12TH INTERNATIONAL DESIGN STRUCTURE MATRIX CONFERENCE, DSM'10
22 – 23 JULY 2010, CAMBRIDGE, UK

THE IMPACT OF PACKAGING INTERDEPENDENT CHANGE REQUESTS ON PROJECT LEAD TIME

Naveed Ahmad, David C. Wynn and John Clarkson
Engineering Design Centre, University of Cambridge

Keywords: batching change requests, project lead time, DSM, DMM

1 INTRODUCTION

In a complex product development environment with many interdependencies between concurrent design activities, changes occurring during a project can have significant knock-on consequences in due to the rework required. It is thus important to effectively manage change work as it arises, in order to minimise its impact on the project schedule.

To help understand how change requests can be most effectively managed in such environments, we explore the impact of batching change requests for concurrent execution. A simulation model is developed which simulates the processing of changes in batches and allows estimation of the impact of the redesign work on the overall project duration. Our model extends state-of-the art by considering how the scope of change requests overlap, such that batching reduces not only co-ordination overheads but also the absolute amount of work that must be done.

We show how the model can be used to help managers determine a change request batching policy that will be suitable for a particular product development environment, and compare our findings to other simulations in this area.

2 BACKGROUND

Engineering changes are ubiquitous in design projects and unpredictable in nature (Rowell et al., 2009). During any given period of time a number of new change requests might be initiated (Loch and Terwiesch, 1999), resulting in a new batch of change requests which then have to be dealt with. At times, the unpredictable nature of change requests makes it unfeasible to execute them straightaway, for instance due to resource unavailability. As a result subsequent changes can accumulate during a product development project (Rowell et al., 2009; Giffin et al., 2009).

Subsequent changes may overlap, causing unnecessary rework termed here as '*re-rework*'. *Re-rework* in an activity is caused by a change request followed by a new change request such that the later one requires revisiting some of the same activities affected by the former change request. So, in order to understand the impact of changes on the project schedule it is important to consider the possibility of *re-rework*. Understanding re-rework is the focus of the present paper.

Many authors in the academic literature have studied issues that relate to this problem. For instance, Gartner et al. show how the effect of a single change request on project schedule can be estimated by identifying rework in the activities and consequently their impact on the project lead times (Gartner et al., 2008). Similarly, in our related article we have estimated the impact of changes to product requirements on the process, by identifying the rework required in the activities to execute that change (Ahmad et al., 2010). These approaches do not directly assist with understanding the impact of interdependent change requests on project schedules.

One way of dealing with the effects of successive changes and to minimise the effects of re-rework is to allow the changes to accumulate, then process them in a batch. Loch and Terwiesch discussed how the execution of changes in batches can reduce costs by allowing the setup costs to be shared. There are two main limitations of their analysis from the point of view of analysing re-rework: firstly, they do not consider changes in terms of rework in activities; and secondly, their method looks to batch similar changes (where setup costs are shared) and not interdependent changes (where, in addition, re-rework effects occur). Analyses of the impact of packaging change requests on project schedules can be found in software process dynamics literature as well. For example, mirroring the sharing of setup

costs in engineering tasks, Kilpinen discusses how packaging can save software testing time if similar changes are made together, as all testing will only have to be done once (Kilpinen, 2008). But, in literature we have found no specific methods which look at packaging interdependent change requests.

This paper explores the impact of packaging interdependent change requests at different time intervals, and estimates the impact of varying the time interval on project delivery dates. In particular, our method simulates the combined affect of all changes, accumulated in an interval (see Figure 1), executed at the same time on the project schedule. This can help the project manager in determining a change management policy by answering questions like: Should changes be managed collectively at regular intervals? And this will also help to eliminate the unnecessary rework that can be saved when packaging interdependent changes at the detailed design process level.

3 RESEARCH QUESTIONS

This paper aims to answer the following research questions.

1. What is the effect of changes occurring during development on the project lead time?
2. How is the rework added to activities affected if changes are processed in batches at intervals?
3. What should be the time interval between batch processing of changes?

These questions are explored through a simulation model discussed in the following section, and applied to a case study as described in Section 5.

4 SIMULATION MODEL

The model introduced in this section simulates the execution of changes occurring during the design at regular intervals to see the affect of varying intervals on the process lead time. Our simulation approach is based on the assumption that the change managers does not know the best interval to package change requests, so we try different size intervals to determine a change policy. Figure 1 and Figure 2 outline our method, which is described in greater detail below.

