

Wiederholung letzte Stunde:

Unified Process:

Phasenabschluss durch Erreichen eines Meilensteins

In jeder der 4 Phasen (Inception, Elaboration, Construction, Transition) werden die Subphasen (Requirements, Analyse, Design, Implementation, Test) durchgeführt! Dieser Kreis kann abgebrochen werden (z.B.: in der Analyse merkt man dass Requirements nicht passen), muss dann jedoch bei Requirements neu gestartet werden.

Requirements:

Interview mit Kunde

Um

...

Analyse

Erste Beschreibung der Requirements

Use Cases

Klassendiagramme

Alles auf Verständnisebene des Kunden (für nicht Fachkundige)

Elaboration

Beschreibung der Problem domaine

Festlegen der Architektur

Welche Klassen gehören zusammen? -> kann man Komponenten bilden?

Modulbildung?

Am Ende der Elaboration steht die Architektur (Welche Klassen braucht man, wie werden die Komponenten gebildet, ...) Hier sollten auch alle User Anforderungen durch die Requirements festgelegt sein;

Im Wasserfallmodell wird jeder Phase gleich viel Zeit gewidmet, die Phasen starr hintereinander ausgeführt; beim Unified Process werden die verschiedenen Phasen immer wieder durchlaufen, verschieden viel Zeit pro Durchlauf je Phase investiert.

Man entwickelt immer in einem lauffähigen System, d.h. es ist möglich dem Kunden etwas zu zeigen!

Dazwischen ist es möglich Risikoanalysen durchzuführen um festzustellen ob ein Projekt durchführbar bleibt.

Im Gegensatz zum Wasserfallmodell wird bei Unified Process sehr früh begonnen kleine Teile zu implementieren und wirklich zu coden; dadurch kann man Designfehler früher erkennen bzw. auf verspätete Wünsche leichter reagieren.

Vo vom 13.11.2007

#### Analyse:

Requirements betrachten und einen Lösungsansatz ermitteln (ist ev. nicht die finale Lösung);  
Dient dem besseren Verständnis der Requirements  
Grundsätzliches Festlegen, grobe Klassendiagramme zeichnen

Komplett

Vollständig

Lesbar auch für Laien

Testbar mit realen Anwendungsfällen

Was macht das Programm, nicht wie arbeitet es!

#### Business Class Models:

Erstes Klassendiagramm;

Haben meist nur einen Klassennamen, kaum Attribute oder Funktionsnamen;

Es soll sehr abstrakt bleiben primär die Businesssprache (der Problem domain) verwenden;

Anfangs im Klassendiagramm sehr abstrakt denken, z.B.: nur Assoziation; ev. nicht mal die Multiplizität bzw. Komposition / Aggregation festlegt.

#### Design:

Spezifiziert den Lösungsvorschlag aus der Analyse;

Aus dem Design soll es 1:1 möglich sein das man beginnen kann zu implementieren.

Das Design kann durchaus Implementierungssprachen abhängig sein.

Aus der Analyse kommen ev. Klassen mit viel zu vielen Aufgaben; wie teile ich diese auf, wie kommunizieren diese untereinander; welche ruft welche auf, wie kommunizieren sie untereinander;

Das Design ist komplett wenn man aus dem Design heraus den Code autogenerieren kann und beginnt Code zu schreiben.

#### Architektur (in der Elaboration Phase):

Beziehungen zwischen den Elementen (Klassen bei kleinen Projekten /Komponenten bei grossen Projekten);

Wie ist das System strukturiert; wie teile ich das Problem auf, welcher Teil macht was?

Ev. Teilung der Aufgaben, Gliederung in Funktionsgruppen

Für die einzelnen Komponenten soll es wieder eine Analyse und ein Design geben!

