

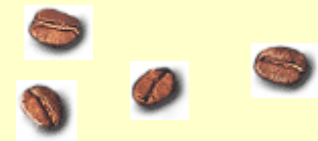
Vortrag am 19.Mai 2006
von
Hannes Restel

(eins vorweg: sorry für das denglisch!)

.... I spend the time on dependencies, because during the entire project execution phase, I benefit from them... and that is when I don't have time for anything anymore! These dependencies keep my schedule valid and up-to-date. Whenever I enter a change or whenever I update the schedule with actual Progress, they adjust the remaining schedule. I set them once when I have the time, and I benefit from them when I don't have time. They allow me to be proactive...."

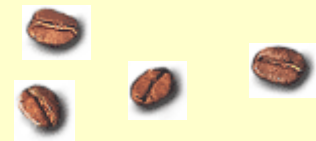


Quelle: Uyttewaal, „Dynamic Scheduling“



Vortrag in zwei Teile geteilt:

- I) Theorie
 - ◇ Was sind Dependencies
 - ◇ Vorstellen von Dynamic Scheduling
 - ◇ Typen von Dependencies
- II) Praxis (Modellierung in *MS Project*)
 - ◇ setzen/modifizieren von Dependencies
 - ◇ Überprüfen: Habe ich gut modelliert?
 - ◇ Tipps & Tricks



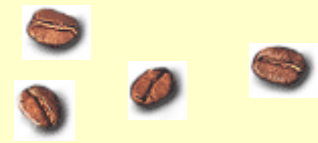
Bisher folgende Modellierung:


- WBS erstellt
- Tasks identifiziert
- Estimates für tasks erstellt

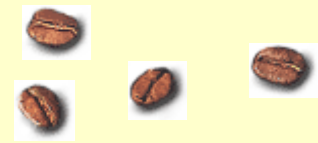
Jetzt folgt:

- Identifizieren und erstellen der Dependencies

Teil I: Theorie



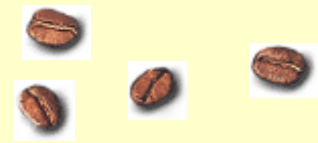
- dt.: *Abhängigkeiten*; oder en.: *relationship*
- Eine Abhängigkeit modelliert eine *gerichtete Verknüpfung* zweier Tasks
- Diese Verknüpfung kann logisch, zeitlich, usw. sein.
- Bsp.: A diagram illustrating a dependency between two tasks. It consists of two light blue rectangular boxes. The left box contains the text 'Kuchen backen' and the right box contains 'Kuchen essen'. A black arrow points from the right side of the 'Kuchen backen' box to the left side of the 'Kuchen essen' box, indicating a directed relationship.

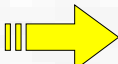


- Sie ermöglichen weitgehend automatische Aktualisierungen des Schedule bei Änderungen
- Änderungen treten meist in Execution Phase auf und müssen deshalb schnell und ohne viel Arbeit ins Modell übertragen werden können
- PM soll sich nicht mit Scheduling befassen sondern mit Durchführung des Projekts

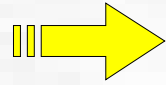
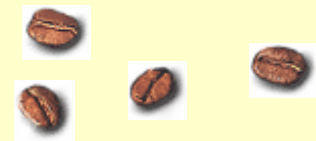
Zauberwort: Dynamic Scheduling

Was ist Dynamic Scheduling?



- Wichtiges Konzept (siehe Buchtitel :-))
- Allen Tasks im Schedule werden anstatt harter Termine und Zeiten nur Vorgänger/Nachfolger-Tasks zugewiesen -> Dependencies werden gesetzt.
-  Alle Tasks stehen in Verbindung zueinander, bilden also ein **Netzwerk**

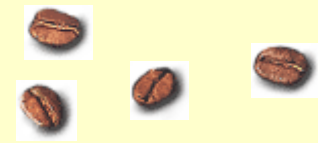
 **Ideal: Eine Änderung in Realität benötigt nur eine Änderung im Modell (Schedule)**



Änderungen wirken sich augenblicklich und automatisch auf gesamtes Netzwerk (also Schedule) aus, der Schedule ist immer up-to-date.

