22. Iteration [2]

12 November 2008

Bob Glushko

Plan for ISSD Lecture #22

- More Big Ideas and Issues about Iteration
- Iteration in UI Design
- Iteration in Software Development
- Iteration as "Information Exchange" and the "Design Structure Matrix"

Designing Iteration

What is the project scope to which iterative methods will be applied?

What are the processes in each iteration?

What is the "cycle time" of the iteration?

The "metamodel" for feedback

The Iteration Cycle Time

What determines how long the cycle takes?

What is being built on each cycle?

How is it being built? Is it being built "by hand," or is it generated from a model?

How is the feedback obtained? Explicitly or implicitly?

How is the feedback evaluated?

How is the feedback incorporated into the next cycle?

Iteration and Rework

We can imagine that an ITERATIVE process implies an INCREMENTAL one, where each cycle adds to what was there before

But a more likely result of feedback during an iteration is the need to change or REWORK some aspects of what was there before

Fairley & Willshire analyze the extent and nature of this work and create a taxonomy for understanding it

Iteration and "Scope Creep"

When the iteration cycle is short, it only allows for small changes to be made on each cycle

When stakeholders propose small changes, they often think that designers are being inflexible or unreasonable if they refuse these change requests

But the cumulative effect of many small unplanned additions can be significant "scope creep" that distracts from planned iterations, consumes the project's resource, and causes it to miss schedules

How can scope creep be prevented?

Iteration and "Local Optimization"

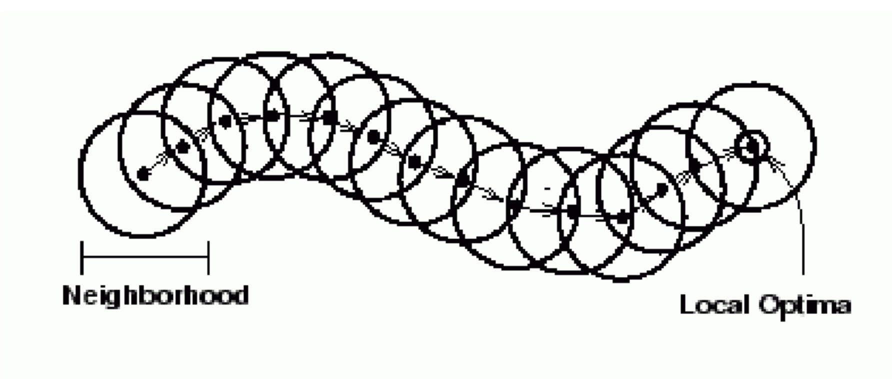
The design changes from one iteration to the next are often motivated by specific features or functions that caused used difficulties or otherwise failed to meet expectations

This specificity focuses the design/redesign activity on alternatives in the "neighborhood" of the current design

It makes it unlikely that radical design ideas will be considered, even though they might be significantly better

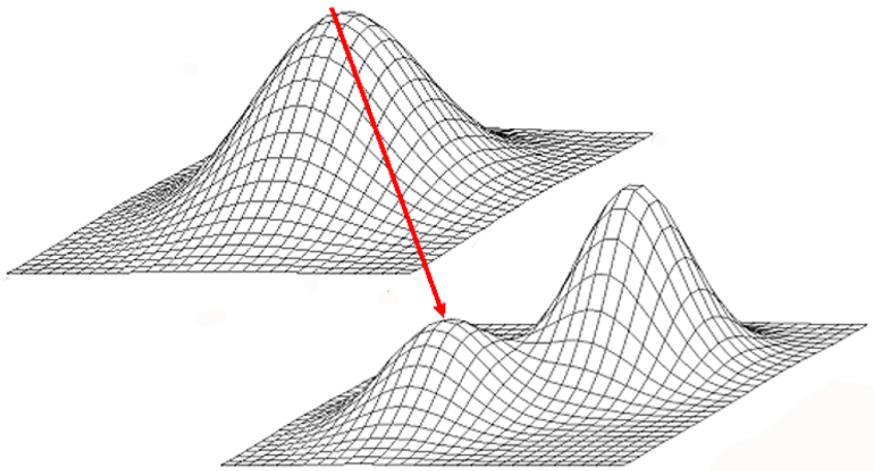
So the best solution that can be developed is the "locally optimal" one, which makes the starting point critical in retrospect, even though it might have been arbitrary or accidental