Simulating normal project execution in the absence of change

The simulation algorithm was developed to use a model of a process specified as an activity DSM, in which the duration of each task is specified. It is based on transitions between discrete process states, as explained below.

Representation of the process state: During simulation, each activity in the detailed design process has one of three possible states: pending, completed or executing:

1. *Pending*: All the activities that are not yet executed reside in the pending queue.
2. *Executing*: Once an activity starts it is moved from the pending queue to the executing queue.
3. *Completed*: The activities that are already completed reside in the completed queue.

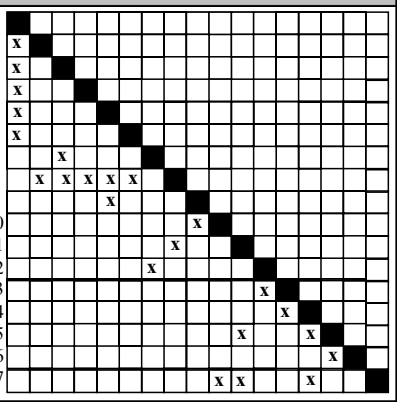
Algorithm: The 'normal execution' simulation algorithm (ie the project simulated in the absence of any changes) consists of the following steps:

1. *Initialization*: At the time '0' all the activities are placed in the pending queue. The activities in this queue are prioritized based on their respective start times. This will ensure the correct order of execution of the activities. The change review interval after which all the pending change requests are processed is also initialized to a value at the start of the simulation.
2. *Normal execution*: All the tasks in the execution queue and pending queue are updated as follows. The work done for each activity in the execution queue is updated as a discrete event and in case there is no further work left it is moved to the completed queue. The pending queue is then checked for the activities which are ready for execution due to completion of predecessors. All the activities whose start time matches the current time are moved to the executing queue.

Step 2 is repeated until all the tasks are executed. At change processing intervals, the merging of changes into the workflow is simulated using the algorithm described below.

#	Name	Duration
1	Acquire prefabricated items	2
2	Testing of other items	1
3	Testing of microcontroller (MC)	2
4	Testing of the LCD	2
5	Testing of the power supply	2
6	Testing of the clock	1
7	Writing software	11
8	Acquire PCB	3
9	Replacing wires in power supply	2
10	Final testing of power supply	3
11	Acquire casing	4
12	Testing the software	4
13	Loading software in MC	1
14	Final test of MC	4
15	Assemble devices on PCB	2
16	Final testing of the circuit	4
17	Input and output sockets	1

Process DSM



Normal execution					
1	2	3	4	5	6
○	●	●	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	●	●	●	●
-	-	-	○	○	○
-	-	-	○	○	○
-	-	-	○	○	●
-	-	-	-	-	○
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-

Figure 1. Simulation during normal execution (where activity status is pending (-), executing (○), completed (●))

Simulating the impact of batch processing of change(s) on normal project execution

The steps taken to initiate, evaluate and simulate changes in the project execution are:

Initiation of change requests *('1' in Figure 2)*: As discussed earlier change requests are assumed to arrive randomly during a product development project and accumulate over time. To emulate the initiation of change requests in our simulation model the distribution shown in Figure 2(1) is used.

Evaluation of change requests (*'2' in Figure 2*): Each change is evaluated in three steps: 1) identify the components affected; 2) identify the activities requiring rework as a result of change in components; and 3) identify the indirectly affected activities.

