical and management framework established for applying tools, methods, and people to the software task.

- A thoughtfully defined and approved process can be a great help for the software professionals to do a consistently professional job.



Objectives

- to assess and analyse the as-is process
- to design the to-be process
- to forecast process trajectories for a better project control
- to simulate outcomes under different what-if conditions
- without affecting the actual environment



Process Models

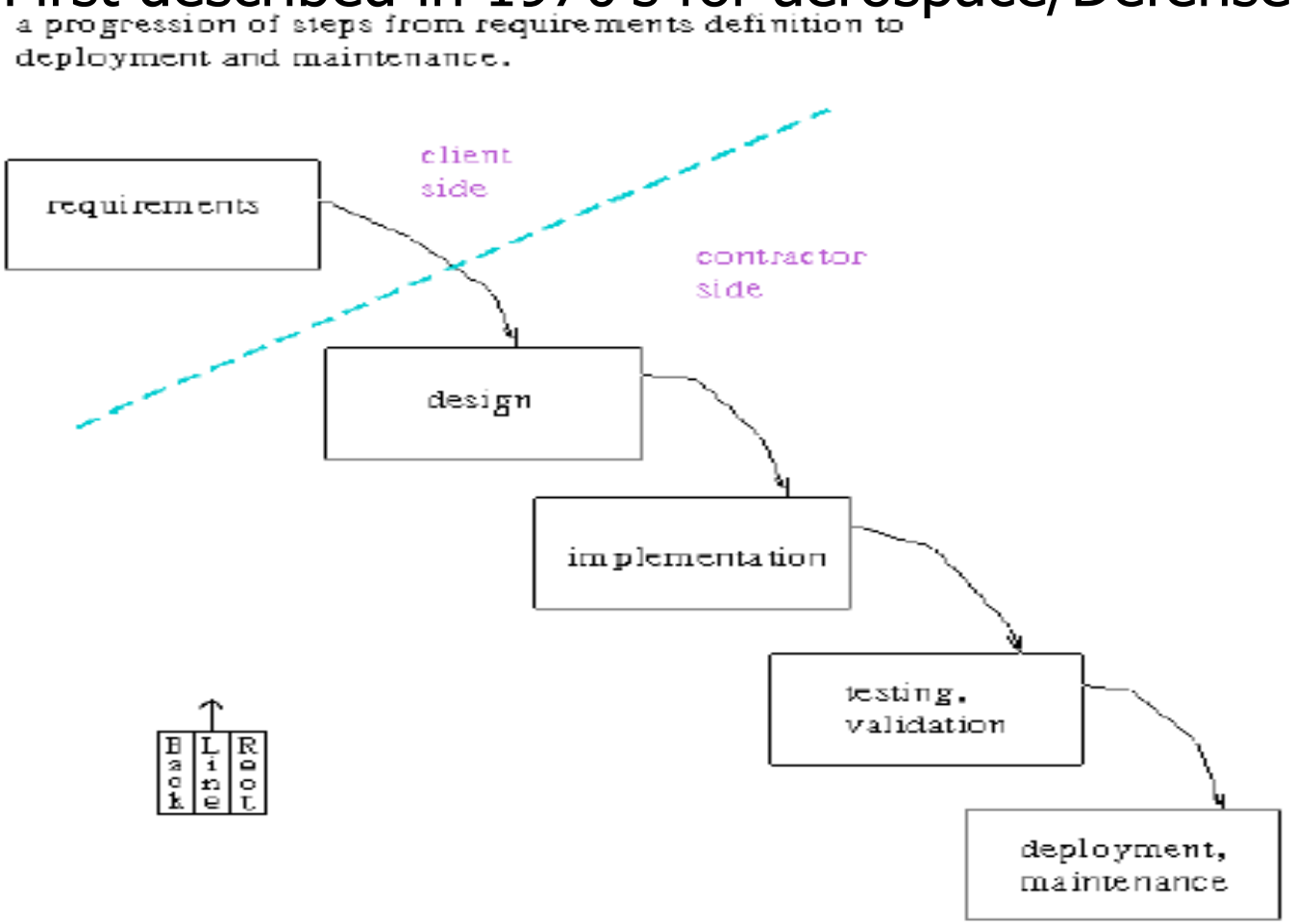
- Represent the way the work is actually (or is to be) performed.
- Provide a flexible and easily understandable, yet powerful, framework for representing and enhancing the process.
- Be refinable to whatever level of detail is needed.

Current Software Process Models

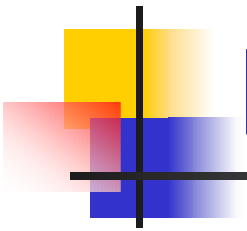
- Waterfall Model
- Spiral Model
- Incremental Development Model
- Entity Model
- …

Waterfall Model (the traditional view)

- First described in 1970's for aerospace/Defense projects.

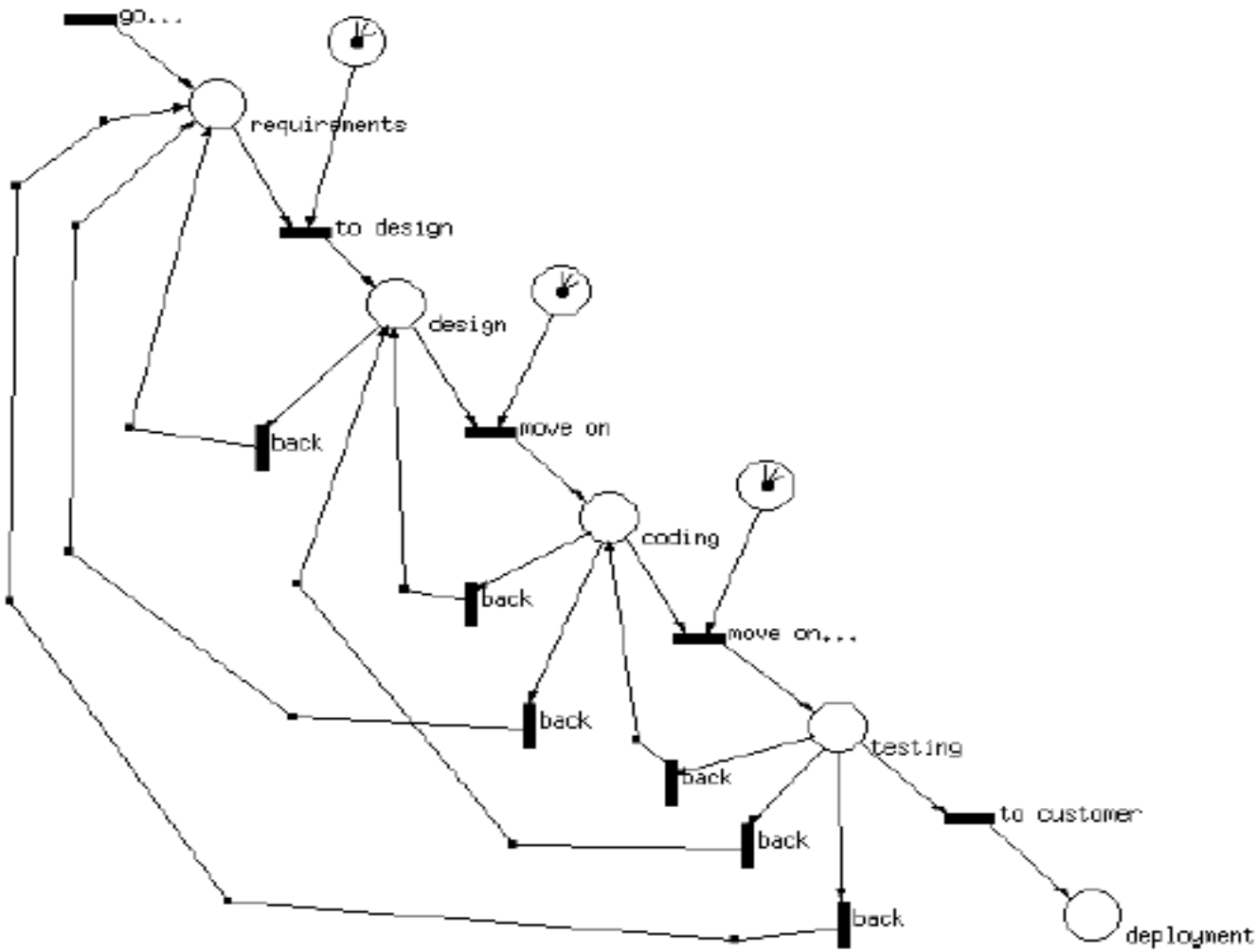


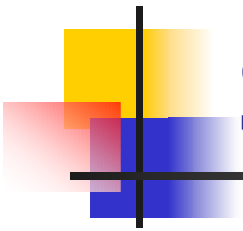
Shortcomings of the Waterfall Model



- It does not adequately address the pervasiveness of changes in software development.
- It unrealistically implies a relatively uniform and orderly sequence of development activities.
- It does not easily accommodate such recent developments as rapid prototyping or advanced languages.
- It provides insufficient detail to support process optimization.