Local Optimization



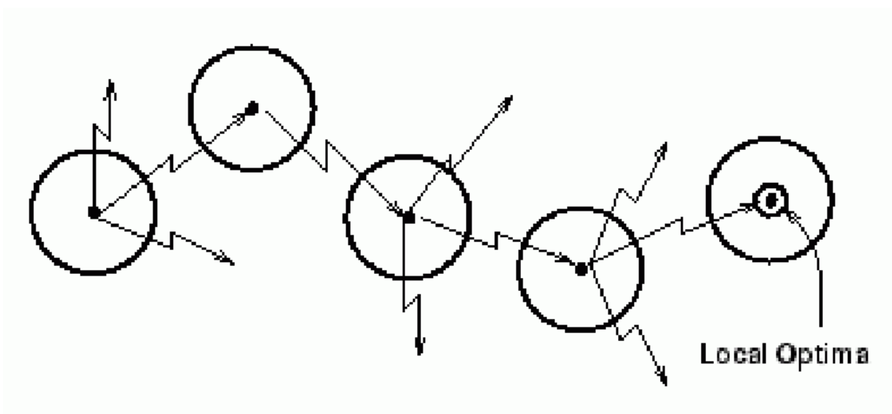
"Local optimization" results whenever the search for a better solution is limited to "nearby" alternatives in the design space

Another Depiction of Local Optimization



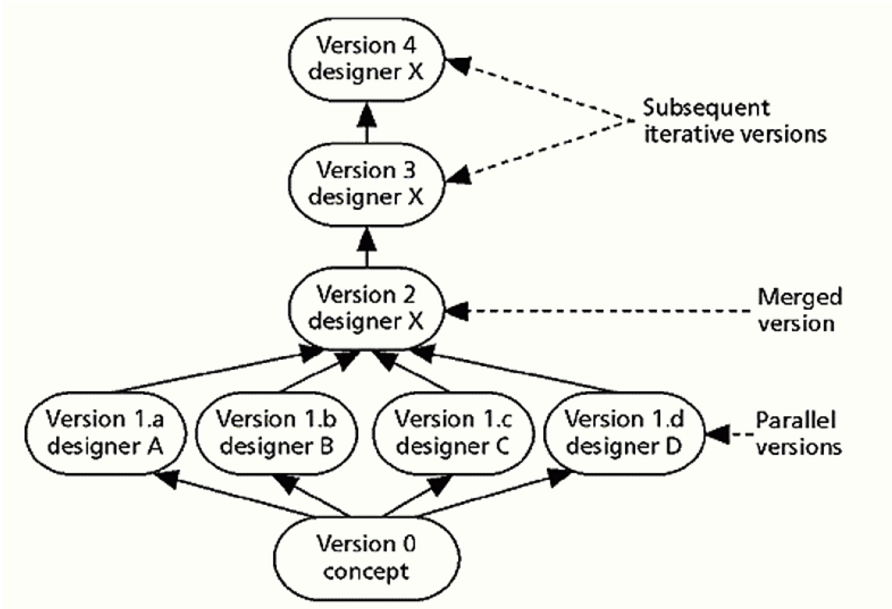
Local optimization gets to the top of the hill you're on... but you can't get to a higher hill because you'd have to first go down to get there

Finding a Global Optimum



You can increase the likelihood of finding the overall best solution if (a) you try many different starting points and (b) make longer and more random movements in "design space"

Parallel Iterative Design



Iteration in UI Design

Most user interface design and development involves steady refinement based on user testing and other evaluation methods