1. *Identify the affected components*: The product is modelled as a DSM of components where each connection between two components had a likelihood value and an impact value which gives the probability of change propagation from a component to all other components, and the impact of that change respectively. This product DSM is used as input to the CPM method to identify the risk of change propagation to other components of the product (Clarkson et. al, 2004). The output of the CPM method is a risk matrix which gives the risk of change propagation from one component to the all the other components. The components with the high value of risk are selected as ones that may be affected by a particular change.
2. *Identify the directly affected activities*: The mapping of product components to detailed design process activities is used to identify the list of activities that will be affected by change in the components. Gartner et al. also used a product-to-process DMM to identify the rework in activities (Gartner et al., 2008). This product-to-process DMM can identify the direct rework in the design activities as a result of changes in the components. Rework in each activity is decided based on its current state of completion as explained further in the following sections.
3. *Identify the indirectly affected activities:* In case of a design process if there is a change in an activity then all the downstream activities are prone to change. All the directly affected tasks are identified using a component-activity DMM in Step-1. In step-2 all the knock-on rework in the detailed design process is identified using the activity DSM. When dealing with the activities in the detailed design process, work always flows from an activity to its downstream activities, apart from iterations. It is important to supply the activities in their original order of execution so that the activities that are executed first in the original execution of design process will be reworked first, as this will ensure that no unnecessary rework occurs.
   Our simulation model also follows this assumption, so the activities identified in step 1 along with all the downstream activities are labelled as 'rework activities'. Figure 2(2) illustrates how for each change case a list of activities requiring rework is generated.

Simulate the 'change in project execution status' following a change processing interval *('3' in Figure 2)*: The interval to process pending change requests and a buffer containing all the change requests are input to our simulation method at the change processing interval. At each time interval all the change requests in the buffer are executed at once and the project schedule is updated accordingly.