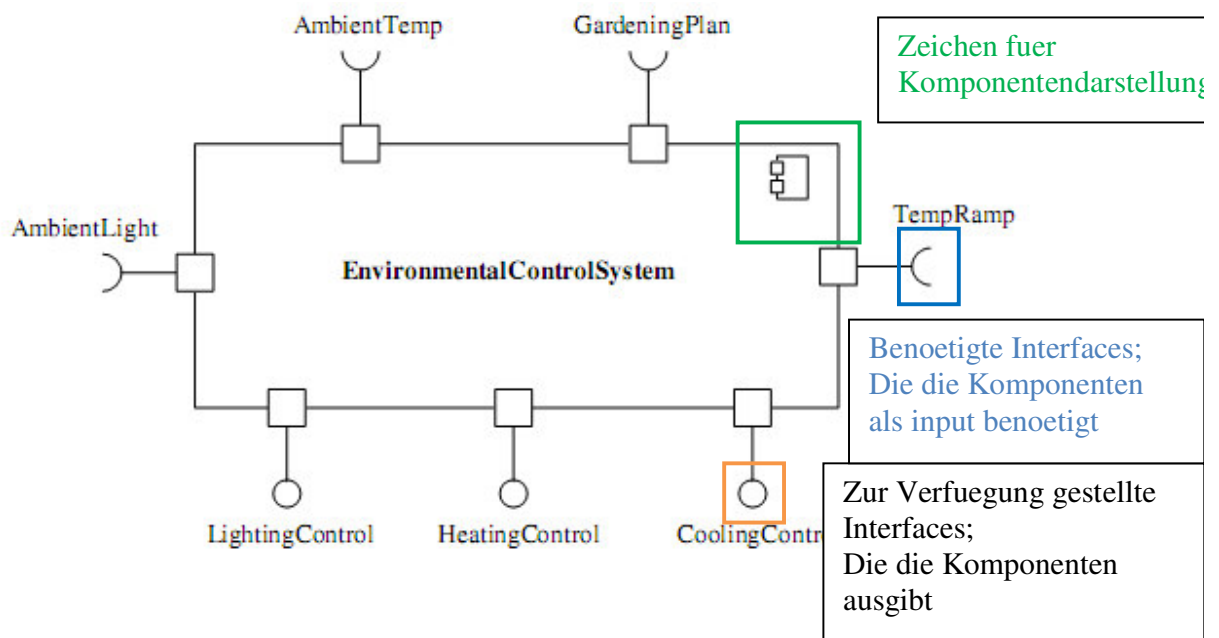
Komponenten:

Komponenten sind ausfuehrbare Softwaremodule, sie sind wiederverwendbar;  
 Es soll Funktionalitaet sinnvoll zusammengefuegt werden;  
 Es soll nicht zu viele Assoziationen zwischen den Komponenten geben; den Datenaustausch soll moeglichst gering gehalten sein; dazu ist ein wohl definiertes Interface notwendig! Im Gegensatz zu Klasseninterfaces werden zwischen Komponenten nur basisdatendypen ausgetauscht (String, int, ...) um Wiederverwendbarkeit und Austauschbarkeit sicher zu stellen.

UML 2.x Notation fuer Komponentendarstellung:

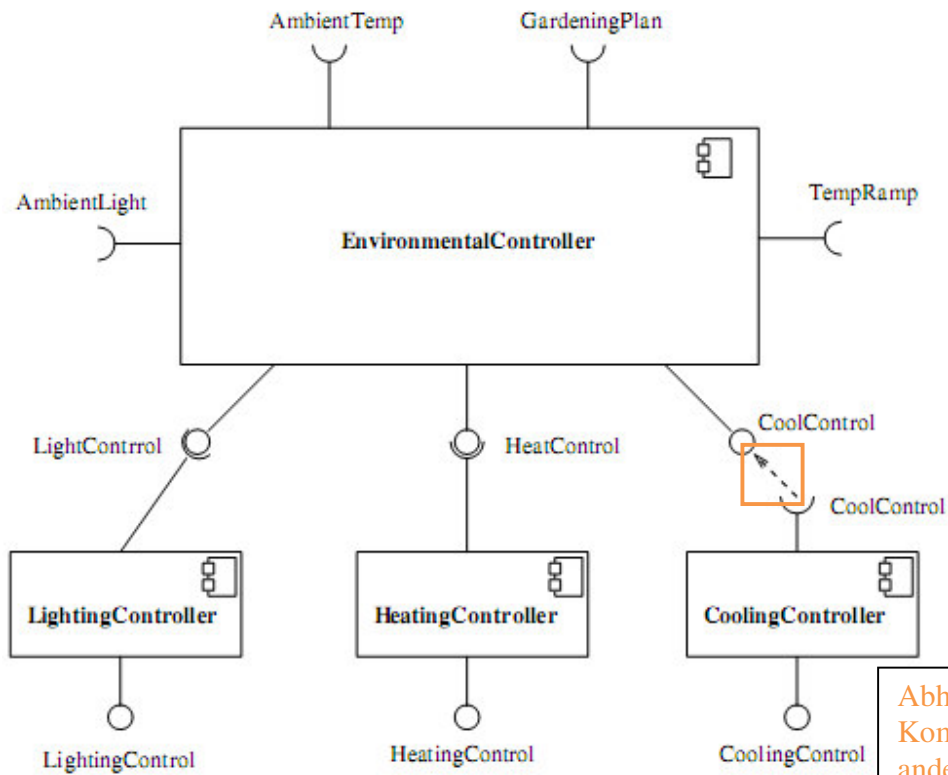
Black Box view:

Nur die Draufsicht auf das System, welche Anschluesse gibt es nach aussen;  
 nicht wie funktioniert es im Detail:

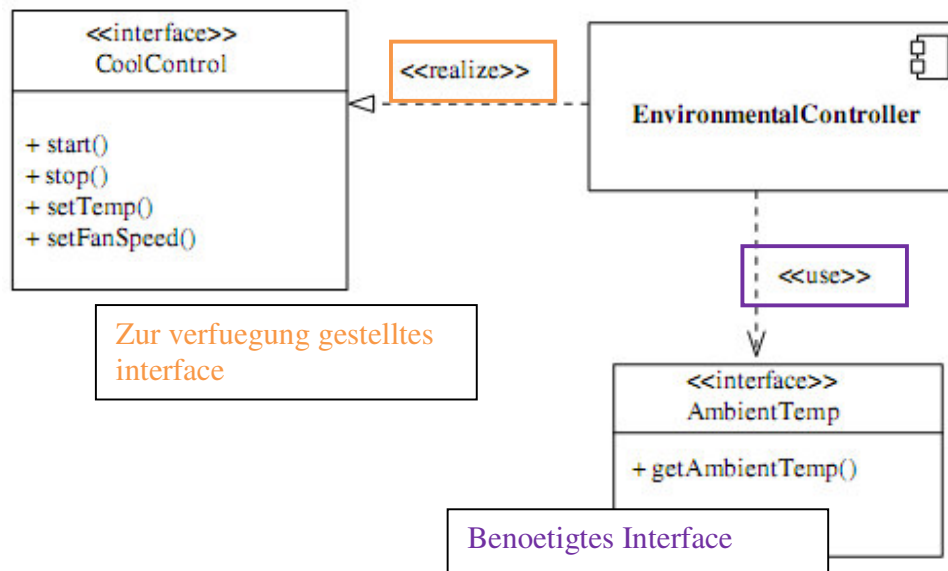


White box view:

Wie ist die Komponenten aufgebaut (kann auch sein das es mehrere Komponenten beinhaltet)

