Component-Oriented Design and Architecture

The purpose in this lab is to create a coupling table for the first monolithic Asteroids game implementation (IntroLab), perform a dependency analysis on The Monolithic Asteroids Game and the The NetBeans Asteroids Game (NetBeansLab2), then finally reflecting on what influence Component-oriented Design has on large systems.

First the coupling table for the first monolithic Asteroids game (IntroLab) implementation was made, just as following:

Number	Class	Depends on	Dependency depth
1	Player	Game, SpaceObject	4
2	SpaceObject	Game	3
3	GameState	GameStateManager	2
4	PlayState	Player, GameStateManager, GameKeys	5
5	Game	GameInputProcessor, GameKeys, GameStateManager	7
6	Main	Game	8
7	GameInputProcessor	GameKeys	2
8	GameKeys	**none**	0
9	GameStateManager	GameState, PlayState	6

And as we see there is a lot of dependency depths out there in the project, now lets take a look at the dependency depth on the NetBeansLab2 project.

We will look at the build-time dependency:

Number	Component/Library	Depends on	Dependency depth
1	Common	**none**	0
2	NetBeansLab1-branding	**none**	0
3	NetBeansLab1-app	SilentUpdate	1
4	Core	Common	1
5	Player	Common	1
6	Enemy	Common	1
7	SilentUpdate	**none**	0

Reflection and comparative analysis
As we see, there is a very big difference between these two projects.
Just look at how many classes in the monolithic Asteroid project depends on other classes. It is a LOT compared to the component-based NetBeansLab2 project and that's why it is good to use component-based software systems in terms of larger software systems, since we will reduce the coupling in the system by only dependending on bigger parts of the system than on the many classes. Component-based softwares are also more efficient when it comes to dependencies, as we see we only dependend on max 1 component in most of the components in NetBeansLab2. This will then help us a lot when selling the software to other company or when we buy a software from other company, because we will now be able to switch out a specific component and avoid any dependency-related conflicts, when we want to use our own components with the software system we just bought for example.