- Voraussetzung: alle Dependencies sind richtig gesetzt
 - ◇ wie macht man das: später
- Schedule ist automatisch
 - ◇ Valide
 - ◇ Up-to-date
 - ◇ änderungsfreundlich

Dynamic vs. Static Scheduling



Gegenteil vom *Dynamic Scheduling* ist *Static Scheduling*

Dynamic Scheduling:

- dependency-basiert
- Aufwand in Planning Phase
- Kein Zeitaufwand bei Änderungen
- Deshalb immer up-to-date
- Automatisch valide
- Da änderungsfreundlich -> Scenarios leicht erstellbar

Static Scheduling:

- zeit-basiert
- Kein Aufwand in Planning Phase
- Sehr viel Zeitaufwand bei Änderung
- Zu zeitaufwendig -> meist nicht aktuell
- Sehr fehleranfällig, da sich eine Änderung auf gesamten Schedule auswirken kann. Jeder Task muss geprüft werden.
- Scenarios praktisch unmöglich (Erklärung von Scenarios)

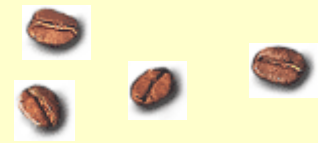
Beispiel aus Buch (100 Tasks, 180 changes):

Extraufwand in Planning Phase:	dynamisch: 8h;	statisch 0h
Extraufwand in Execution Phase:	dynamisch: 8h;	statisch 72h



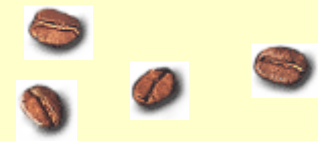
...noch Fragen?

Was sind Dependencies II ?



- Fundamental zum Verständnis: Abhängigkeiten beziehen sich **NICHT!!!** auf die Zeit!!!!!!
- sondern:
 - ◇ Abhängigkeiten sind Pointer!
 - ◇ Ausgedrückt in Konzepten:
 - Vorgänger – Nachfolger *oder*
 - Ursache & Wirkung *oder*
 - driver & follower

Was sind Dependencies III ?



- Fundamental zum Verständnis: Abhängigkeiten beziehen sich **NICHT!!!** auf die Zeit!!!!!!

Bsp. (allgemein verständlich):

Kuchen backen

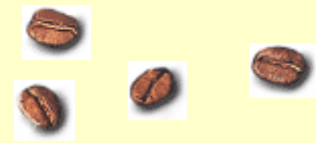
driver

Kuchen essen

follower

Zeit

Was sind Dependencies III ?



- Fundamental zum Verständnis: Abhängigkeiten beziehen sich **NICHT!!!** auf die Zeit!!!!!!

Bsp. (allgemein verständlich):

Kuchen backen

driver

Kuchen essen

follower

Bsp. (schwerer):

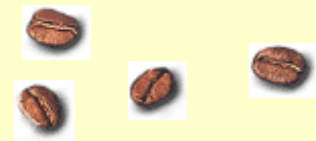
Prüfung vorbereiten

follower

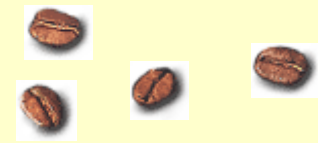
Prüfung ablegen

driver

Zeit



- Eine Abhängigkeit setzt immer am Beginn oder am Ende eines Tasks an
 - ◇ Später auch anders...
- Bei Abhängigkeiten zwischen zwei Tasks gibt es deshalb vier verschiedene Typen von Dependencies:
 - ◇ Finish-to-Start (*FS*) ← am häufigsten genutzt
 - ◇ Start-to-Finish (*SF*)
 - ◇ Start-to-Start (*SS*)
 - ◇ Finish-to-Finish (*FF*)

