SSW-540: Fundamentals of Software Engineering

Software Development Processes

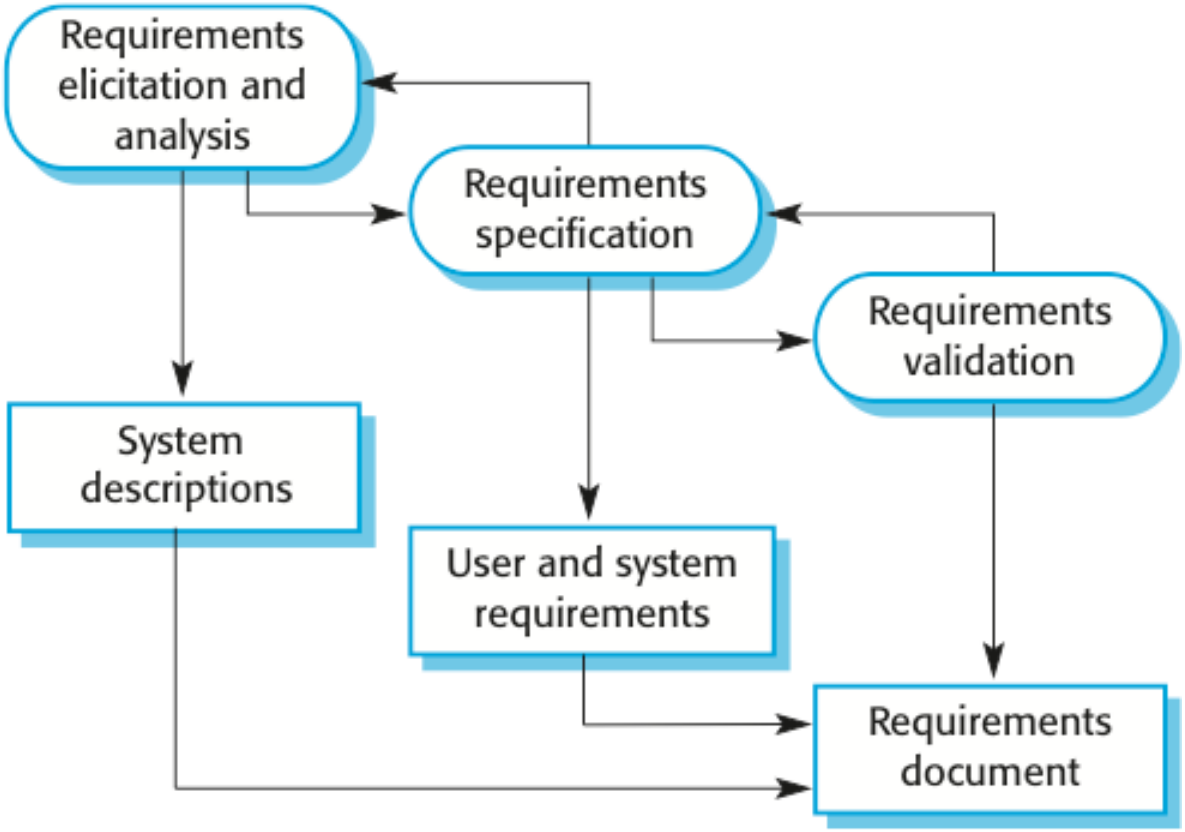
Dr. Richard Ens
School of Systems and Enterprises

Software Development Processes

Process Models – abstract representations

- Fundamental activities
  - Specification
  - Development
  - Validation
  - Evolution
- Each has *deliverables*
- Process models capture an abstract view of the activities, the deliverables, their sequencing and interdependencies

