1.2
Difference between Generic and Custom made software:

Specification:

In a custom software process the specifications are initially specified by the customer. As such the software developer has the task translating them into something useful.

In a generic should process the specifications are determined by developers according to a business case. As such the requirements have a technical nature from the start, but will lack knowledge of the domain.

Duration:

In a generic process the timeline is completely determined by the software company, and not influenced by a customer “breathing down your neck”. As such other considerations can be taken into account (eg. Market, ressources etc.)

Customer involvement:

In a generic process the customer is not involved in development. However, domain knowledge can be obtained from consultants etc. A generic piece of software will typically have to be tailored afterwards to a customers specifications.

A custom tailored piece of software will meet user demands from the beginning. For users of a generic piece of softawre the tailoring has to be done by the company if they want a useful piece of software.

1.3

- Acceptability
- Maintainable
- Dependable/secure/Reliable
- Efficientability

- Heterogeneity
- Economically sound
- Reusable (for other projects)
- Future proof.

1.8
Unlike the doctor/medical profession, our business benefits from creative thinkers who act outside the norm. Preventing “homebrewers” from creating software would kill a lot of innovation. The dangers of the “homebrewing” is that too few people will teach them selves to properly document/test/specify their code. As such the standard in quality will drop, but these faults can be corrected if software reaches a certain level of success.

2.1
Car braking system: Waterfall model. Strict specifications are required from day one for safety and legal reasons. Also an iterative process is not really usable because that would be expensive in car recalls, and possibly lethal (for the users).

A virtual reality system to support software maintenance:
Iterative development. Its very hard to determine requirements for the final piece of software so they would naturally change throughout the development process.

A university accounting system that replaces an existing system:
This might be a problem solvable with a hybrid development process. The existing system can act as an extensive specification, and the users current change request can be seen as further specifications, making it perfectly clear what the final piece of software should be able to to. (had the customer been a national bank, a strictly planned process would be the perfect choice. This system isn't too large to accommodate other type of development, and certain aspects of the end product (eg. UI/UX) would benefit greatly from the deeper customer involvement that an iterative process offers. A university is a public institution, and has to offer tender. Because of that, here has to be an extensive specification of costs and duration. There's also specific laws for accounting calculations, that should be met.

An interactive travel planning system that helps users plan journeys with the lowest environmental impact: The main risks of a faulty system is that efficiency isn't good enough. This is due to the fact that the system will depend on a huge amount of data, which will change frequently, and stem from many different sources. With a travel system, specifications often change, making it a good candidate for an iterative process. Also the customer base will consist of end users, which means that UI should be prioritized, if the system shall attract a customer base (and keep them)

2.5

Specification -> Requirements documents It gives you the overall goals for the system.
Design -> UML diagrams, class diagrams, CODE
Validation -> Test plans(acceptance, system integration plans), unittest-CODE
Evolution -> change requests -> new System.

2.9
What are the advantages of providing static and dynamic views of the software process as in the Rational Unified Process?
In your group, discuss the following exercises from the [SE9] book:

Exercise 1.2, 1.3, 1.8, on pp. 25.
Exercise 2.1, 2.5, 2.9 on pp. 54-55.
Write up a resume of your discussion in the report.for

Static view will give you an idea of the integrations throughout the system (eg. how parts are connected) Dynamic views however will give you keen insight into the actual function of the software. As such static view is only useful in the system design phase.

UML diagrams for car rental system:
We've chosen to illustrate with our diagrams a point of sale in our system. We've done so because it spawned some problem when we discussed it. Our discussion primarily centered around the existing entities in the system. For instance we had a specific discussion about which roles should be present where. A specific example is a salesman. He, of course, plays a key role in the usages of the system, as he is the main point of entry for sales. As such we have included him in the use case diagram, to illustrate to high level functionalities of the system. We have not however included him in the sequence diagram, as he is not an "object" in the system where we abstract the sale to a level where it only exists as an order. We've also chosen to this

part of the system to illustrate the difference between the diagrams. For instance the domain model is very high level, where as the sequence and class diagrams include the backend database code.

Sketch of layers:
As with our UML diagrams we've looked into the "place order" scenario. This sketch once again illustrates the important layers of the application. The interface layer, which doesn't map directly to the entities in our system, the application logic layer which shows out acting entities. And lastly the technical services layers, which handles the backend functionalities of database access.