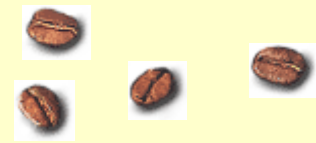


- Am häufigsten vorkommende Dependency

Bsp.:

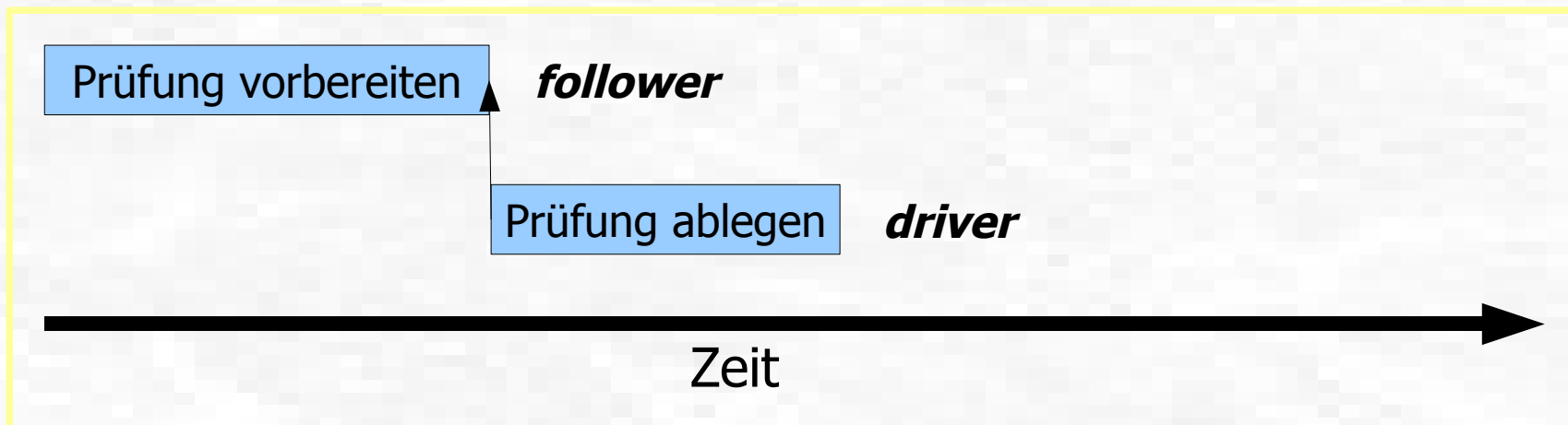


Erst nach Vollenden des Kuchenbackens (*finish*), kann mit Beginn des Kuchenessens gestartet werden (*start*).



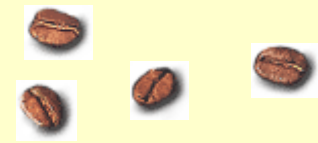
- Eher weniger gebräuchlich

Bsp.:



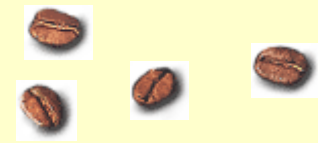
Genau bei Start der Prüfung muss das Ende (*finish*) der Vorbereitung eintreten.

Start-to-Start

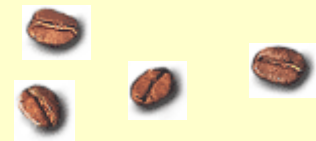


Zeitgleich mit Beginn der Aktivität **A** muss die Aktivität **B** beginnen.

Finish-to-Finish

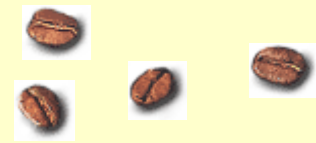


Zeitgleich mit Ende des Verputzens der Wände muss die Elektrik gelegt sein, beispielsweise um sofort mit dem Tapezieren der Wände beginnen zu können.