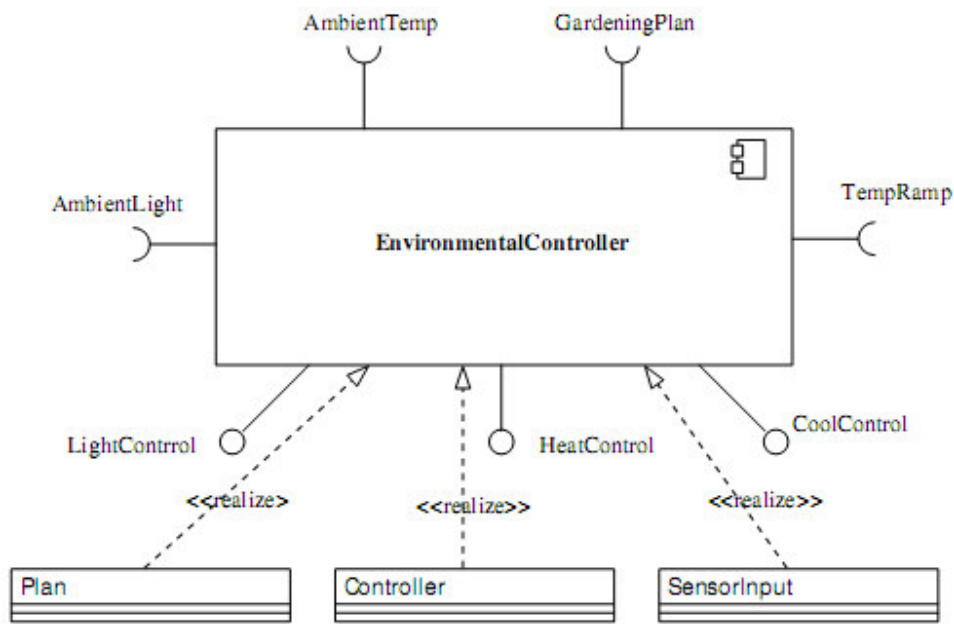


Abhängigkeit;  
Komponenten benutzt  
ander Komponente



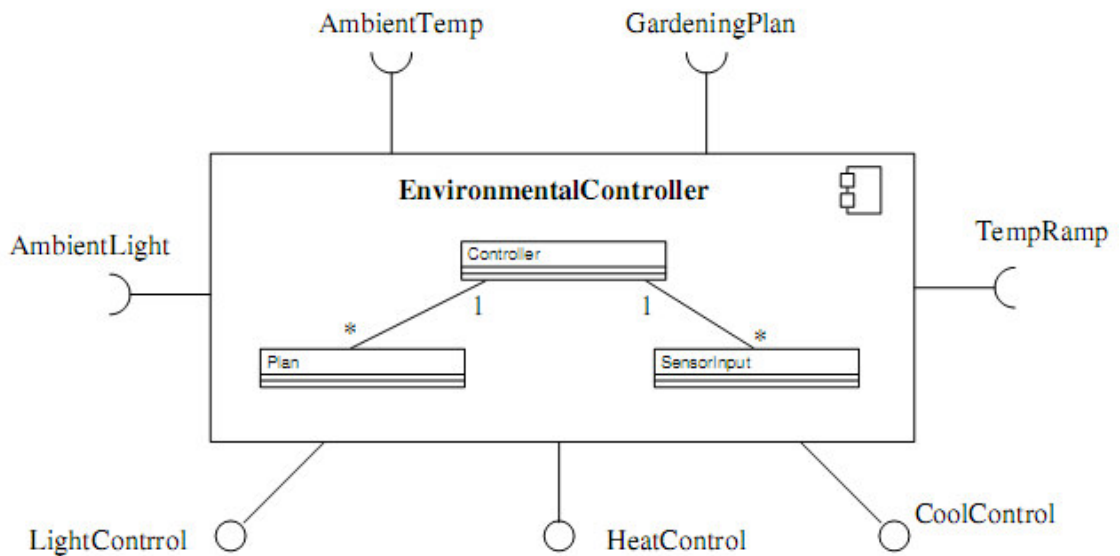
Zur verfügbaren  
interface

Benötigtes Interface



Der gesamte Kontroller wird dur diese 3 Klassen realisiert!

Alternative Darstellung:



Das Klassendiagramm kann in die Komponente gezeichnet werden, so fern dieses nicht zu komplex ist:

Es ist darauf zu Achten:

- Es gibt Analyse auf Anforderungsebene
- Es gibt Analyse auf Klassenebene
- Es gibt Analyse auf Komponentenebene