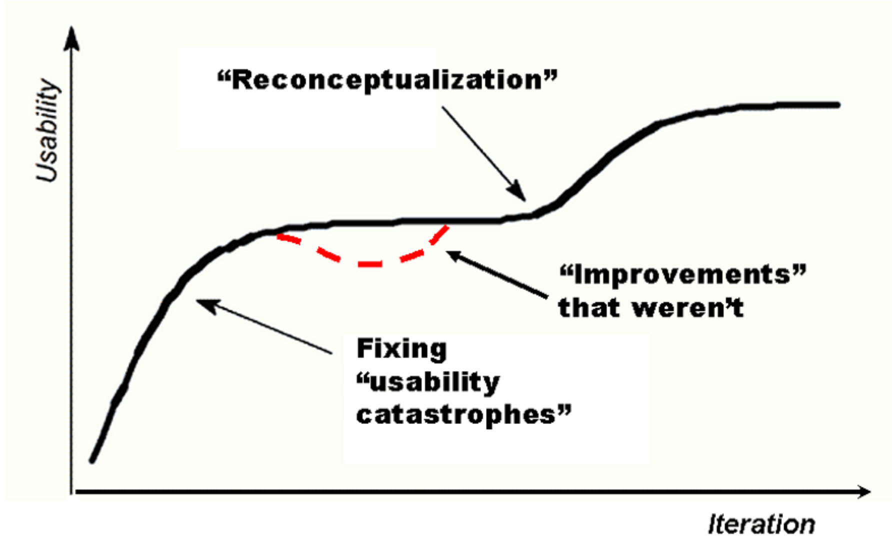
Most often, a design (or prototype or some other surrogate) is shown to or experienced by users, and the designer notes any problems that it has

These problems are then fixed in subsequent iterations

The design changes from one iteration to the next are normally local to those specific interface elements that caused user difficulties.

This iteration continues until time/resources run out or quality reaches an acceptable level

The "Benefits" of Iteration (Hypothetical Graph)



Interpreting "Iteration x Usability"

Ideally, each iteration improves on the previous version

The first few iterations should result in major improvements because they are finding and fixing "usability catastrophes"d

Later iterations yield diminishing returns because most usability problems have been eliminated

(And some changes to an interface may turn out not to be improvements after all)

Interface reconceptualizations -- moving to a new part of the design space -- can sometimes produce big gains in usability

The "Benefits" of Iteration?

Version	Efficiency (inverse task time)	Subjective Satisfaction	Correct Use (inverse error frequency)	Catastrophe Avoidance	Overall Usability Improvement
1	0	0	0	0	**0**
2	-18%	+6%	+114%	+156%	**+48%**
3	+12%	+9%	+268%	+582%	**+126%**
4	+12%	+4%	+513%	+595%	**+155%**
5	+26%	+17%	+513%	+1,406%	**+242%**

Version	Efficiency (inverse task time)	Subjective Satisfaction	Correct Use (inverse error frequency)	Help Avoidance (inverse help requests)	Overall Usability Improvement
1	0	0	0	0	**0**
2	+17%	+61%	-10%	+145%	**+43%**
3	-1%	+56%	-20%	+125%	**+29%**
4	+17%	+77%	-20%	+200%	**+49%**
5	+22%	+49%	0%	+575%	**+87%**

Key Questions

How do you know where you are on a Usability vs. Iteration function?

What are you measuring to assess the quality of each design? Is it objective?

Should you measure the same thing in every design context?

Is the function the same for all types of users?

Parallel Iterative Design in S-as-a-S Applications

Lindholm contrasts the UI design techniques typical of "shrink-wrapped" or deployed applications with those emerging in the "Software-as-a-service" context

Traditional "long cycle time" UI techniques don't fit well with "short cycle time" web development methods

Sometimes users are advised by labeling an application as "alpha," "beta," "labs," or "limited release" that it is has been made available for the purpose of getting early feedback

But the most often used technique is to introduce small changes to UIs without notice, and then measure if desirable behaviors increase (time spent on each page, transactions completed, ads selected, etc.)