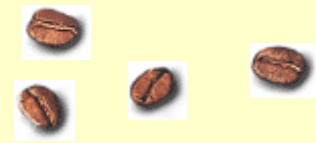
Oft sind zwei abhängige Tasks zu Verzögerungen gezwungen.

- Lag (*dt. Verzögerung*)
 - ◇ Positiver Zeitabstand zwischen zwei Tasks
- Lead (*dt. Führung*)
 - ◇ Negativer Zeitabstand (*Überschneidung* zweier Tasks)
 - ◇ Wird häufig eingesetzt, um Schedule zu stauchen
 - Das birgt natürlich Risiken!

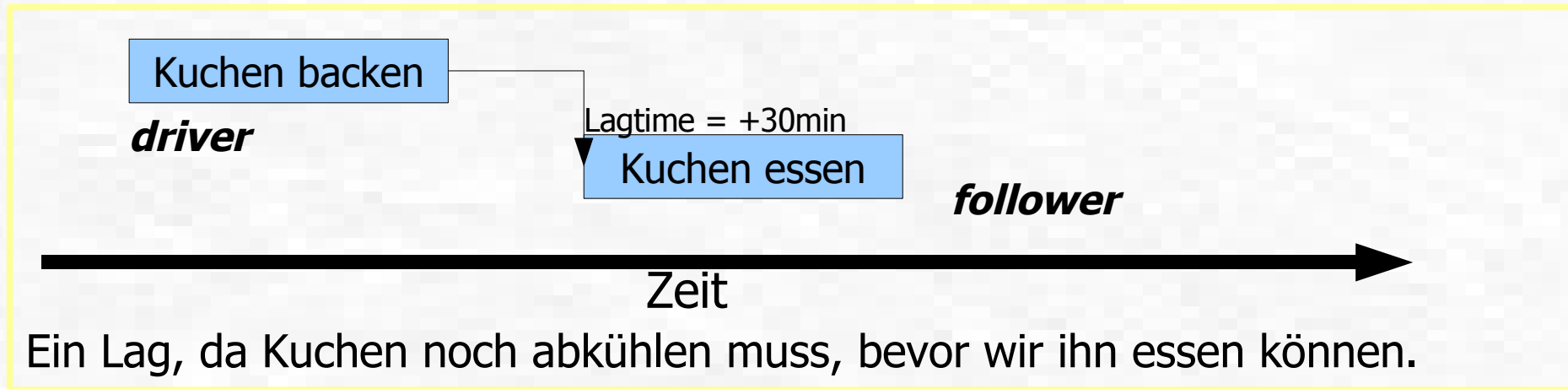


- Zwei Arten der Verzögerung:
 - ◇ Absolut
 - Bsp.: +2d, -4h, -30min, +4ed
 - ◇ Relativ
 - Bsp.: +30%, -50%
 - Der Prozentsatz ist immer bezogen auf relative Länge des *drivers* (bzw. *Vorgängers*)
- vorweg: *MS Project* kennt nur *Lags*
 - ◇ Also: wenn *Leads* gewünscht, dann negative *Lags* eingeben!