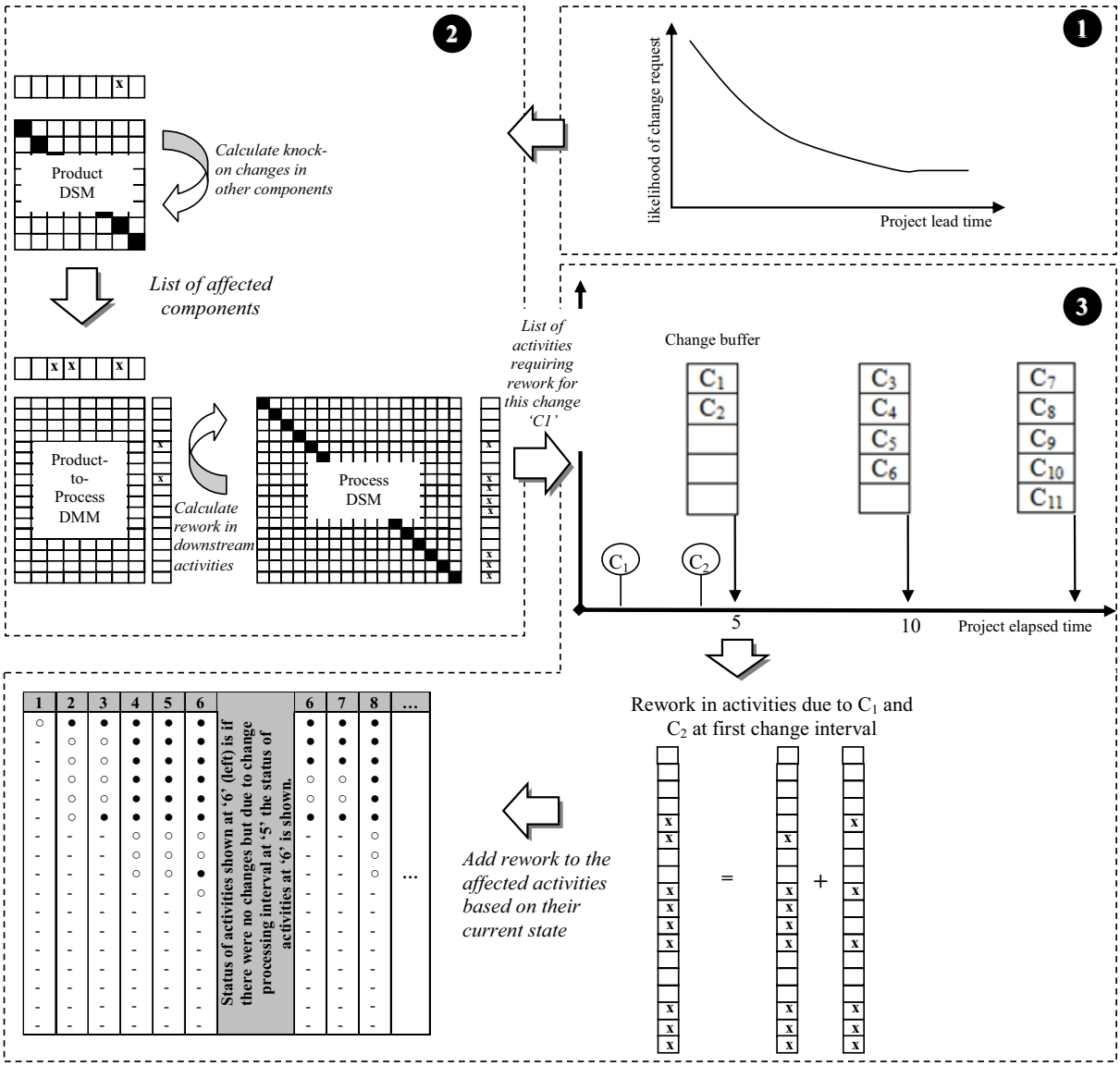


Figure 2. Simulation model (showing steps taken when processing changes at each change interval)

Execution at the change processing interval: We have assumed that processing interdependent changes in intervals can save extra rework, so the interval for processing batch of changes can be fixed at the start of the simulation. All the changes that are queued up to this point are processed after each of these fixed intervals. All the activities requiring rework from the current batch of changes, as identified in Step-2, are checked and the following steps are taken based on their current status.

1. If an activity is in the pending queue then no rework is required, because it is already waiting for execution.
2. In case an activity is currently in execution, the rework is decided based on the current state of the activity – specified as a function of the total time required by the activity to complete and of the work left in the activity. In general, an activity's rework behaviour can be defined by a case-specific curve as shown in Figure 3 (left) (Carrascosa et al., 1998). In Carrascosa's model each activity has its own curve. For simplicity we have used the function on Figure 3 (right) for every activity in our model. It gives an approximate measure of how much rework is required based on the work left in the activity, so if an activity has more work left then the additional time required will be less and vice versa. For instance, if an activity has 90% work left then the additional rework added to it will be 10%.

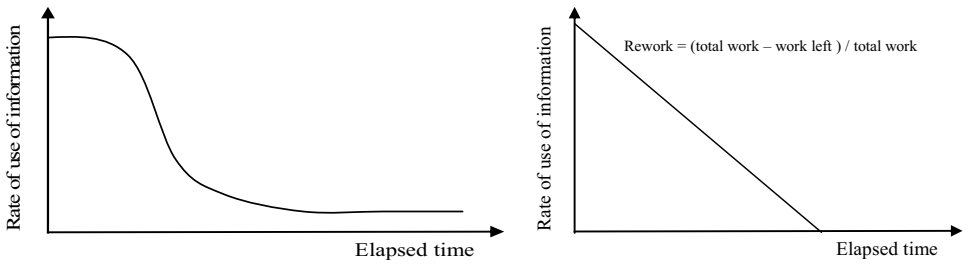


Figure 3: 'Rate of information use' by an activity as a fraction of its elapsed time

3. In case change requires rework of a completed activity, all the activities currently in execution are interrupted and moved to the pending queue. The completed activity requiring rework is moved again to the execution queue and it is assumed that whole activity will require rework. This assumption is also supported by Krishnan et al. in their simulation (Krishnan et al., 1997). In case the activities interrupted are not one of the other activities affected, then they are also moved back to the execution queue. All the interrupted activities when added to the queue require additional times due to switching queues termed as 'penalty'.

Results: At each change processing interval some amount of rework is added to the affected activities based on their current state. The affect of this rework on the overall project schedule is calculated and the schedule is adjusted accordingly to reflect the delay on total project completion time. Simulation then proceeds according to the algorithm explained at the start of Section 4, until the next change processing interval.

5 EXPERIMENT

The simulation model was applied to a microcontroller based device AUTOBELL, a product of Digital Research Labs (DRL). The first author conducted a case study to construct the model consisting of the product DSM, the activity DSM and a product-to-process DMM where each component is connected to at least one activity in the detailed design process.

13,000 simulations were run using the AUTOBELL model. For each change processing interval the simulation was executed 1000 times. During each simulation 20 different change requests are initiated at random times. In our simulation model the occurrence of changes is random, depicting any real life product development scenario where a change can occur at any time. As described above, all the changes occurring during an interval are stored in a buffer until the specified time is elapsed, then all the batched changes are executed at once and their affect is translated on the project schedule.