For applications or services with large numbers of users, multiple design alternatives can be tested in parallel and iterative improvements made on a continuous basis

Iteration in Software Engineering -- Idealized

Because of the abstract medium in which applications or services are implemented, software can rarely be designed and developed without some amount of iteration

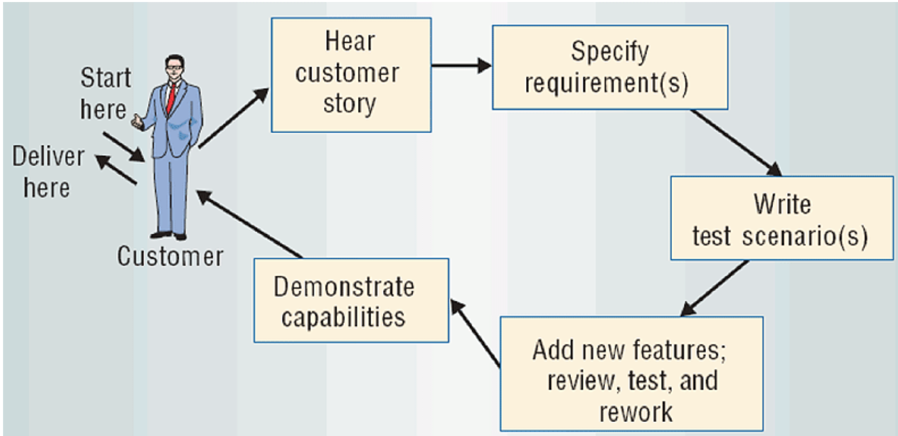
The idealized view of iteration in software engineering is that each cycle adds well-defined functionality or features

SW methodologies differ in how much breadth or scope of the software process is included in each cycle

At the end of each cycle the software (or other design artifact) is of production quality, fully tested, and deployable

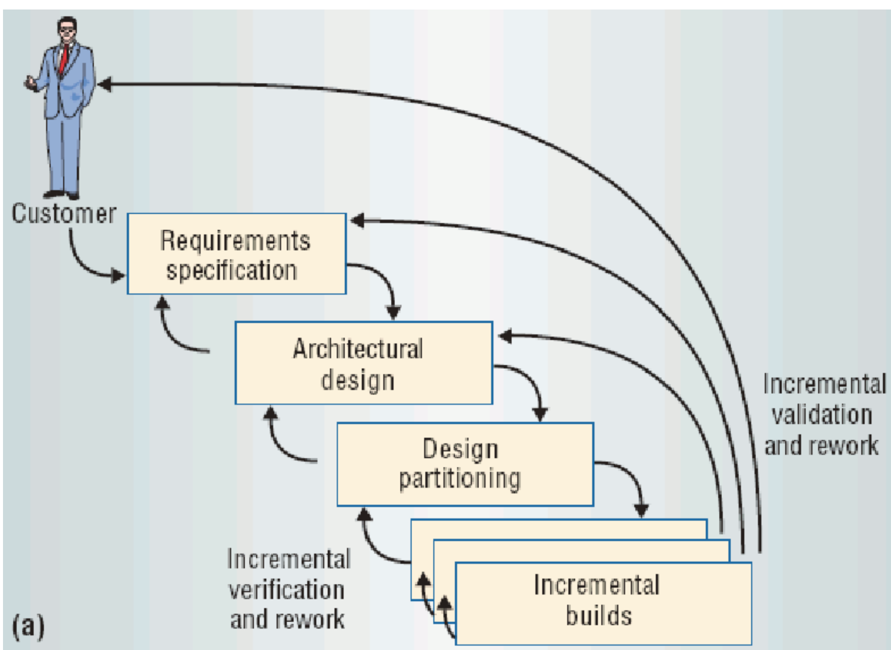
Put another way... the iterations can stop at any point because each iteration produces a stable design artifact