What must these processes do? Specification…

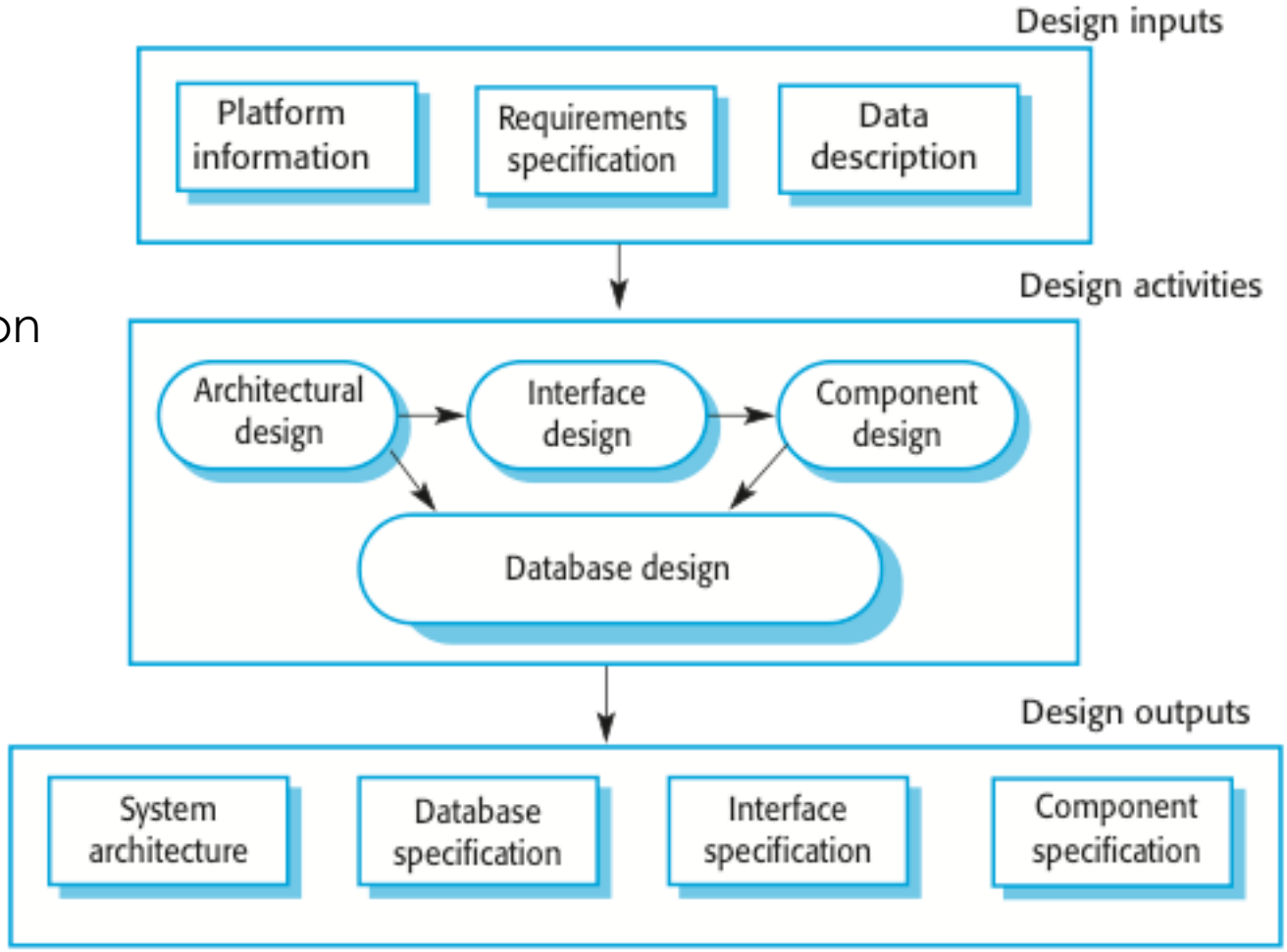


Specification is the process of establishing

- what services are required
- the constraints on the system's operation and development.