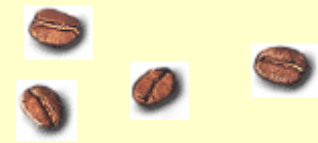
Bsp. Lead & Lag



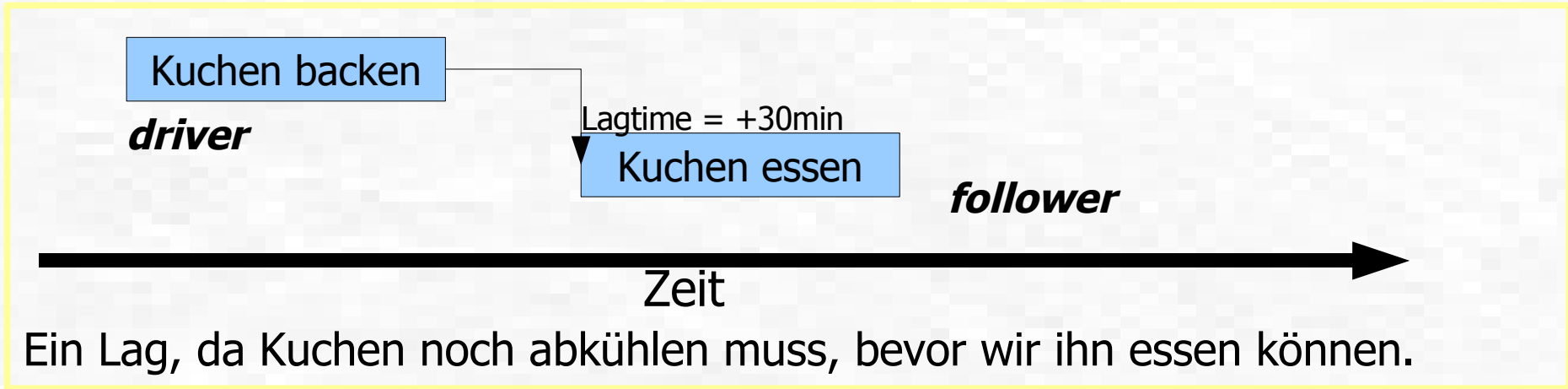
Bsp. Lag:



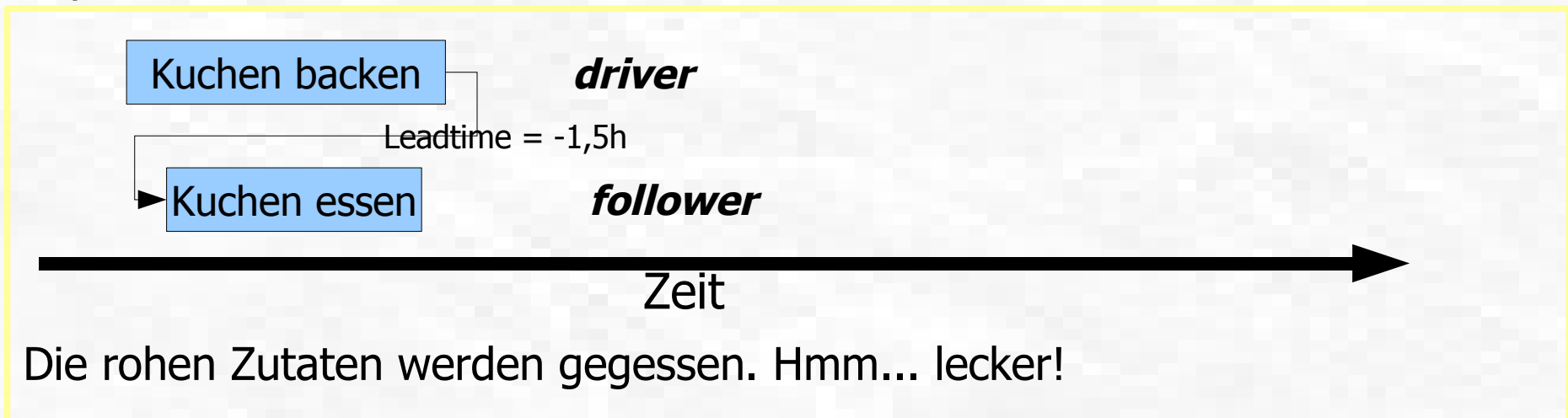
Bsp. Lead & Lag

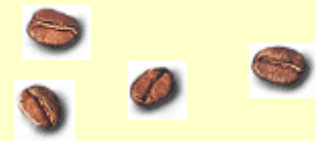


Bsp. Lag:

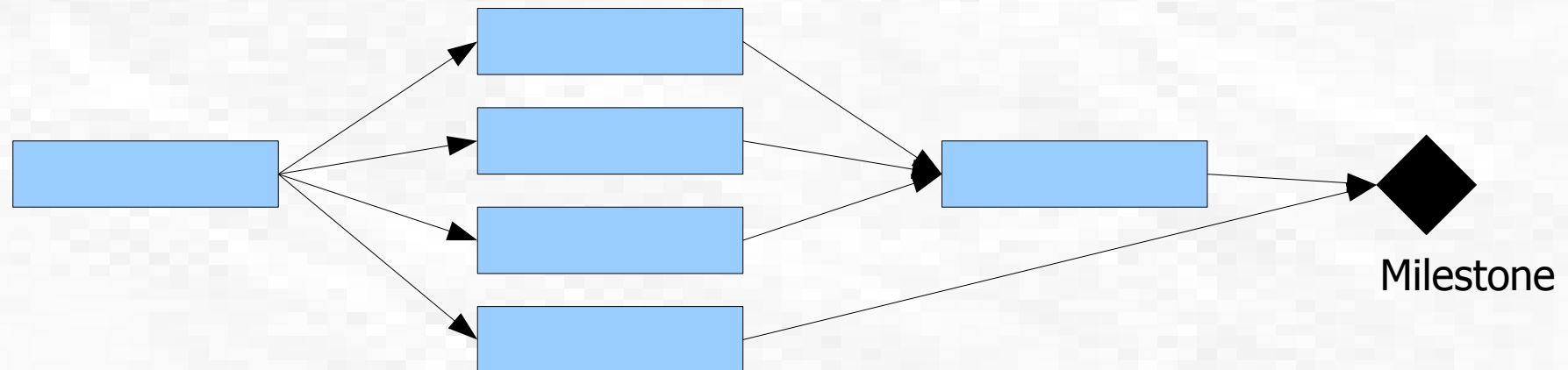


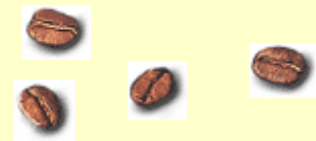
Bsp. Lead:





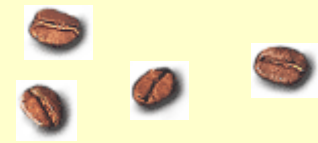
- Das Netzwerk (Schedule) kann multiple Dependencies pro Task beinhalten
- Vorteil: erhöhte Parallelität von nicht zueinander abhängigen Tasks mit ggf. daraus resultierendem kürzerem geplanten Projektende



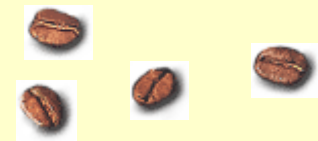


...noch Fragen?

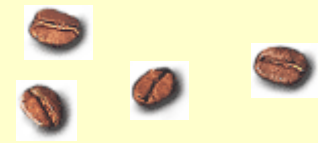
Teil II: Praxis mit *MS Project*



- MS Project bietet folgende Möglichkeiten, Abhängigkeiten einzugeben
 - ◇ Im View ***Spreadsheet***
 - Demonstration
 - ◇ Im View ***Gantt-Chart***
 - Demonstration
 - Tipp: der *Task Information Dialog* (Shift + F2)
 - ◇ Im View ***Network Diagram***
 - Demonstration
 - Tipp: die Buttons *Hide Fields* und *Show Link Labels*

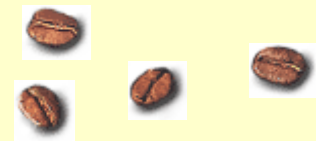


- Hinweise und Tipps für *MS Project*
 - ◇ Meilensteine haben die Dauer „0“
 - ◇ Harte (*mandatory*) und weiche (*soft*) Abhängigkeiten
 - „hart“: absolute Abhängigkeitsreihenfolge ist vorhanden
 - „weich“: Reihenfolge ist willkürlich, kann also verändert werden
 - Da keine *MS Project* keine Unterscheidung macht, für weiche Abh. einen Eintrag im Task Notes-Feld machen
 - ◇ External Dependencies
 - sind Abhängigkeiten, auf welche wir keinen Einfluss haben (z.B. Liefertermine für Hardware)
 - Modell dafür in *MS Project*: Meilensteine setzen
 - Hinweis geben wg. Zuständigkeiten/Responsibilities