Agile Iteration

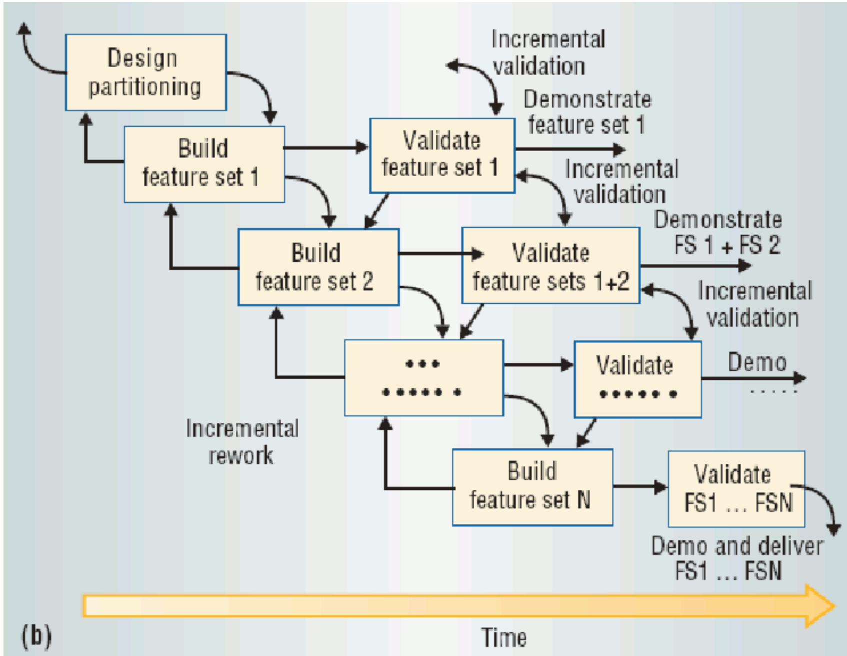


Each iteration in Agile methods includes requirements and development activities

Incremental Build Methodology



Incremental Build Iteration



Iteration and Use Cases

Use cases provide a straightforward way to manage iterative software design and implementation

Each iteration can complete the functionality needed for a simple use case, or a part of a complex one

The sequence of iterations can follow the priority of the use cases

Iteration in Software Engineering -- Reality

A more realistic view of iteration in software engineering is that adding new capability is only one of its purposes

This view acknowledges that adding new capability can introduce defects into existing capabilities

The challenge is to devise an iteration model that results in an acceptable amount of rework

A Taxonomy of Iterative Rework

Evolutionary

- Work that enhances or adds value to an existing artifact
- Required because of changes in requirements, constraints, or target contexts that could not have been foreseen

Avoidable Retrospective

- Problems or work identified previously that could have been dealt with then

Avoidable Corrective

- Work to fix defects detected in new capabilities or in older capability that the new capability exposes

Evolutionary Rework

Is "Good" if it adds value while meeting resource and schedule constraints

Is "Bad" if if violates resource and schedule constraints

Is "Ugly" if it shouldn't have been done because it wasn't meeting real requirements, was "gold plating," or "scope creep"

Avoidable Rework

Is never really "Good," but better to do a little bit now than a lot later and the total amount is controlled

Is "Bad" if it is routine, because that means that staffing or schedules are unrealistic, and developers are intentionally or unintentionally pushing work into

Is "Ugly" if it is so common and excessive that it means that developers aren't productive enough, are following poor process, or are being poorly managed

The "Design Structure Matrix" - Analyzing Information Dependencies

Experimentation and innovation in product development is facilitated by information exchange between designers of different subsystems or components

But if this "concurrent engineering" or "iterative design" isn't carefully managed, it can be inefficient and cause excessive rework and delays

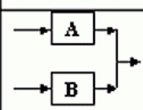
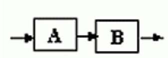
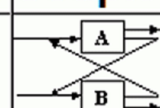
The "Design Structure Matrix" is a notation for analyzing and optimizing these iterative information flows

DSM needs to be in our "method toolkit" to help in modeling and designing processes