The Reality…


go...
requirements
to design
design
back
move on
coding
back
move on...
back
testing
back
to customer
back
back
deployment

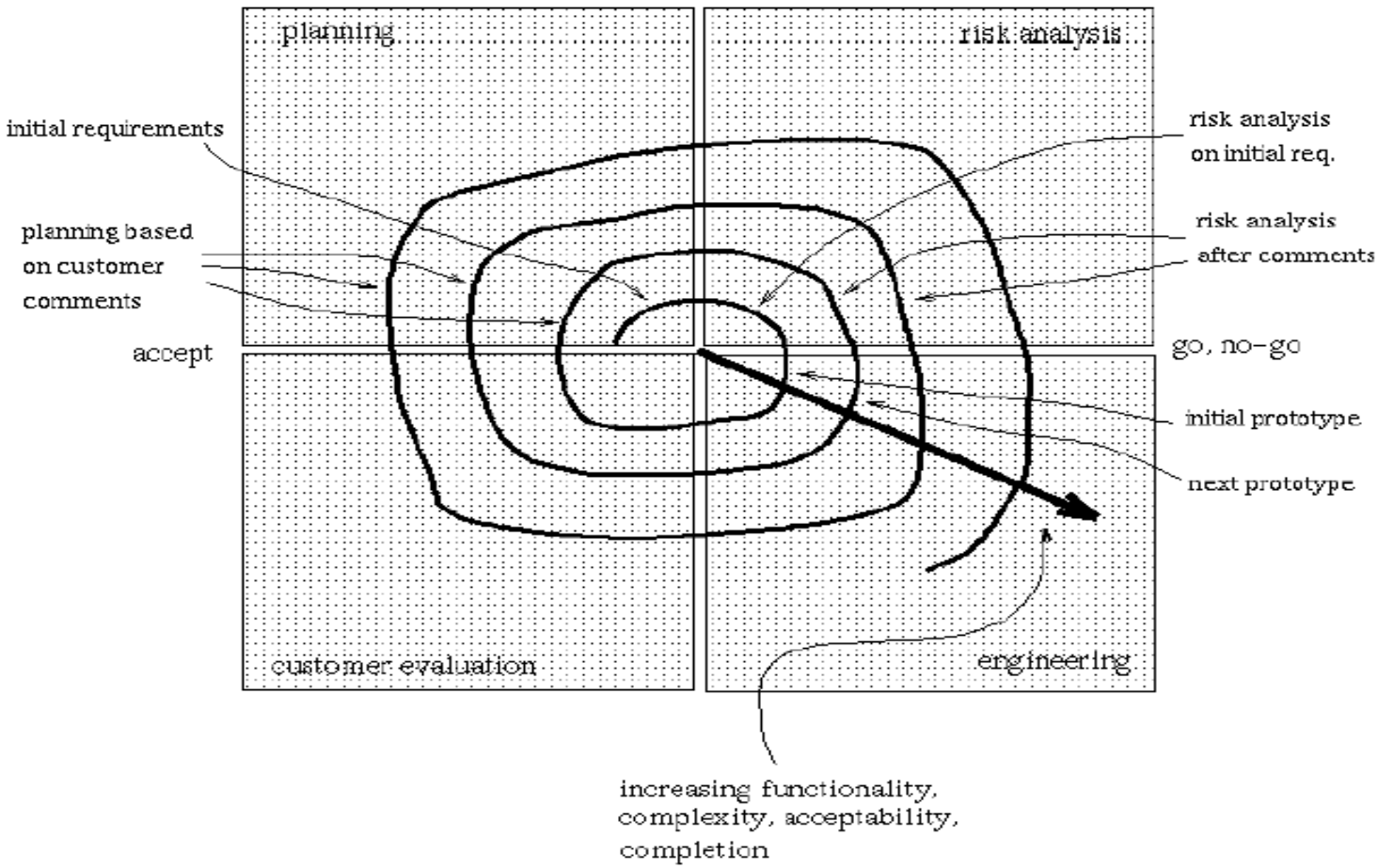


Spiral Model (1)

- This is a refinement of the traditional waterfall, explicitly recognizing that development cycles. The spiral incorporates risk analysis into the process, and allows developers to stop the process as well as clients, depending on expected returns from new requirements.

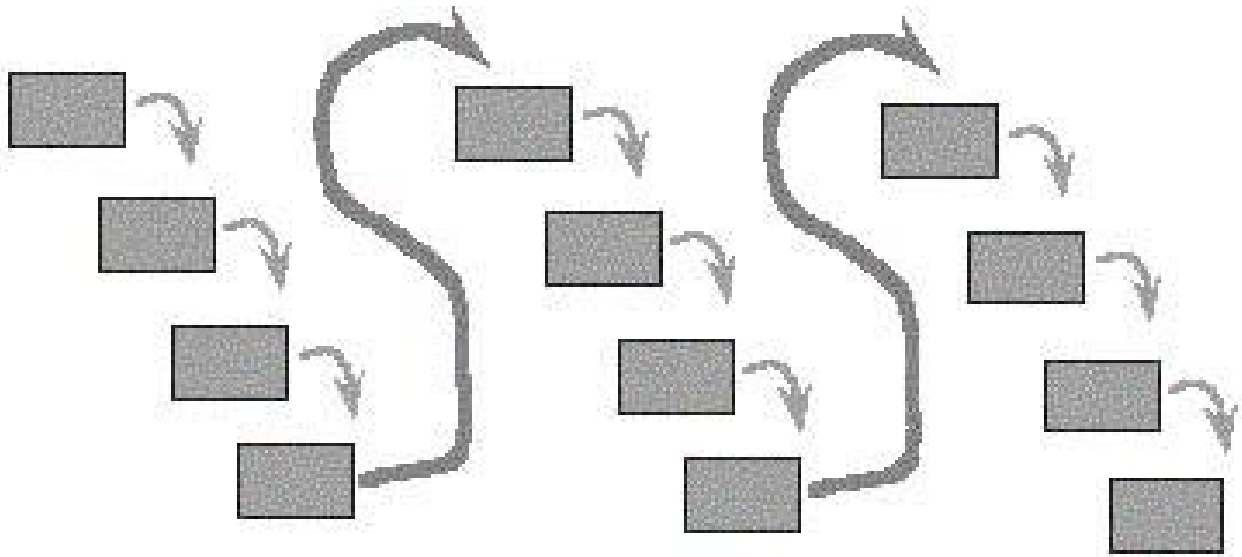
Spiral Model (2)

- Described by Barry Boehm as a metamodel…



Incremental Development Model

- Each release is a mini-waterfall

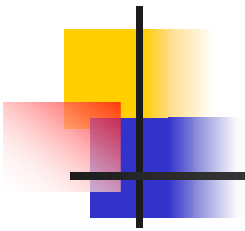


Causes of Current Model Problems

- The fundamental problem with current software process models is that they do not accurately represent the behavioural (or timing) aspects of what is really done.
- Traditional process models are extremely sensitive to task sequence; consequently, simple adjustments can require a complete restructuring of the model.

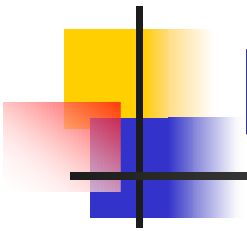
Process Modeling Considerations

- "What is the right way to model the process?" but "What is the most appropriate way to model this process for this purpose?“
- Most efforts have focused on the functional, or task-oriented, aspects of processes.
- The Entity Process Model proposes an entity orientation to behavioural modeling.



Entity Process Model

- Considers basing process models on entities. Here, one deals with real entities and the actions performed on them. Each entity is a real object that exists and has an extended lifetime.
- Examples: the requirements, the finished program, the program documentation, or the design.