The resulting process durations, shown in Figure 4, give the average delay in project completion from the baseline case where no changes are initiated. Figure 4 shows the need to appropriately choose the time interval between processing batches of change requests. On both the extreme left and extreme right ends of the plot, the delay in project completion increases - showing that it is inefficient to process changes as soon as they arrive, but also inefficient to wait too long for batch processing of change requests.

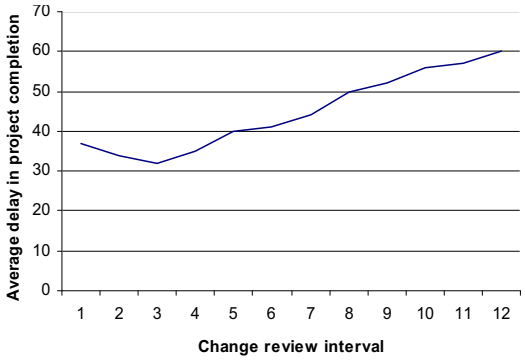


Figure 4: Plot of average delay in project lead time for different intervals

6 DISCUSSION AND CONCLUSION

The simulation approach discussed in this paper explored the affect of batch processing of changes at different change processing intervals on the project lead time. The results of the experiment highlight the importance of choosing an appropriate change processing interval.

The following conclusions can be drawn from the results of the experiment:

1. Changes occurring during the development cause delay in the project lead time, and change management policies can be chosen to mitigate their impact.
2. The result of the simulations shows that if the appropriate interval is chosen then unnecessary rework in activities can be avoided. This is because when changes are held and processed together, this saves revisiting the same activities several times (i.e. reduces re-rework).
3. The appropriate interval depends on the structure of a particular project. In case of the project modelled in this paper, the appropriate change processing interval is 3 days as shown in Figure 4 – but this will vary for a project with different activity duration and dependency structure.

In future, we plan to further develop the simulation model presented here to incorporate other factors influencing the execution of changes – perhaps most importantly, the availability of resources to execute change requests. Moreover, we suggest that trade-offs between factors like time and cost could help to get a better estimation of the change processing interval. The simulations also indicate the number of times an activity was reworked and the amount of rework in different activities. This data can be further analysed to adjust the product architecture in order to avoid this repeated rework.

REFERENCES

Ahmad, N., Wynn, D.C. and Clarkson, P.J. (2010). Development and Evaluation of a Tool to Estimate the Impact of Design Change, *International Design Conference – Design 2010*, Dubrovnik, Croatia, May 2010.

Carrascosa, M., Eppinger, S.D., and Whitney, D.E. (1998). Using the Design Structure Matrix to Estimate Product Development Time, In *Proceedings of DETC'98 1998 ASME Design Engineering Technical Conferences*, Atlanta, Georgia, USA, September 1998.

Giffin M., de Weck O., Bounova G., Keller R., Eckert C., and Clarkson P.J. (2009). Change Propagation Analysis in Complex Technical Systems, *Journal of Mechanical Design,* 131(8), 081001.

Gärtner, T., Rohleder, M., and Schlick, C.M. (2008). Simulation of Product Change Effects on the Duration of Development Processes based on the DSM, In *10th International DSM Conference*, Stockholm, Sweden, November 2008, pp. 199-201.

Eckert, C.M., Clarkson, P.J., and Zanker, W. (2001). Aspects of a Better Understanding of Changes, In *International Conference on Engineering Design, ICED '01, Vol. 1,* Glasgow, August 2001. Bury St Edmunds: Professional Engineering Publishing.

Kilpinen, M.S. (2008). The Emergence of Change at the Systems Engineering and Software Design Interface: An Investigation of Impact Analysis, PhD Thesis, University of Cambridge, United Kingdom.

Krishnan, V., Eppinger, S.D., and Whitney, D.E. (1997). A Model-Based Framework to Overlap Product Development Activities, *Management Science*, 43(4), 437-451.

Loch, C. H. & Terwiesch C. (1999). Accelerating the Process of Engineering Change Orders: Capacity and Congestion Effects, *Journal of Production and Innovation Management*, 16, 145-159.

Rowell, W.F., Duffy, A.H.B, Boyle, I.M., and Masson, N. (2009). The Nature of Engineering Change in a Complex Product Development Cycle, In *7th Annual Conference on Systems Engineering Research 2009 (CSER 2009)*, Loughborough, UK, April 2009.