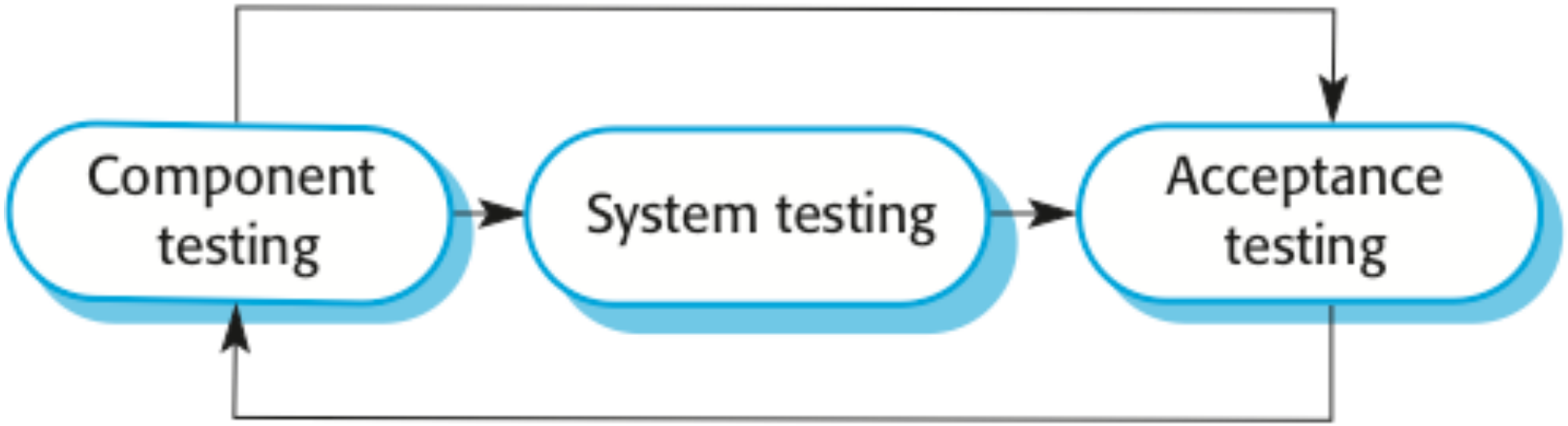
What must these processes do? Development…

Development is the process of converting the system specification into an executable system via design and implementation

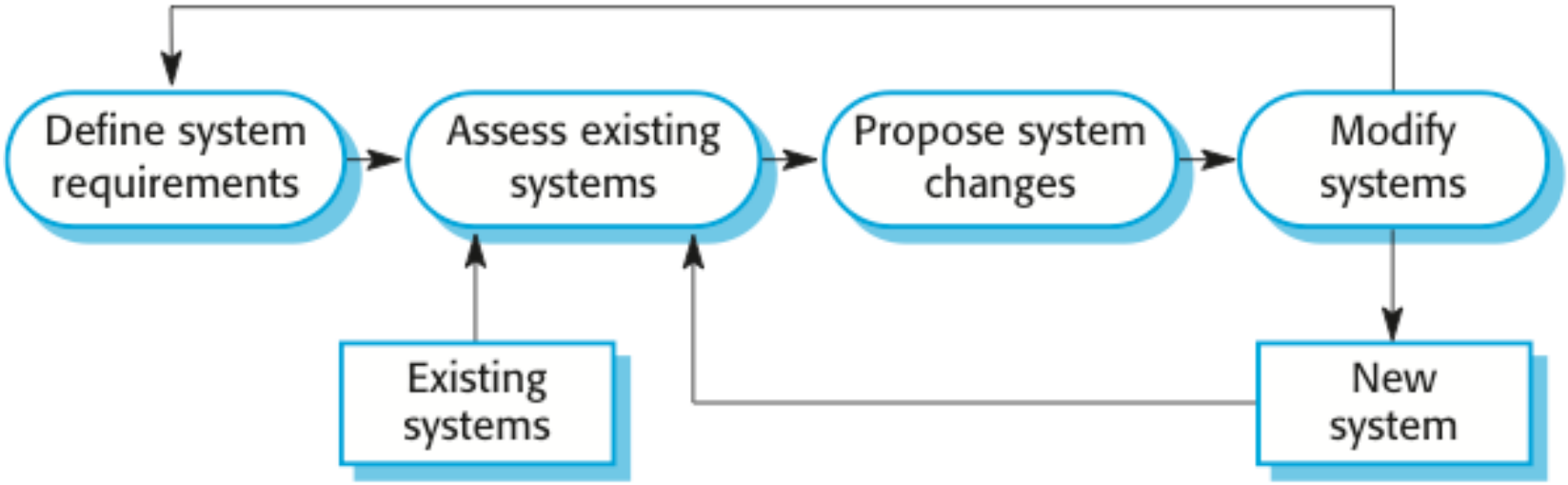


What must these processes do? Validation…

- Verification and validation is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Testing is the most commonly used V&V activity.