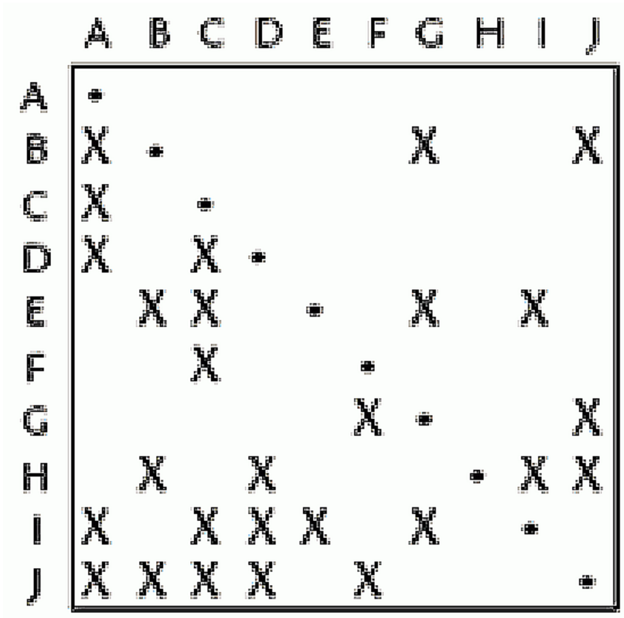
DSM Notation

Processes/activities/tasks are the rows and columns of a square matrix

The dependency or information flow relationships between any two processes i and j is indicated by the presence or absence of a mark in cells (i,j) and (j,i)

<table>
<tr><td>Relationship</td><td>Parallel</td><td>Sequential</td><td>Coupled</td></tr>
<tr><td>Graph Representation</td><td>
</td><td>
</td><td>
</td></tr>
<tr><td>DSM Representation</td>
<td><table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td></td></tr><tr><td>B</td><td></td><td></td></tr></table></td>
<td><table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td></td></tr><tr><td>B</td><td>X</td><td></td></tr></table></td>
<td><table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td>X</td></tr><tr><td>B</td><td>X</td><td></td></tr></table></td>
</tr>
</table>

DSM Example

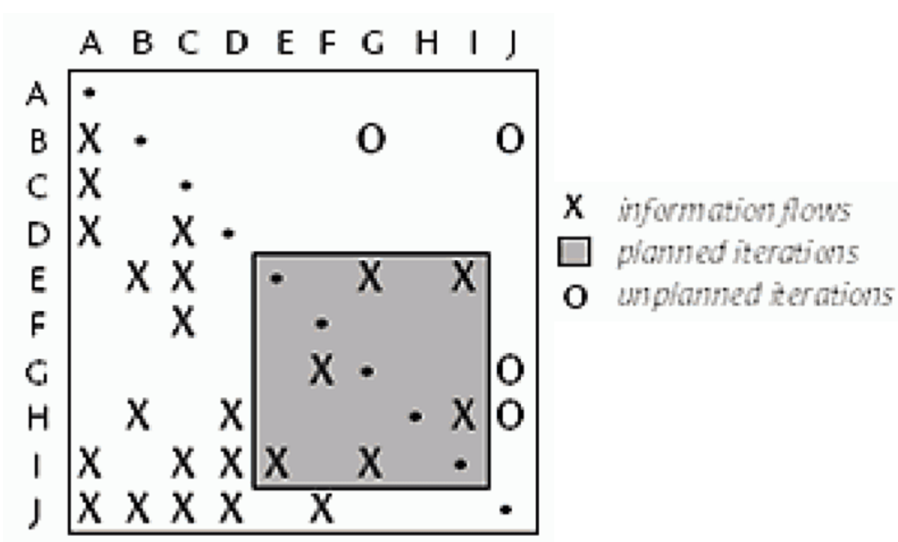


Reading across a row shows the inputs to the activity that names the row

Reading down a column shows the outputs from the activity that names the column

Marks above the diagonal indicate feedback from a later activity to an earlier one

DSM Example - Planned and Unplanned Iterations



DSM Techniques for Improving Process Models [1]

The overall goal is to reorder the rows or redefine the tasks to eliminate the Xs that were in the upper half matrix or at least to move them closer to the diagonal (this minimizes the number of tasks that are coupled in a cluster)

Start by identifying the first and last tasks (these have no inputs and no outputs, respectively)