Wynn, D.C., Caldwell, N.H.M., and Clarkson, P.J. (2010). Can Change Prediction Help Prioritise Redesign Work in Future Engineering Systems? *International Design Conference – Design 2010*, Dubrovnik, Croatia, May 2010.

Contact: Naveed Ahmad
Engineering Design Center (EDC), Department of Engineering
Trumpington Street, CB2 1PZ, Cambridge, United Kingdom
e-mail: na315@cam.ac.uk

The Impact of Packaging Interdependent Change Requests on Project Lead Time

Naveed Ahmad
David C. Wynn
John P. Clarkson

University of Cambridge

Technische Universität München

UNIVERSITY OF CAMBRIDGE

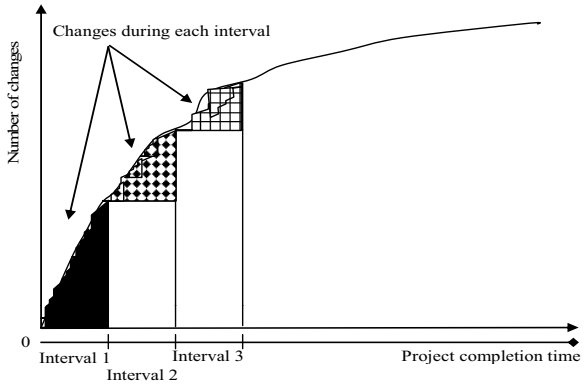
MANAGING COMPLEXITY
BY MODELLING DEPENDENCIES

Index

- Problem
- Solution
- Research questions
- Simulation approach
  - Variables
  - Simulation during normal execution
  - Simulation during change processing interval
- Results
- Discussion
- Future work
- Conclusion

Problem

- Complex product development environment with many interdependencies between concurrent activities
- Consequences of changes in terms of knock-on rework
- Change accumulation



Solution

Packaging change requests…

Research Questions

- What is the affect of changes occurring during development on the project lead time?
- How is rework added to the activities affected if changes are processed in batches at intervals?
- What should be the length of interval for batch processing of changes?

Simulation Variables

Constants that specify product/process system being simulated	
change_interval	Time between change processing actions.
Product/process model	Dependencies between subsystems in the product; information flows in the process; dependencies between activities and subsystems they consider/affect; sequence of activities.
time_step	Timestep for the simulation.
Variables associated with process, that change during simulation	
current_time	At the start of simulation current time is initialized to '0'. At each step it is incremented by *time_step*.
Pending queue	All the activities that are not yet executed always reside in the pending queue. This is sorted in order of increasing start time.
Executing queue	Once an activity starts it is moved from the pending queue to the executing queue. This is sorted in order of increasing end time.
Completed queue	The activities that are already completed reside in the completed queue.
change_time	Next time at which changes will be processed.
change_buffer	All the change requests are stored here until change_time is reached.
Variables associated with every activity in the process, whose values change during simulation	
start_time	Start time of the activity is specified during initialization.
total_work	Total work is amount of the work to be done by in an activity.
work_remaining	Work remaining is the amount of work that is to be done in an activity. This is initialized to *total_work*.

Simulation during normal execution

- The algorithm operates as follows (detailed in Listing 1):
  - **Initialization**
  - Update all activities in the execution queue
  - Check the pending queue if an activity is read to start
  - Check if its time to for change processing interval
  - Run this algorithm until all the activities are finished

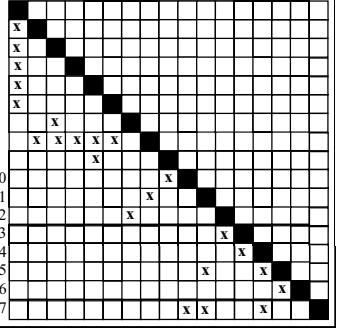
```
1. Set current_time='0'; change_time = change_ interval;
     Place all activities in pending queue.
2. FOR-EACH activity i in executing queue:
     a.  work_remaining[i] = work_remaining[i] – timestep
     b.  IF work_remaining[i]<=0
             remove activity i from executing queue and place
                 in completed queue.
3. FOR-EACH activity i in pending queue:
     a.   IF start_time[i] <= current_time
             remove activity i from pending queue and place it
                 in execution queue.
4. IF current_time >= change_ time
     a.  EXECUTE CHANGES [described  listing 2]
     b.   change_time = change_time + change_interval
5. IF there are activities in executing queue or pending queue.
     a.  current_ time = current_time+ timestep
     b.  GOTO 2.
```