What must these processes do? Evolution…



- Software is inherently flexible
- As "business" circumstances change, software supporting the "business" must evolve and change.

Plan-driven and agile processes

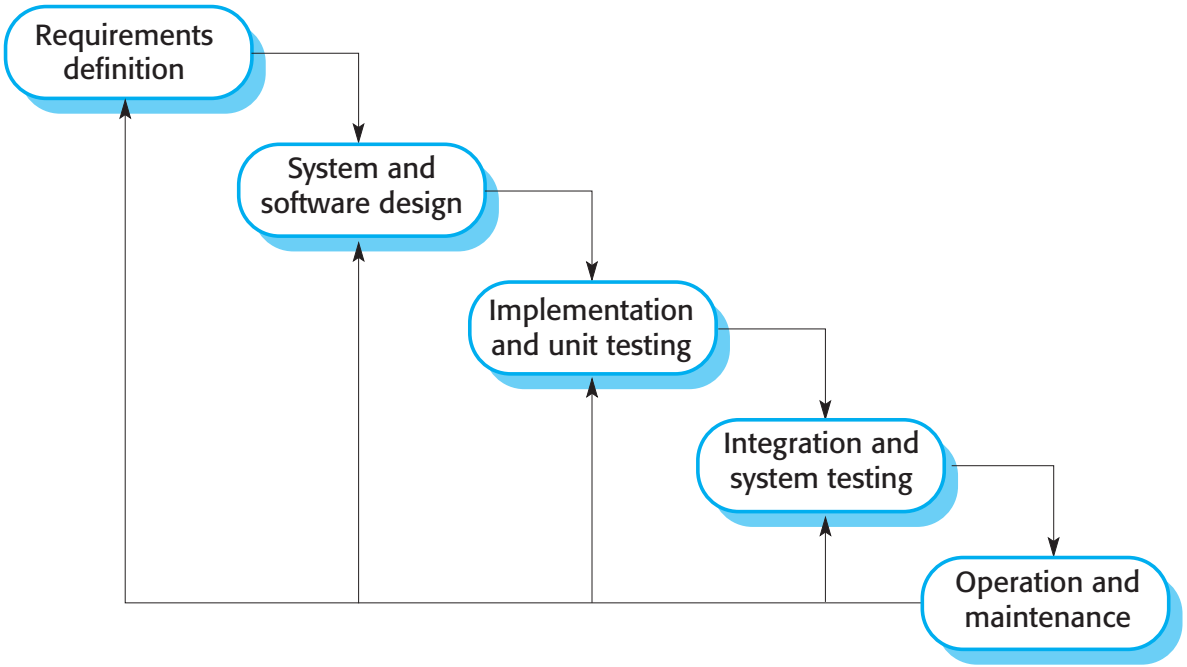
- Plan-driven processes are processes where all of the process activities, usually separate development stages with outputs, are planned in advance and progress is measured against this plan.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The Waterfall Model

✧ There are separate identified phases in the waterfall model:



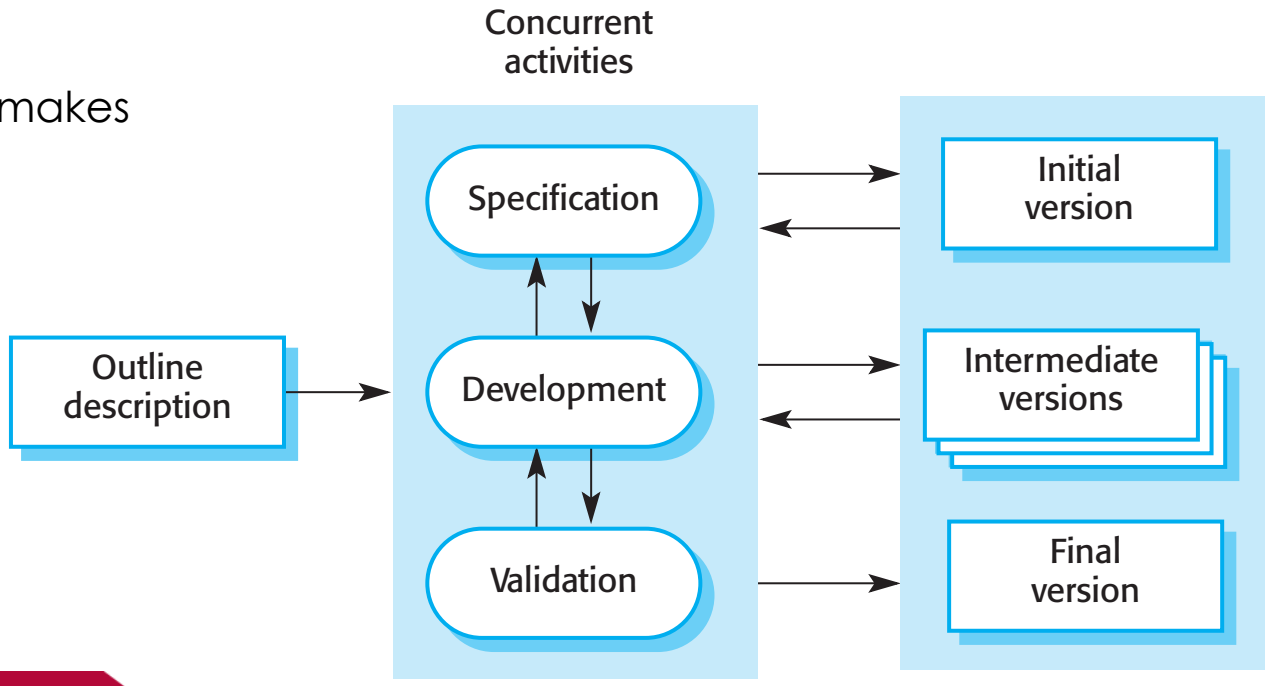
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

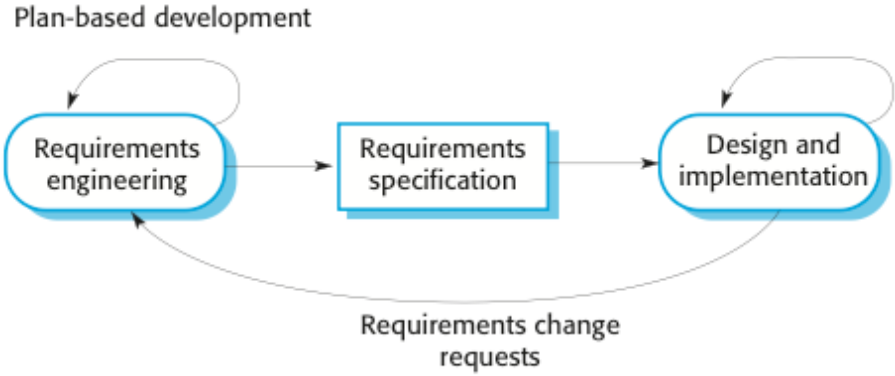
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - (Few business systems have stable requirements.)
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.
  - And the coincident documentation is useful to the large teams working on the project.

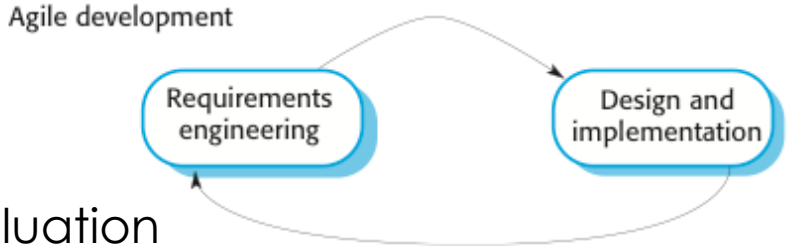
Incremental development

- Benefits
  - It's more efficient to fail "small".
  - The cost of accommodating changing customer requirements is reduced.
  - It is easier to get customer feedback on the development work that has been done.
  - More rapid delivery and deployment of useful software to the customer is possible.
- Downsides
  - Less documentation makes process less visible.
  - System structure degrades with added increments.