Differences of EPM and WM

- The traditional waterfall model deals with tasks such as producing the requirements. This task is then presumed completed before the next one (design) starts.
- In reality, the requirements entity must survive throughout the process. While it undergoes many transformations, there is a real requirements entity that should be available at all later times in the process.
- The same is true of the design, the implementation, and the test suite.

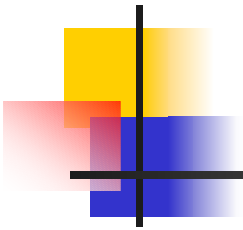
The Stability of Entity Process Models (EPMs)

The reasons that EPMs provide a useful representation of a software process are:

- EPMs deal with real objects (entities) that persist.
- Each entity is considered by itself and is viewed as having a defined sequence of states.

The Stability of EPMs (2)

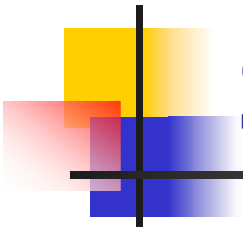
- State transitions result from well defined causes, although they may depend on the states of other entities as well as process events and conditions.
- As long as the relative sequential relationships of these transitions are retained within each entity stream and as long as any prerequisites and dependencies between entities are maintained, the timing within the various entity streams is not material.



Entities

An entity must:

- Exist in the real world and not merely within the model or process.
- Be identifiable and uniquely named.
- Be transformed by the process through a defined set of states.



Software Process Entities

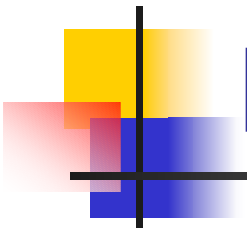
Some obvious entities are:

- Deliverable code
- Users' installation and operation manuals

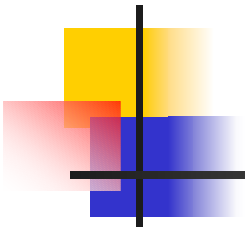
Some more entities are (debatable):

- Requirements documents
- Design
- Test cases and procedures

Producing Entity Process Models

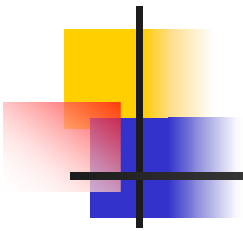


- Identify the process entities and their states.
- Define the triggers that cause the transitions between these states.
- Complete the process model without resource constraints — an unconstrained process model (UPM).
- Impose the appropriate limitations to produce a final constrained process model (CPM).



Example EPM (1)

- Modeled using a commercially available software system called STATEMATE.
- Focuses on behavioural modeling perspective
- Approach to behavioural modeling utilizes statecharts



Example EPM (2)

Considering the activities occurring between the time when

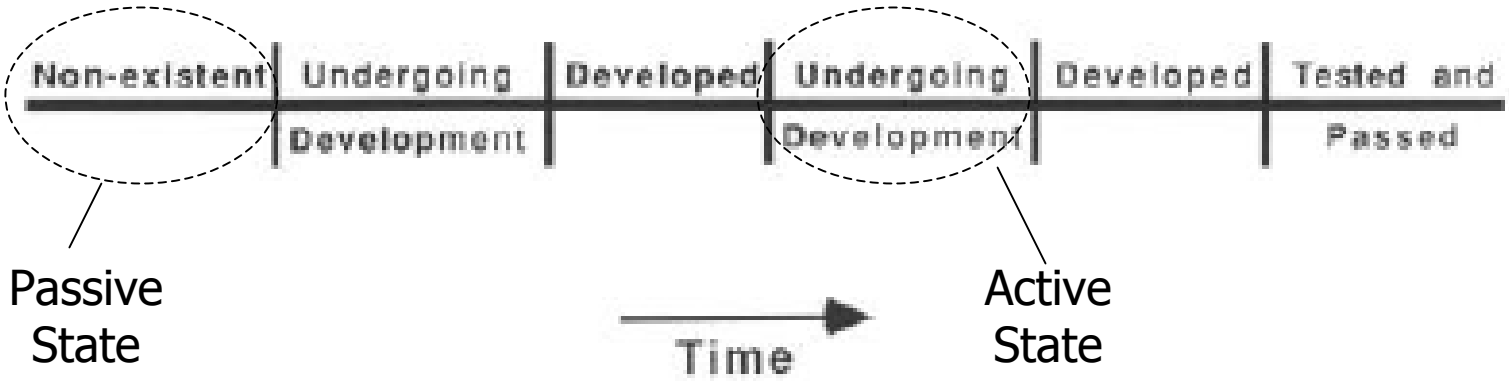
1. detailed design for the module has been developed, and
2. the module has successfully passed unit testing.

Three entities of interest:

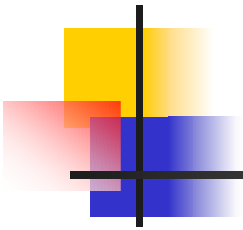
1. Module code
2. Unit tests for the module
3. Test execution and analysis results

Example EPM (3)

Example Time Line for Module Code Entity



- Entities remain for a non-zero time in each state.
- Transitions take negligible time.
- In the life span of an entity, it must always be in some state.



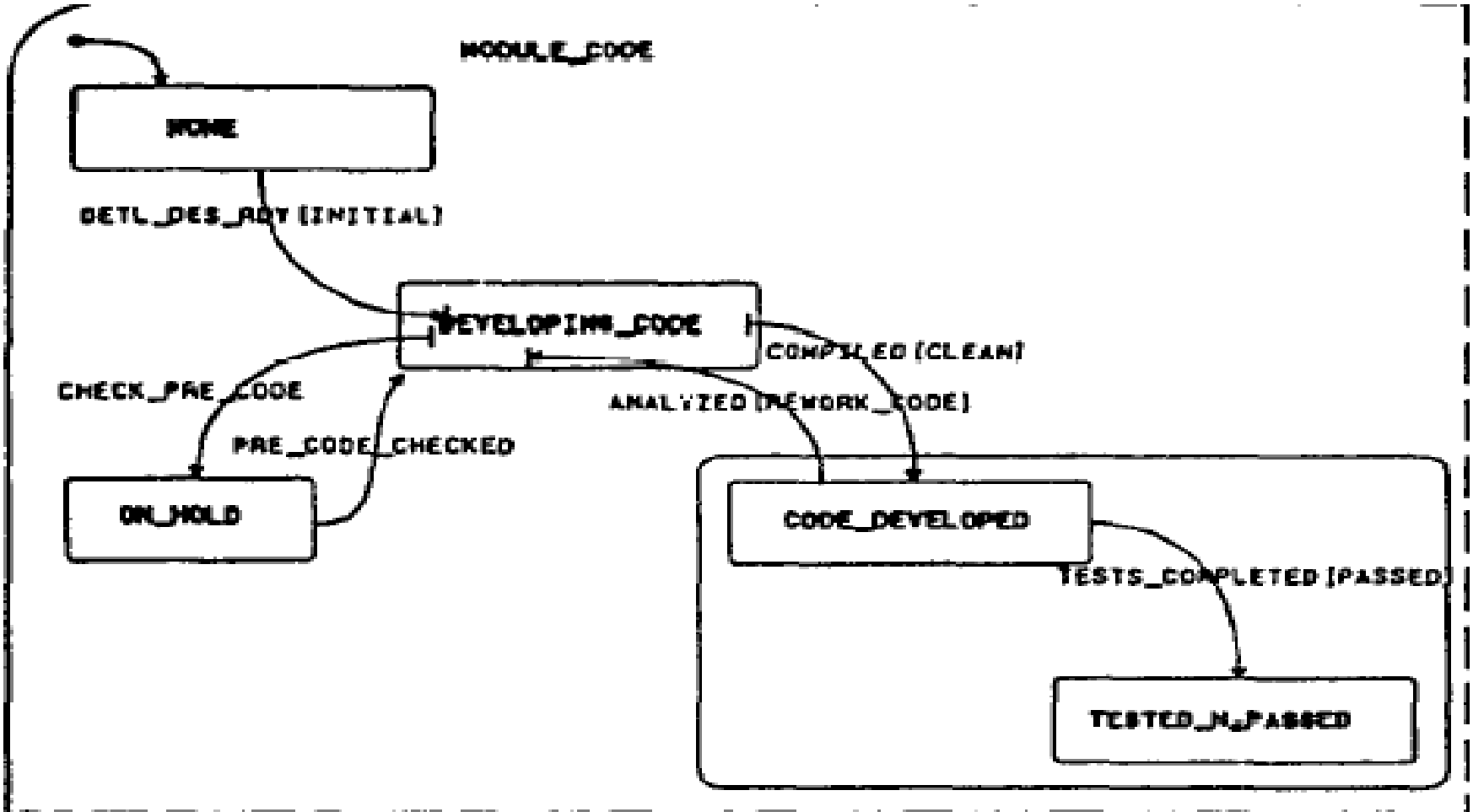
Example EPM (4)

- Statechart depicting an entity process modeling view of the example process.