Simulation during normal execution (cont…)

- Normal execution… *where activity status is pending (-), executing (○), completed (●)*

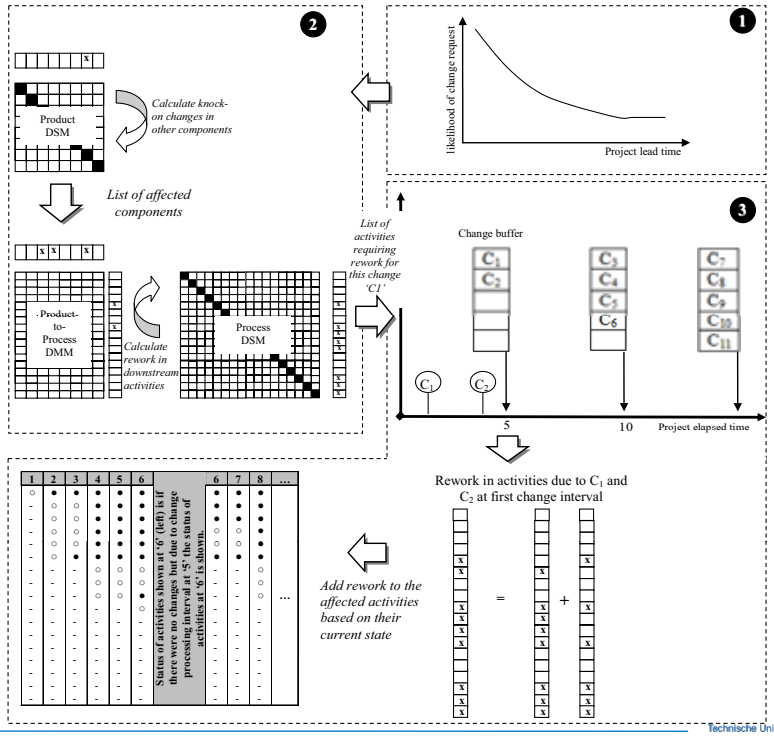
#	Name	Duration
1	Acquire prefabricated items	2
2	Testing of other items	1
3	Testing of microcontroller (MC)	2
4	Testing of the LCD	2
5	Testing of the power supply	2
6	Testing of the clock	1
7	Writing software	11
8	Acquire PCB	3
9	Replacing wires in power supply	2
10	Final testing of power supply	3
11	Acquire casing	4
12	Testing the software	4
13	Loading software in MC	1
14	Final test of MC	4
15	Assemble devices on PCB	2
16	Final testing of the circuit	4
17	Input and output sockets	1

Process DSM



Normal execution					
1	2	3	4	5	6
○	●	●	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	○	●	●	●
-	○	●	●	●	●
-	-	-	○	○	○
-	-	-	○	○	○
-	-	-	○	○	●
-	-	-	-	-	○
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-

Simulation during change processing interval



Simulation during change processing interval (cont…)

- Change initiation
  - In our simulation model the occurrence of changes is random, depicting any real life product development scenario where a change can occur at any time.

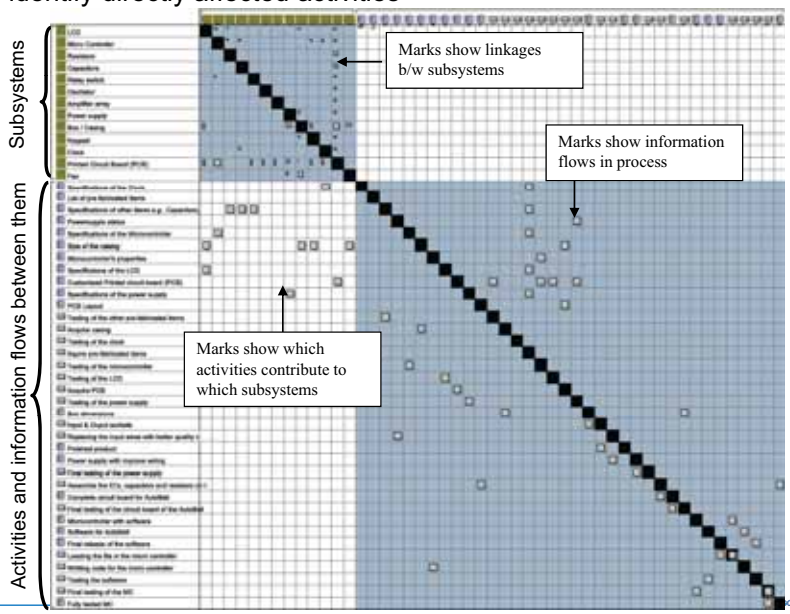
$$\Pr obability\left(Change\ \mathrm{Re}\ quest\right) = \frac{T - CT}{T}$$