One incremental process: Agile development





- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- Agile methods reduce overheads in the software process (e.g. by limiting documentation) and aim to respond quickly to changing requirements without excessive rework.
- The Agile *manifesto* emphasizes the importance of working code and customer collaboration along with responsiveness

Agile method applicability

- Development of small or medium-sized products.
  - Many saleable software products and apps are now developed using an agile approach
- Custom system development within an organization
  - where there is a clear commitment from the customer to become involved in the development process and
  - where there are few external rules and regulations that affect the software.
- Agile requires a skilled and committed development team
  - The company culture needs to be supportive

Agile development techniques

- User stories expressing requirements
- Pair programming, collective ownership of code
- Test-driven development – testing after every code change
- Constant refactoring to maintain simplicity of the code
- Continuous integration
- Small system releases supporting frequent changes
- Extensive customer involvement
  - User stories and their prioritization
  - Test development and validation
  - Release acceptance and feedback

Watch the video on Agile

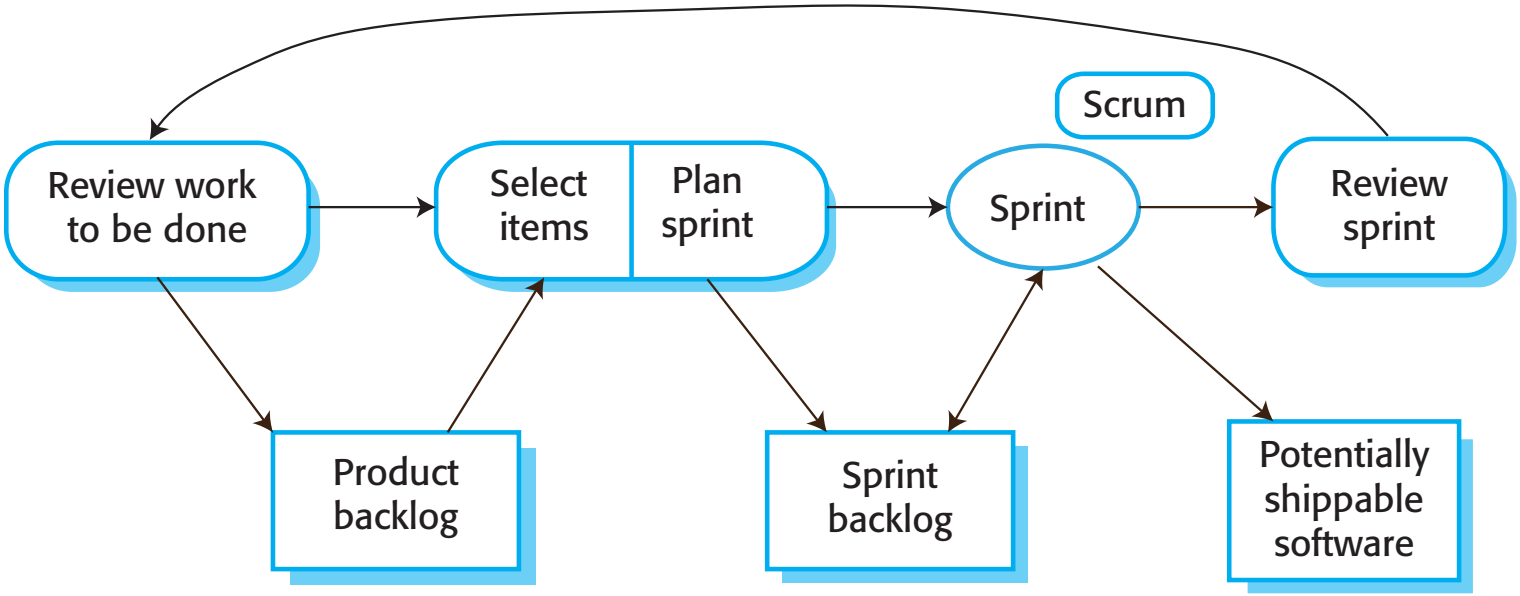
- Remember, "*if you don't know where you are going, being agile won't help*".