- The boxes in the diagram represent states, while the lines represent transitions between states.

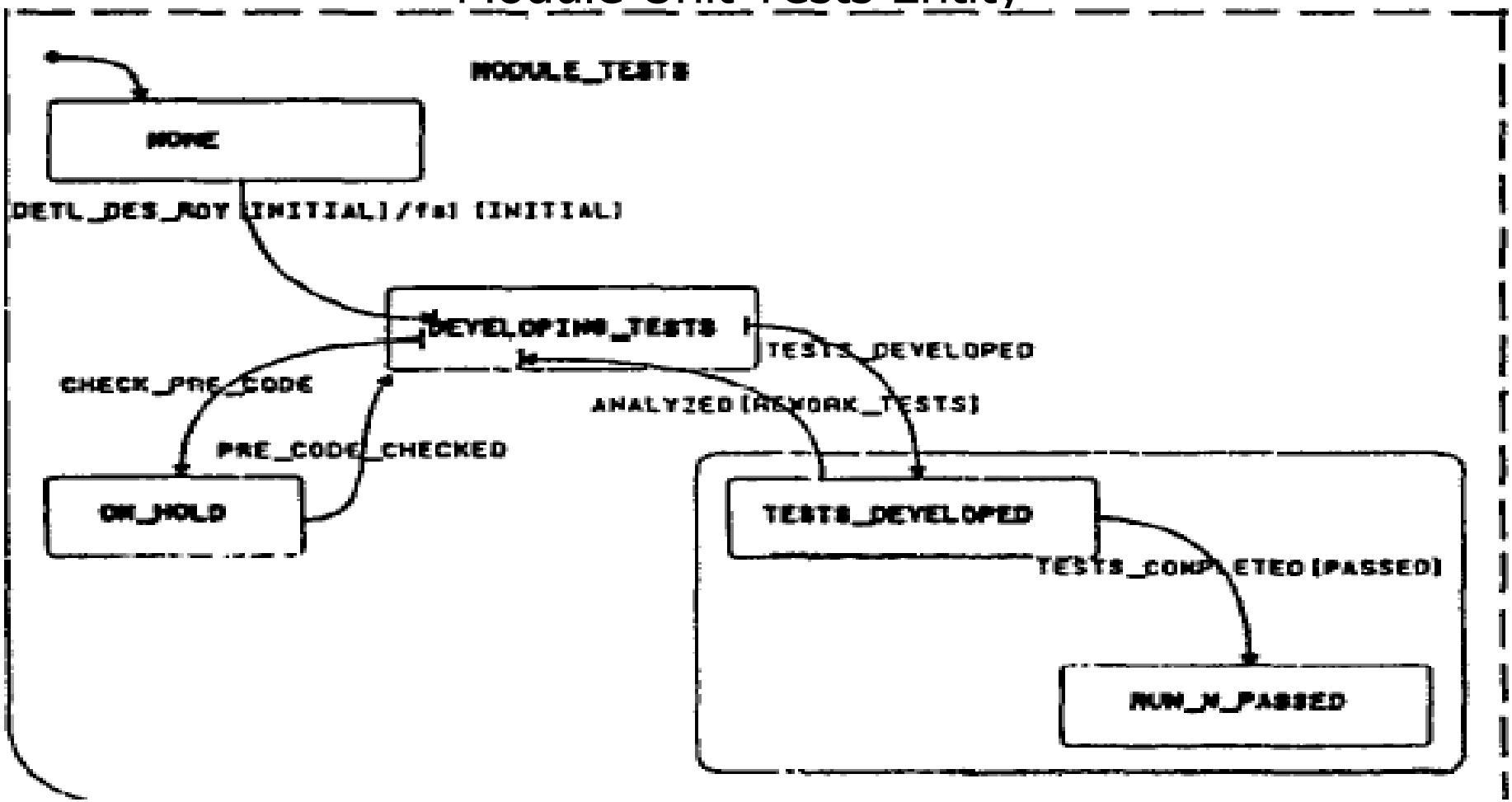
Example EPM (5)

Module Code Entity



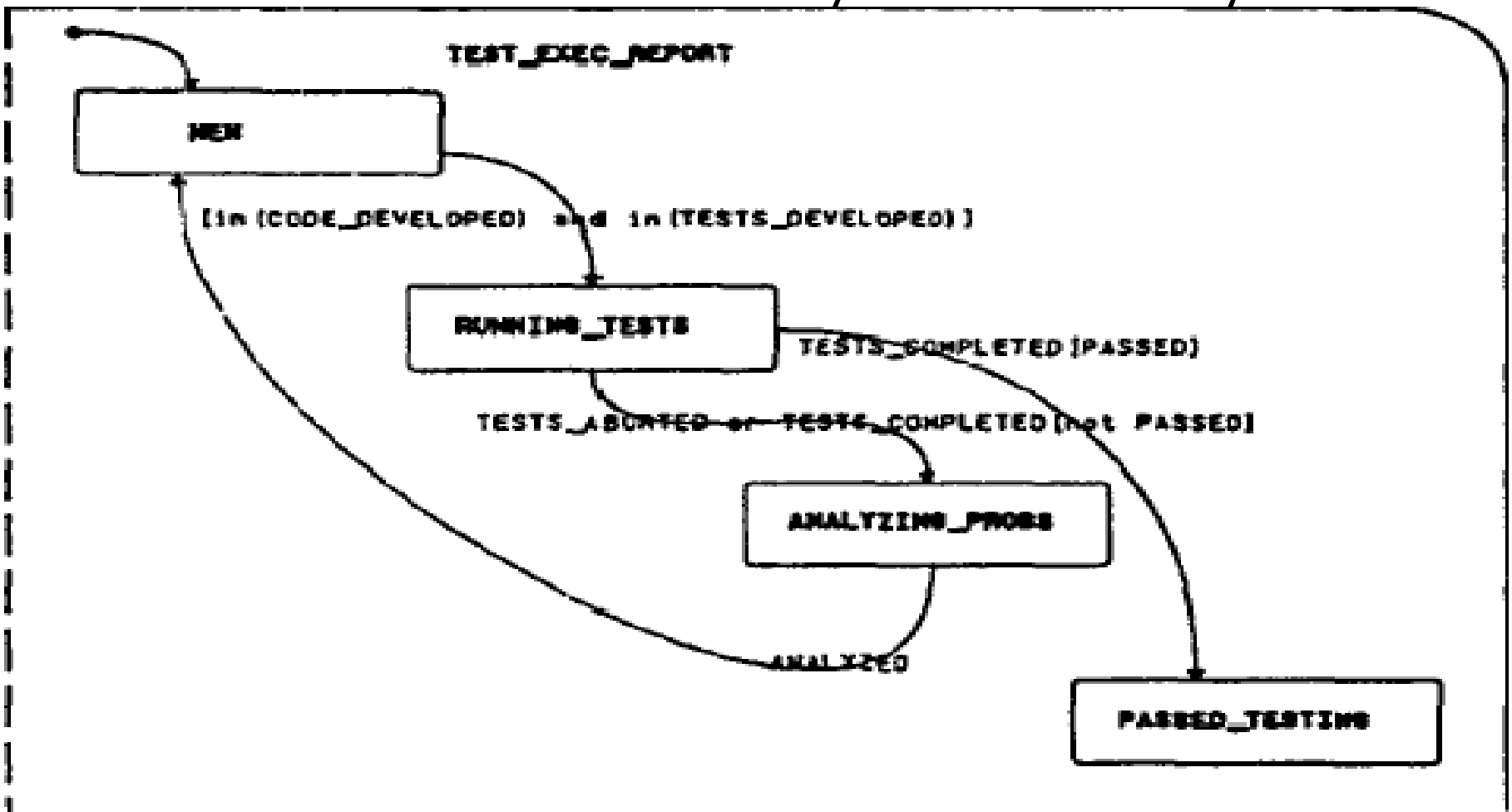
Example EPM (6)