where T - project completion time; CT – current time

- Evaluation of change requests – Step 1
  - ***Identify the affected components: It*** includes using the Change Prediction Method to identify change propagation in other components of the product

Simulation during change processing interval (cont…)

- Evaluation of change requests – Step 2
  - **Identify the directly affected activities:** Product-to-process DMM used to identify directly affected activities



Simulation during change processing interval (cont…)

- Evaluation of change requests – Step 3
  - **Identify the indirectly affected activities:** All the successors of the activities identified in step 2.
- Execution at change processing interval
  - If an activity is in the pending queue then no rework is required, because it is already waiting for execution
  - In case an activity is currently in execution than the rework is decided based on the current state of the activity and it is a function of total time required by the activity to complete and work left in the activity

$$\text{Re}\,work = \frac{TW - WR}{TW}$$

where TW – total_work; WR – work_remaining

  - In case change requires rework of a completed activity then all the activities currently in execution are interrupted and moved to the pending queue. The completed activity requiring rework is moved again to the execution queue and it is assumed that whole activity will require rework

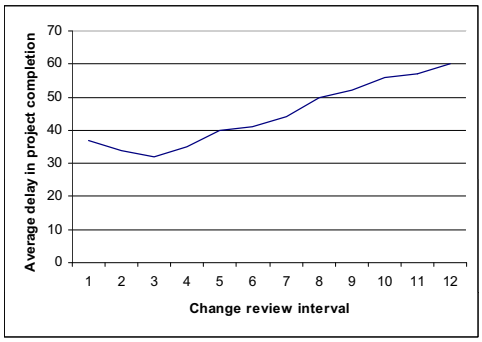
Simulation during change processing interval (cont…)

- Execution at change processing interval algorithm

```
1. Set change_time = change_time + change_ interval;
2. FOR-EACH activity i in the change_buffer:
    a. IF activity i is in executing queue
            rework = total_work – work_remaining /
                total_work;
            work_remaining[i] = work_remaining[i] +
                rework;
            FOR-EACH activity j successor of i:
                start_time[j] = start_time[j] + rework;
    b. ELSE IF activity i is in completed queue
            work_remaining[i] = total_work[i];
            FOR-EACH activity j in executing queue
                remove activity j from executing queue and
                        place in pending queue.
                work_remaining[j] =work_remaining[j] +
                        penality;
                IF activity[j] is not a successor of activity [i]
                        Remove it from pending and place it
                        back in executing queue
                FOR-EACH activity k in pending queue
                        IF k is a successor of j
                                start_time[k] = start_time[k]
                                + work_remaining[j];
    c. ELSE //means activity i is still pending
            GOTO 2.
```


Simulation experiment

- 13000 simulations for different size change intervals
- For each change processing interval the simulation is run 1000 times
- During each simulation 20 different change requests are initiated at random times
- The plot below shows a trade-off when choosing the size of the interval



Conclusion

- Changes occurring during the development cause delay in the project lead time and change management policies can be made to mitigate their impact.
- The result of the simulations show that if the appropriate interval is chosen then unnecessary rework in activities can be avoided because changes are held and processed together so saving multiple changes possibly in same activities.
- Processing multiple changes request in batches can be beneficial but choosing the appropriate interval depends on a particular project schedule. In case of the project schedule used in this paper the appropriate change processing interval is 3 days but this will vary for a project with different activity duration and overall lead time.

Future work

- The simulation model will be further developed:
  - To incorporate other factors influencing the execution of changes such as availability of resources
  - To include trade-offs between time / cost to get a better estimate of change processing interval.
- To analyse the data from simulations such as; which activities are changed more often or amount of rework in activities