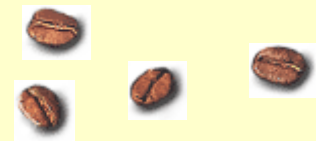


Eine **gute** Modellierung besitzt folgendes („Die Vier Gebote“):

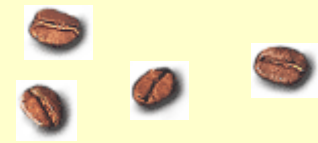
1. Nur einzelne Tasks und Meilensteine, dürfen Dependencies besitzen, *Summary Tasks* nicht
2. Jeder Task besitzt zwei Dependencies
 - Ausnahme: Anfangstask(s) und Endtask
3. Besitzt ein Task keinen logischen Nachfolger, so ist der *Milestone* des zugehörigen *Summary Tasks* der Nachfolger
 - Deshalb: zu jedem *Summary Task* einen *Milestone* schreiben, selbst wenn *Termin* noch nicht vorhanden/bekannt
 - Konzept der Decision Point Dependencies
4. Vermeide und lösche „springende“ Abhängigkeiten



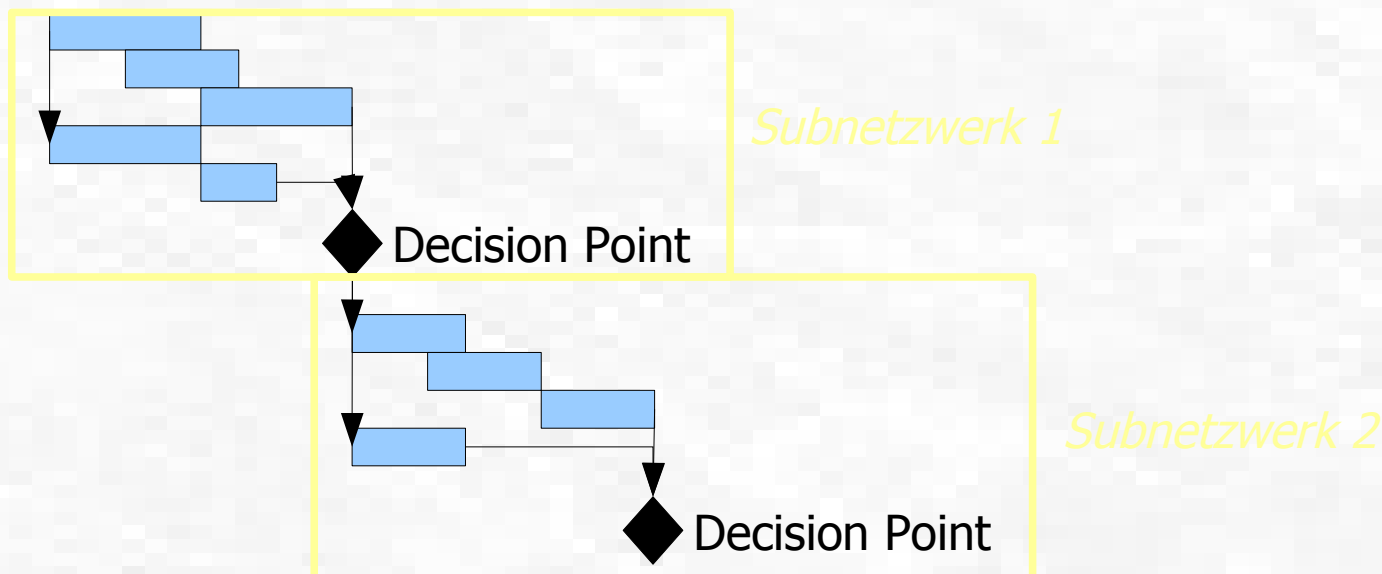
- Nur einzelne Tasks und Meilensteine, dürfen Dependencies besitzen, *Summary Tasks (STs)* nicht
 - Begründung
 - Der *kritische Pfad (critical path)* ist bei *STs* schwerer nachvollziehbar
 - Test auf Vollständigkeit des *Abhängigkeits-Netzwerks* komplexer
 - Die Typen *FF* und *SF* sind nicht bei *STs* modellierbar
 - Demonstration