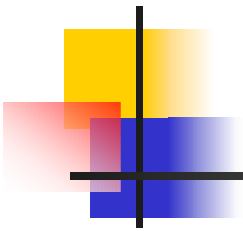
Module Unit Tests Entity



Example EPM (7)

Test Execution and Analysis Results Entity





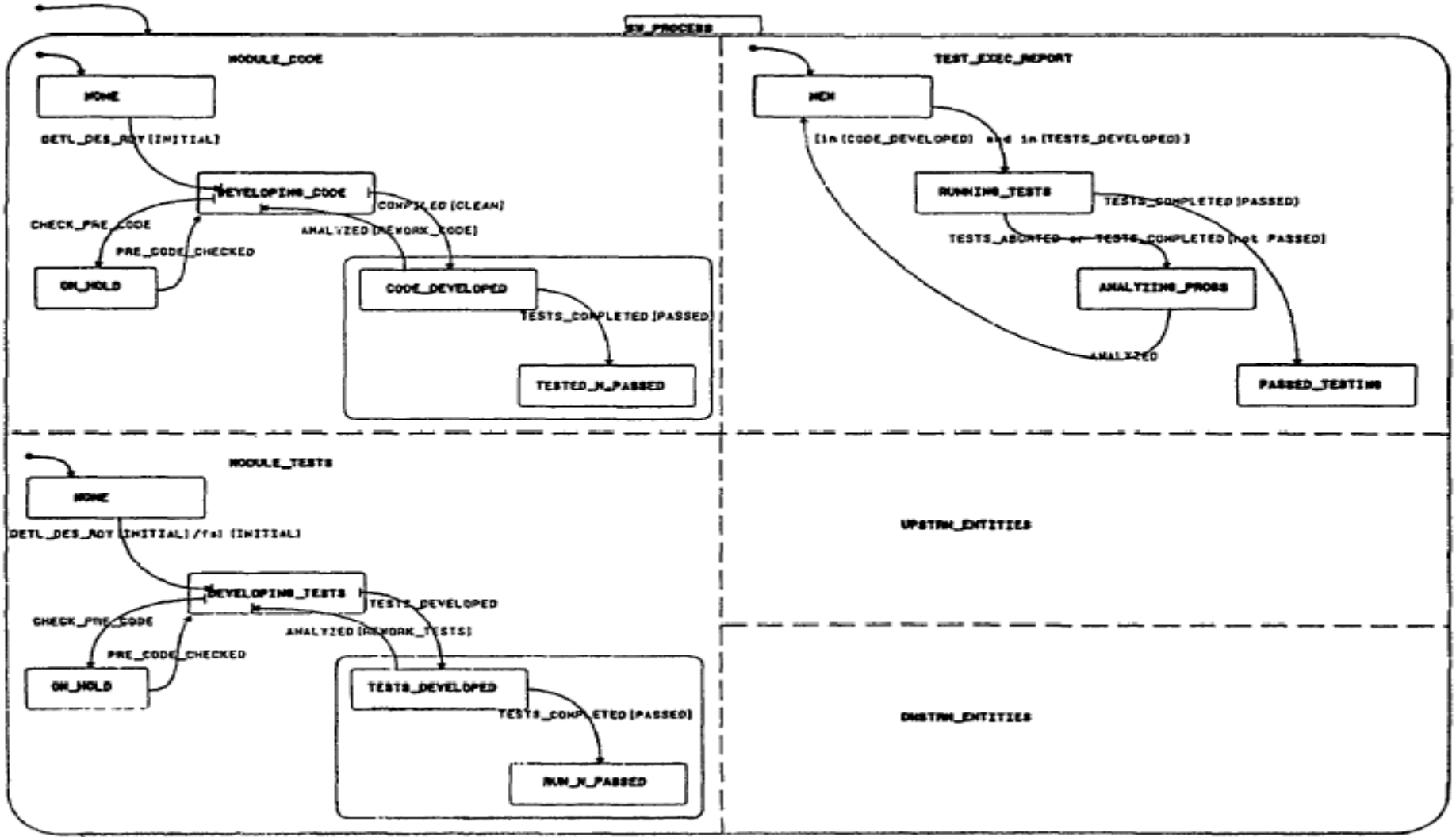
Example EPM (8)

- States can have orthogonal components, separated by dashed lines. These orthogonal components represent parallelism (concurrency); for example, the module code, tests, and test execution report all exist concurrently, as illustrated in the upper left quadrant (labelled module_code), lower left quadrant (labelled module_tests), and upper right quadrant (labelled test_exec_report) of the diagram, respectively.

Example EPM (9)

- Because STATEMATE does not offer an “entity” construct, the entities themselves have to be represented as high-level orthogonal state components, as shown by the major quadrants in the figure. At all lower levels, states in the statechart do indeed depict the various states of these entities.

Basic EPM Example (10)

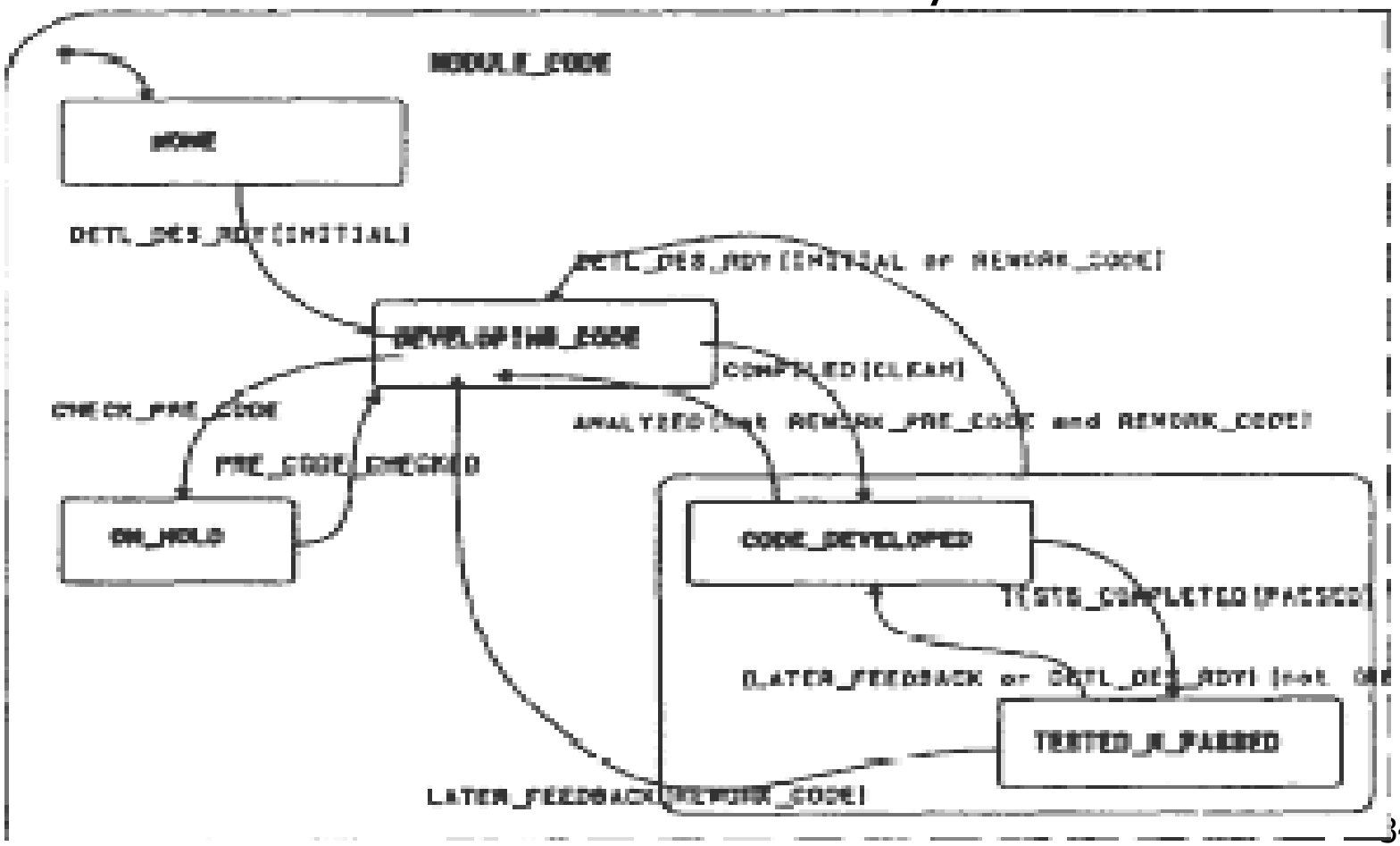


Example EPM (11)

- When the overall process is in a state (such as the large outer state labelled sw_process), it must also be in a substate in each orthogonal component.
- Components have been included in the lower right quadrant for upstream entities and downstream entities. These are simply placeholders to illustrate where the rest of the software process would be depicted, and would include states for the entities requirements, designs, system builds, integration tests, etc.

