Handout: Selected Agile Release Patterns

1. Product backlog patterns

1.1 New user set

- **What:** Create an interface for a user set that previously only used the solution indirectly
- **Context:** Internal system with external stakeholders
- **New context:**
    - Internal users continue to use the old system, but have less work.
    - External users will be interacting directly with the system, which may mean more traffic, increased usability requirements and more business
- **Considerations:**
    - Which user set?
    - Which tasks?
    - Which interface?

1.2 Cut to non-negotiables

- **What:** Include only product backlog elements that will make a release unusable if missing.
- **Context:** You have a large product backlog with many "must-have" features.
- **New context:** The release contains few of the value-added features that made us want to do the project in the first place.
- **Considerations:**
    - What to cut?
    - How to still make the product interesting?
    - Are there non-expensive pain-relievers or differentiators that can compensate for missing value?

1.3 Partition the workflow

- **What:** Analyze the full value chain supported by the old system. Replace some steps in the value chain with the new system
- **Context:** A few long workflows cover the whole system.
- **New context:** Some steps will be performed in the new system and some will be performed in the old system
- **Considerations:**
    - Where to start? Beginning? End? Middle?
    - Which users will need to change behavior? Will some users be required to use both the old and new system?
    - Can parts of the workflow be automated or streamlined in the new system?

2. Legacy data patterns

2.1 Shared database

- **What:** The new and the old system use the same database
- **Context:** The data of the old system must be continued into the new system. The new and the old system need to coexist. Don't have time or money for more advanced integration.
- **New context:** New system uses the old data model and database.
- **Considerations:**
    - What changes can be made to the data model?
    - Can the new use corrupt the data from the point of view of the old system?
    - What will happen if the old system updates data used by new system?

2.2 Replicated database

- **What:** Data is regularly transferred between the new and the old system
- **Context:** A limited subset of the data must be managed by both the new and the old system at the same time. Alternatively: the new system cannot directly interface with the old database
- **New context:** New and old system share some underlying data (with some delays after updates)
- **Considerations:**
    - Should the old system or the new system be master? At what point may we want to change this?
    - Should updates in the "slave" system be propagated into the "master"? How often?
    - How often should the replication happen?
    - Should the whole database be replicated or just changes? How do we find changes?

2.3 Data service layer

- **What:** Expose the old system data through a service. Adapt the data to the desired model for the new system.
- **Context:** The data model will undergo substantial changes or the new system cannot directly interface with the old database.
- **New context:** New system can retrieve and update information in the old system through a service interface. A new storage mechanism that conforms to this interface can be built for the new system.
- **Considerations:**
    - What technology should be used for the integration? What amount of work is involved in the integration?
    - How long should the old and new system live in parallel?
    - How strictly should the new data model confirm to the old data model?

3. Risk reduction patterns

3.1 Limited release

- **What:** A small group of users are pilot users for the new system
- **Context:** The new and the old system can coexist.
- **New context:** Selected users provide feedback based on using the new system in their daily work.
- **Considerations:**
    - What use group is suited for the limited release?
    - For what functionality should they use the old system and the new system?
    - Should pilot users be allowed to switch between using the old and the new version?

3.2 Facilitate switching

- **What:** The new system can open the old system in the current application context. Alternatively, the old system can open the new system in the current context.
- **Context:** Some functionality is available in the new system while other functionality can still only be accessed from the old system.
- **New context:** The user can jump between using the old and the new system while performing a task. This may increase the risk of acting on stale data.
- **Considerations:**
    - Can the (old) technology support switching?
    - Should the old system be modified to start up the new system?
    - What is the meaning of "current" view that both systems share?

3.3 Update downstream first

- **What:** A client system is upgraded to support the old and new version of a server system in preparation of the server system being changed or replaced. One of several patterns where multiple versions of an interface is supported concurrently.
- **Context:** An upstream system will be changed, but the timescale for this is somewhat uncertain.
- **New context:** The upstream system can be upgraded or an upgrade can be reverted at will without affecting the new system.
- **Considerations:**
    - How long will support of multiple versions of the interface be necessary?
    - How will data be kept synchronized between the two versions?