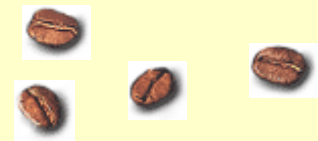


- Jeder Task besitzt zwei Dependencies
 - d.h. das Netzwerk muss vollständig sein
 - Nur dann kann *MS Project* Dynamic Scheduling durchführen
 - Nur dann ist der *critical path* (samt seinen *critical tasks*) eindeutig und macht Sinn
 - Vorausblick: zur Optimierung werden meist diese *critical tasks* verändert / verkürzt / *leads* hinzugefügt
 - Wichtig: Netzwerk darf keine Kreise enthalten
 - Demonstration

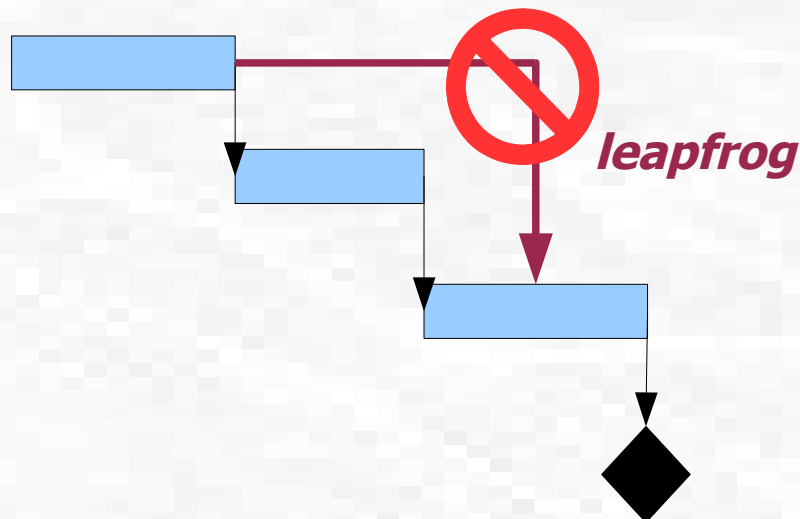


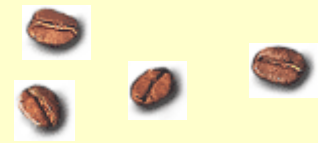
- Decision Point Dependencies + Milestone targeting
 - Jeder Task eines *Subnetzwerks* findet sein Ende (auch über transitive Tasks hinweg) im *Milestone* des Subnetzwerks.
 - Subnetzwerk: Jeder *summary task* bildet ein Subnetzwerk aus
 - Decision Point: der letzte *Milestone* eines *Subnetzwerks*



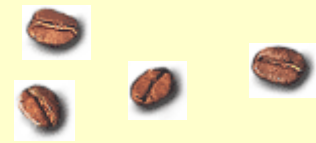


- Vermeide überspringende Abhängigkeiten
 - Auch *leapfrogging* genannt
 - Dependencies sind transitiv. Vermeide deshalb direkte Sprünge, wenn es auch indirekt geht
 - Direkte Sprünge sind redundant

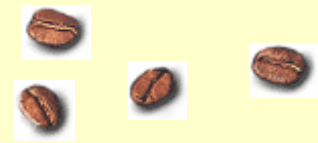




- Gute Modellierung - Zusammenfassung
 - *Dynamic scheduling* anwendbar
 - Weniger fehleranfällig
 - Übersichtlicher und verständlicher
 - Lässt sich leichter an Mitarbeiter und Vorgesetzte vermitteln



...noch Fragen?



Schluss... + Danke !!!