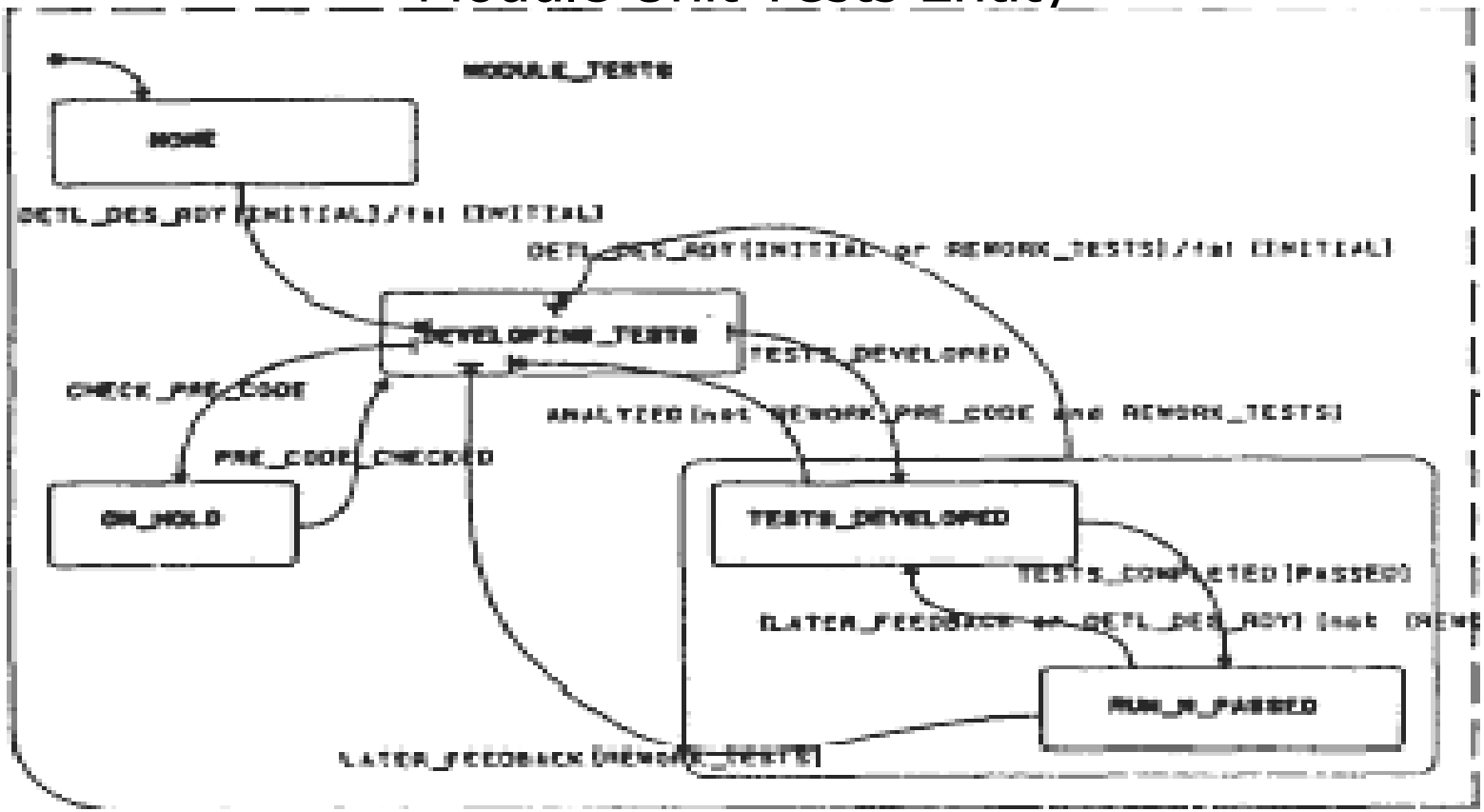
Example with Feedback (1)

Module Code Entity



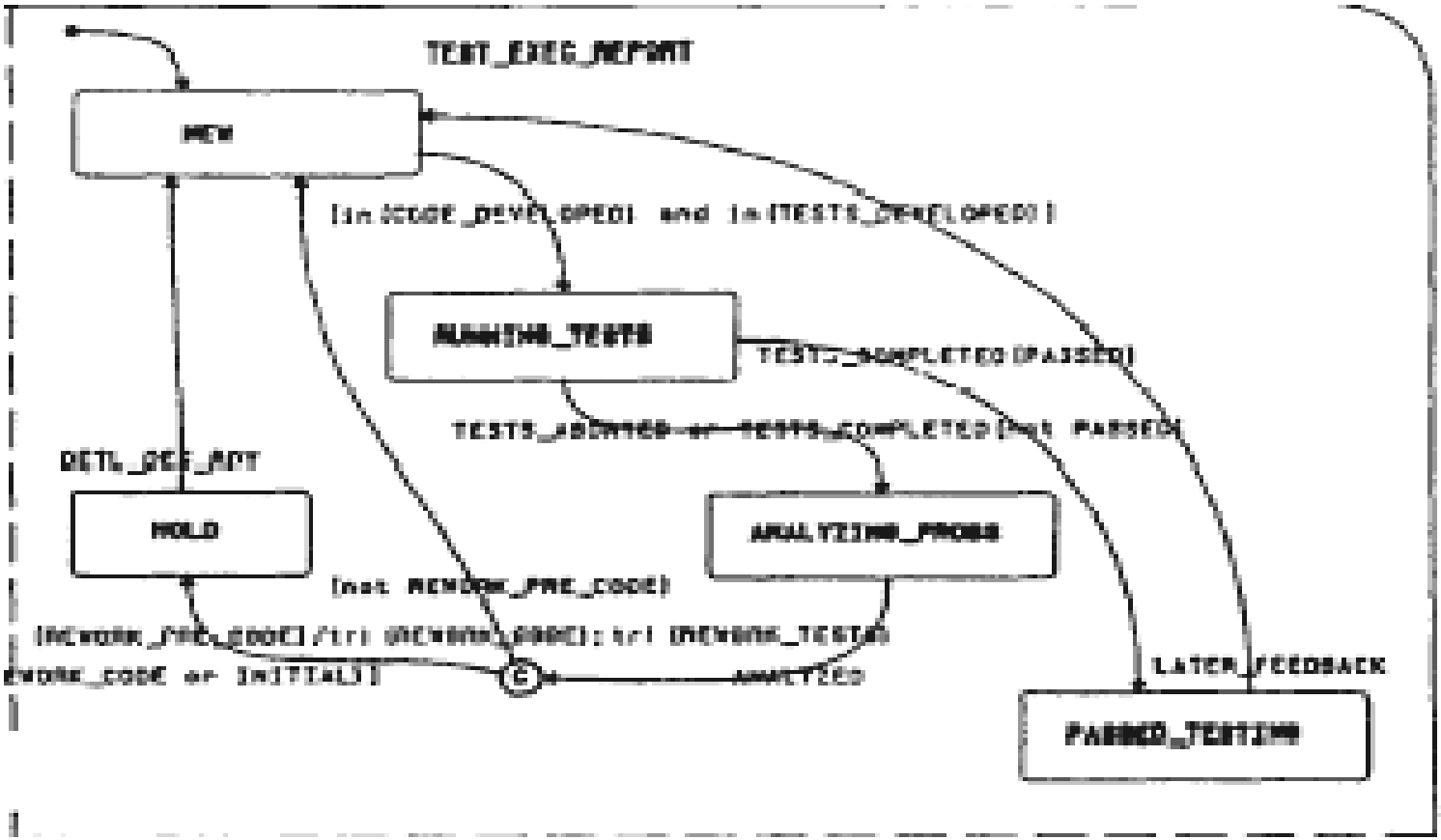
Example with Feedback (2)