Identify tasks that use information from the first task and determine if they are parallel, sequential, or coupled to each other

Reorder the rows to create the smallest set of coupled tasks (the box in which planned iteration takes place)

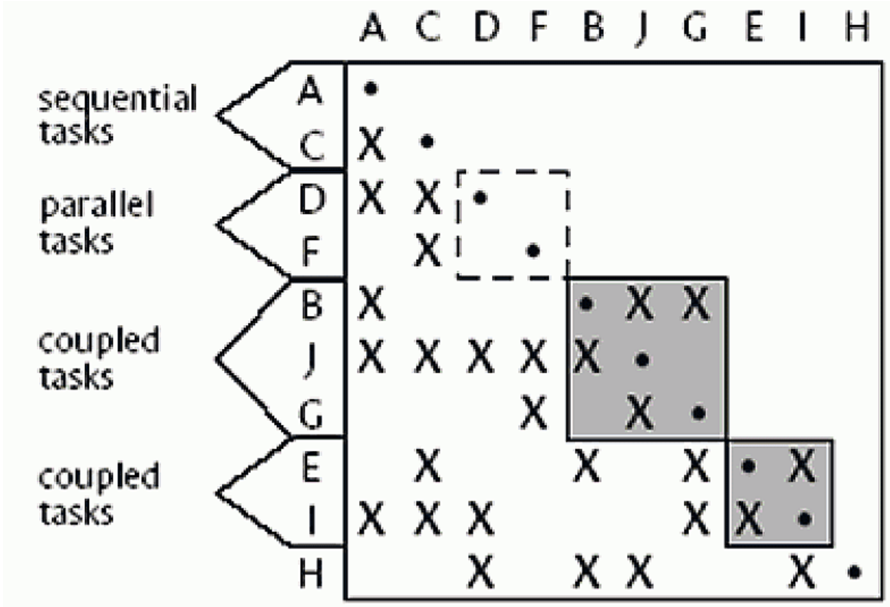
DSM Techniques for Improving Process Models [2]

Likewise, identify tasks that contribute information to the last task and determine if they are parallel, sequential, or coupled to each other

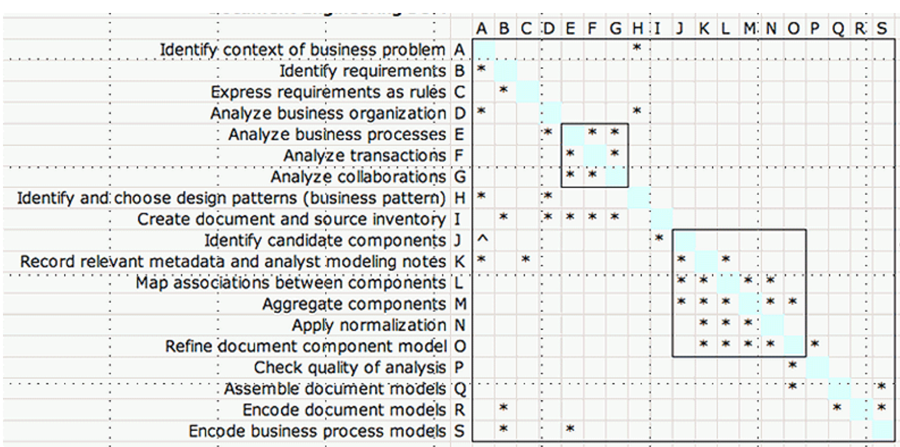
IN ADDITION OR INSTEAD OF REARRANGING THE TASK ORDER, YOU CAN REARRANGE THE PEOPLE WHO DO THEM

OR YOU CAN USE MODELING TOOLS THAT CAN PREDICT/SIMULATE DESIGN IMPACTS

DSM Example - Optimized



Using DSM in the Document Engineering Methodology



Readings for 17 November

Youn-Kyung Lim, Erik Stolterman, & Josh Tenenberg, “The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas”

Lars Holmquist, “Prototyping: Generating ideas or cargo cult designs"

Michael Schrage, “Never go to a client meeting without a prototype”