Module Unit Tests Entity



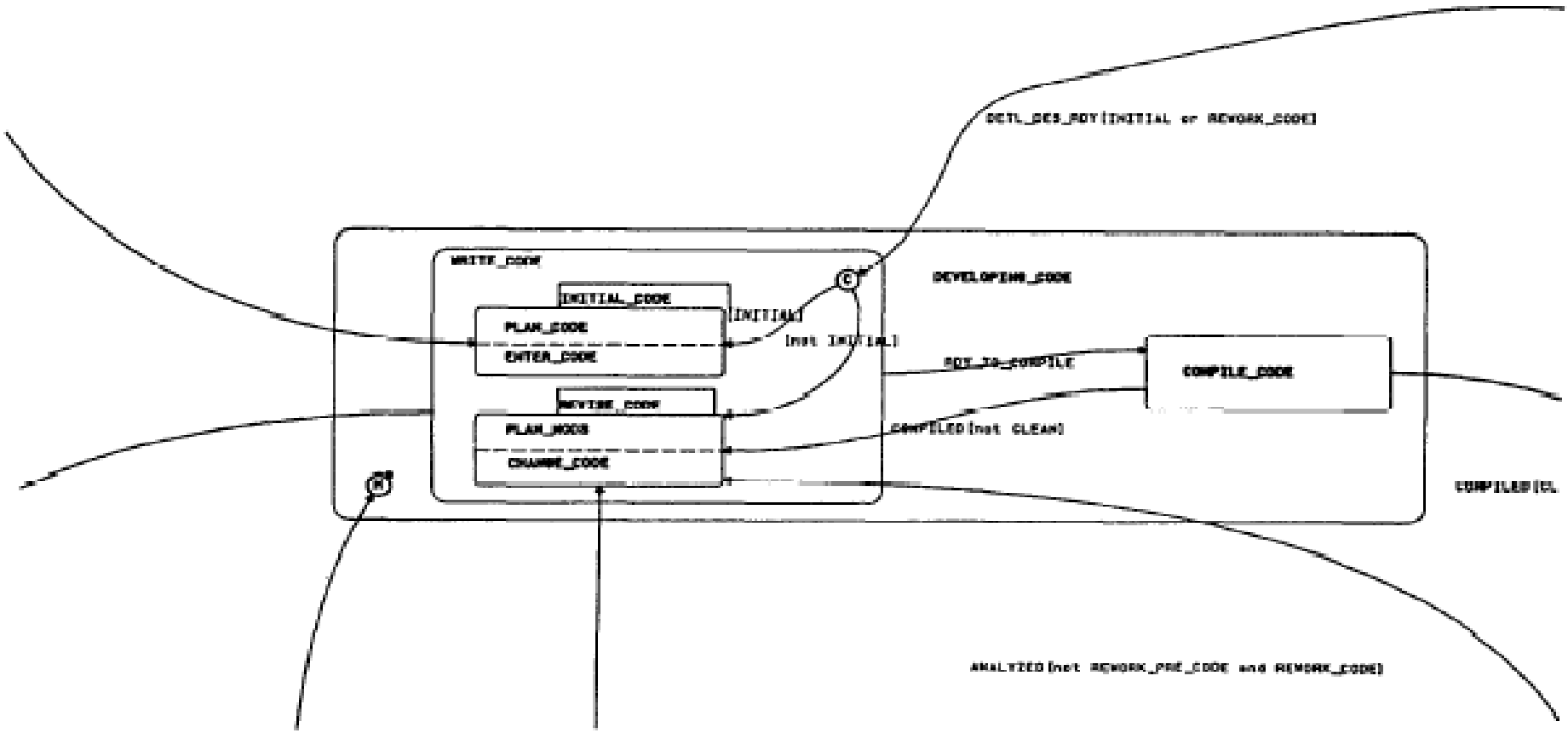
Example with Feedback (3)

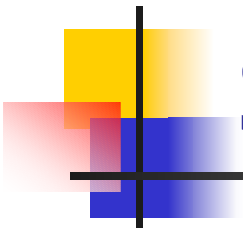
Test Execution and Analysis Results Entity



Further Refinement

EPM Example – Developing_Code Details





Scheduling Considerations

- Entity process models can be used for schedule planning and analysis.
- The example EPM developed is called unconstrained because it does not include any consideration of resource constraints in performing tasks and making transitions between states.
- The EPM can be used to derive an unconstrained process model (UPM), which is a schedule for the unconstrained case.

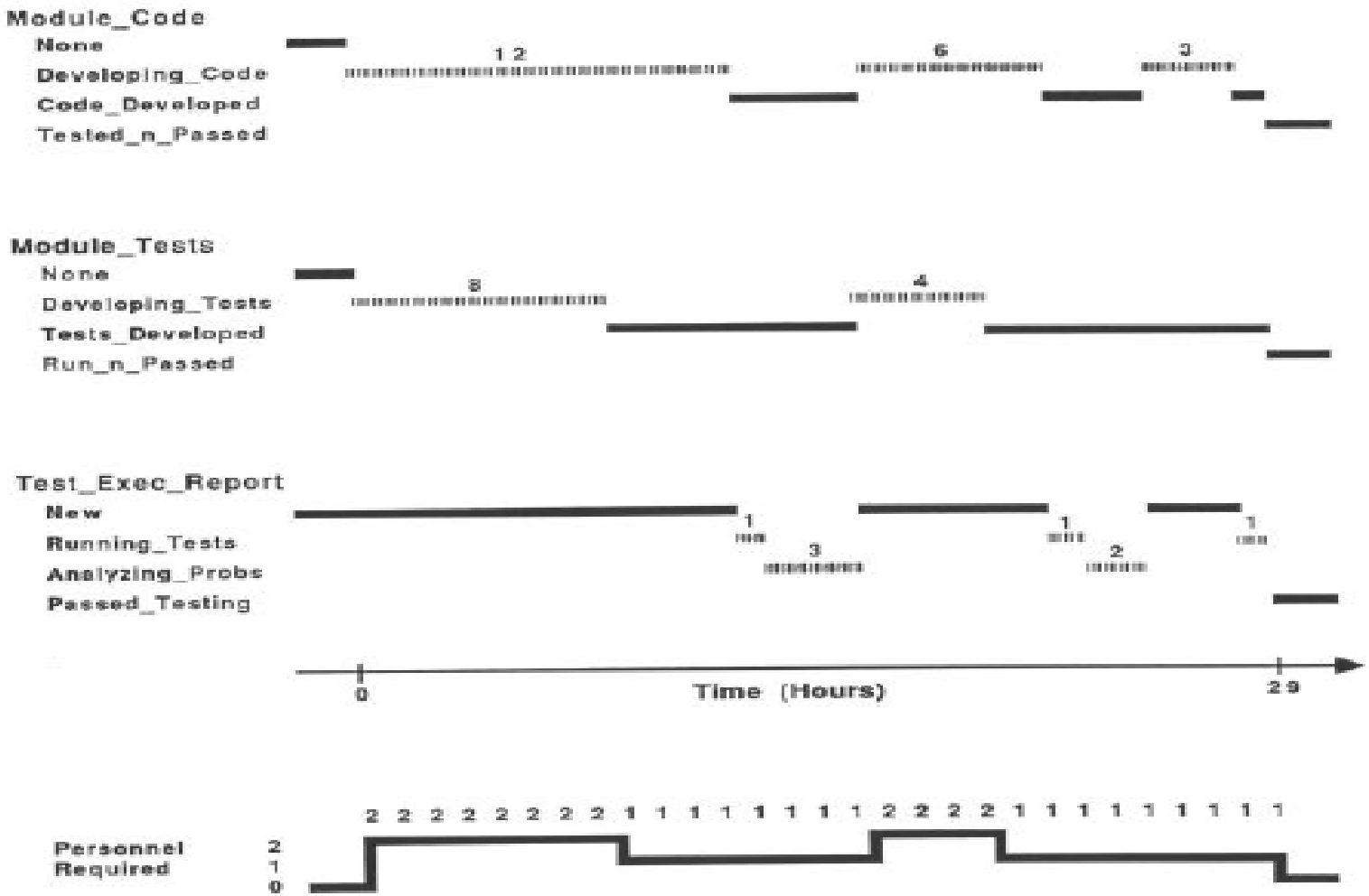
The Unconstrained Process Model (UPM)

- These tasks correspond to the active states in the statechart model.
- The basic plan forecasts that after initial development of code and tests, test execution will uncover errors calling for the rework of both code and tests at half their initial effort level.
- The second round of testing will uncover more errors, but only in the code, requiring one-quarter the initial effort to correct.
- The tests will then be passed on the third round. It has been assumed that each of these tasks is a one-person task that cannot be distributed among multiple workers.

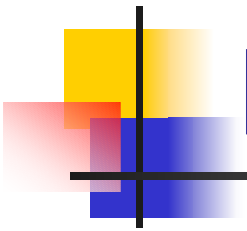
	Hours
Round 1	
Developing Code	12
Developing Tests	8
Running Tests	1
Analyzing Problems	3
Round 2	
Developing Code	6
Developing Tests	4
Running Tests	1
Analyzing Problems	2
Round 3	
Developing Code	3
Running Tests	1
-- Passed --	

Table 1: Resource Requirements for Examples

Example UPM


Module_Code
None
Developing_Code
Code_Developed
Tested_n_Passed
1 2
6
3
Module_Tests
None
Developing_Tests
Tests_Developed
Run_n_Passed
8
4
Test_Exec_Report
New
Running_Tests
Analyzing_Probs
Passed_Testing
1
3
1
2
1
0
Time (Hours)
29
2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1
Personnel Required
2
1
0

The Constrained Process Model (1)

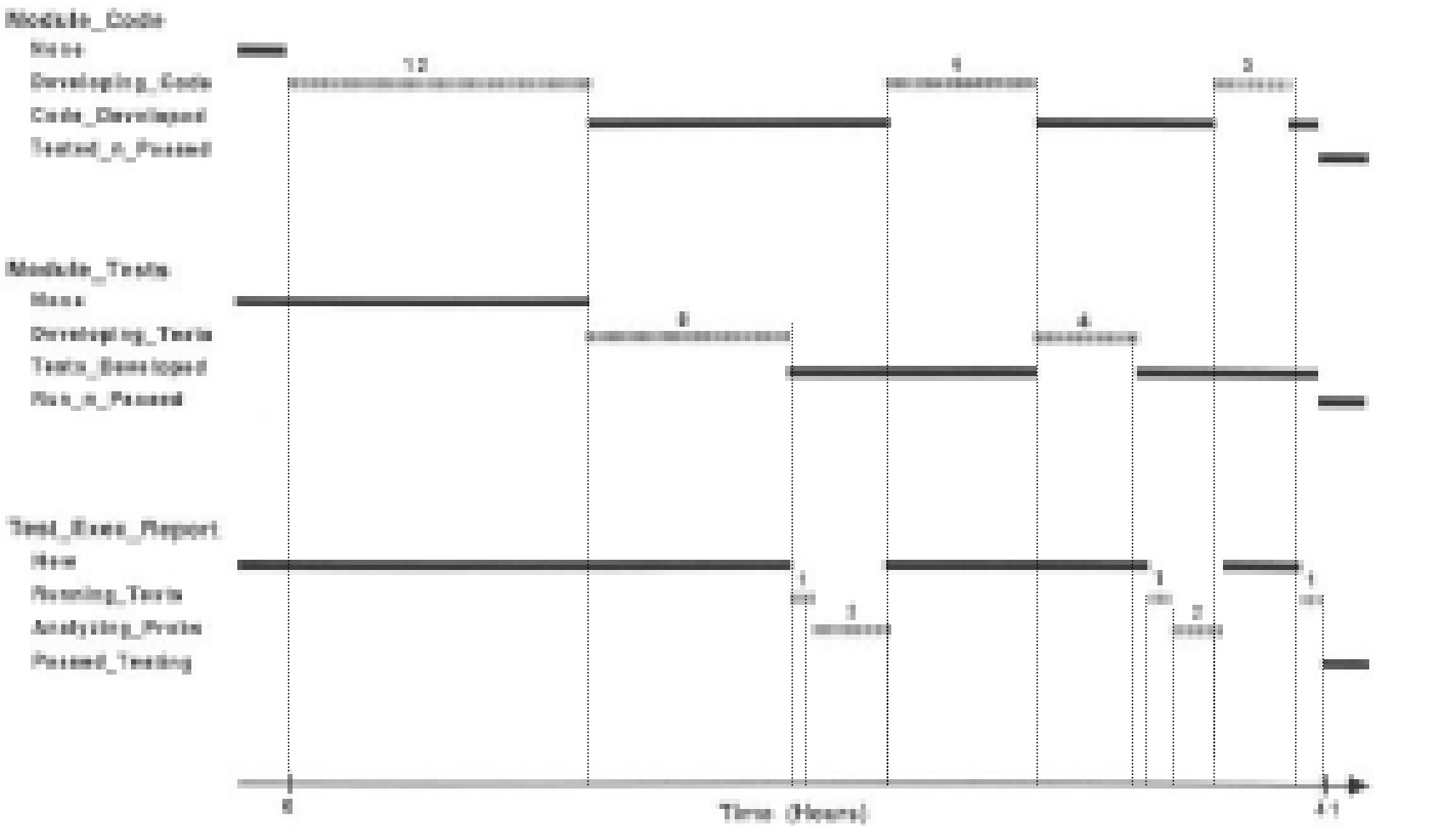


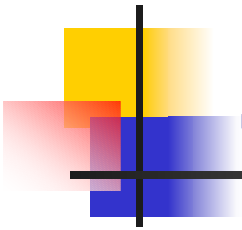
- Real software organizations have limited resources, some tasks may have to wait for personnel to become available to accomplish them.
- The CPM is produced by adjusting task timing to obtain the overall results desired, subject to the resource constraints.
- A typical objective would be to complete the process in the shortest time.

The Constrained Process Model (2)

- The UPM revealed that the process could be completed in 29 hours, but required two workers during 12 of those hours.
- What happens when only one worker is available?
- The shortest possible time with this resource constraint is 41 hours.

Example CPM



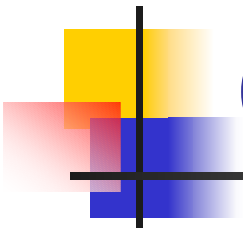


Schedule Management (1)

- Software Process Models can be valuable tools for schedule management.
- From the UPM (slide 41), it can be seen that there is slack time for initial test development: it can begin any time between time 0 and 4 without delaying completion of the process.
- On the other hand, initial code development is on the critical path; if any way could be found to speed up that task, overall completion would occur sooner.
- These charts also indicate that the addition of a second worker at certain points could speed up completion.

Schedule Management (2)

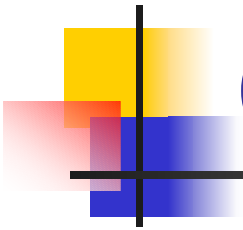
- The models are also useful during process execution.
- In the occurrence of a crisis delaying completion of a key task, the models allow management to determine if this task is on the critical path. If it is, then the completion schedule is threatened, and management can use the models to assess the available corrective actions.
- By examining the models, it becomes apparent whether added resources could help and where they should be applied to rebalance the schedule.



Conclusions (1)

The most attractive feature of EPMs is the new forces they generate:

- The prime entities of the software process are seen as persistent objects.
- A focus on states facilitates the tracking of 100% completed items rather than vague partial task completions.
- The UPM/CPM duality assists in adjusting for crises without bypassing essential elements of the process.



Conclusions (2)

- The use of the UPM/CPM pair simplifies initial scheduling and planning and permits simple adjustments to conform to new demands and available resources.
- The EPM models focus on the dynamic behaviour of a process and its impacts on the relevant entities.
- The EPMs, based on statecharts, are formal and enactable — in that we are able to run interactive, animated simulations of our EPMs with STATEMATE, as well as perform automated tests and analyses.

References (1)

1. Watts S. Humphrey , Marc I. Kellner, Software process modeling: principles of entity process models, Proceedings of the 11th international conference on Software engineering, p.331-342, May 1989, Pittsburgh, Pennsylvania, United States http://portal.acm.org/citation.cfm?doid=74587.74631
2. M. I. Kellner, Representation formalisms for software process modelling, Proceedings of the 4th international software process workshop on Representing and enacting the software process, p.93-96, April 1988, Devon, United Kingdom http://portal.acm.org/citation.cfm?doid=75111.75125

References (2)

3. "Hybrid Simulation Modelling of the Software Process"
   Paolo Donzelli, Giuseppe Iazeolla, Laboratory for Computer Science, University of Rome
   http://www.prosim.pdx.edu/prosim2000/paper/ProSimEA23.pdf

4. Software Process.
   http://www.cs.unc.edu/~stotts/COMP145/KMSnotes/seprocess.html

5. Software Process Models.
   http://www.cc.gatech.edu/computing/SW_Eng/people/Faculty/Colin.Potts/Courses/3302/1-08-mgt/sld001.htm