Agile project management

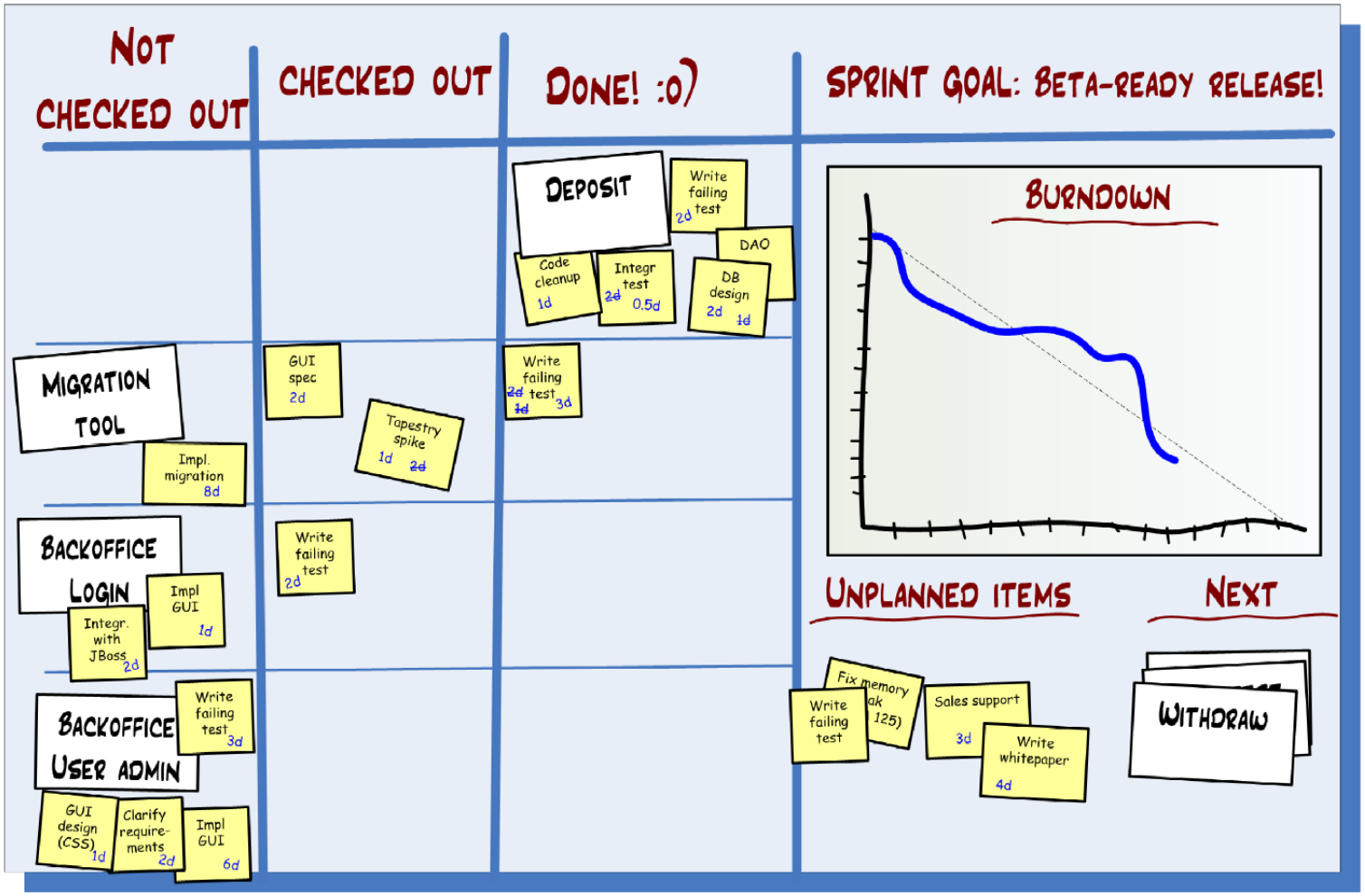
- Software project managers manage the project so that the software is delivered on time and within the planned budget.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.
- Scrum is an agile method focused on managing iterative development

Scrum

- Three phases
  - An outline planning phase establishing general objectives for the product and the design of the software architecture.
  - A series of fixed-length sprint cycles, which each cycle develops an increment of the system.
  - A project closure phase to wrap up the project, complete any required documentation (e.g., system help frames or user manuals), and assess the lessons learned from the project.



Scrum project management system



Source: Henrik Kniberg, ***Scrum and XP from the Trenches***, C4Media, 2007.

Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Issues with scaling agile methods

- Large systems
  - are often collections of separate, communicating systems, where separate teams, working in different places, develop each system.
  - are 'brownfield systems; they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.
  - where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development
- Large system development
  - Their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
  - Large systems have a long procurement and development time so it is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
  - Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

Scaling agile methods

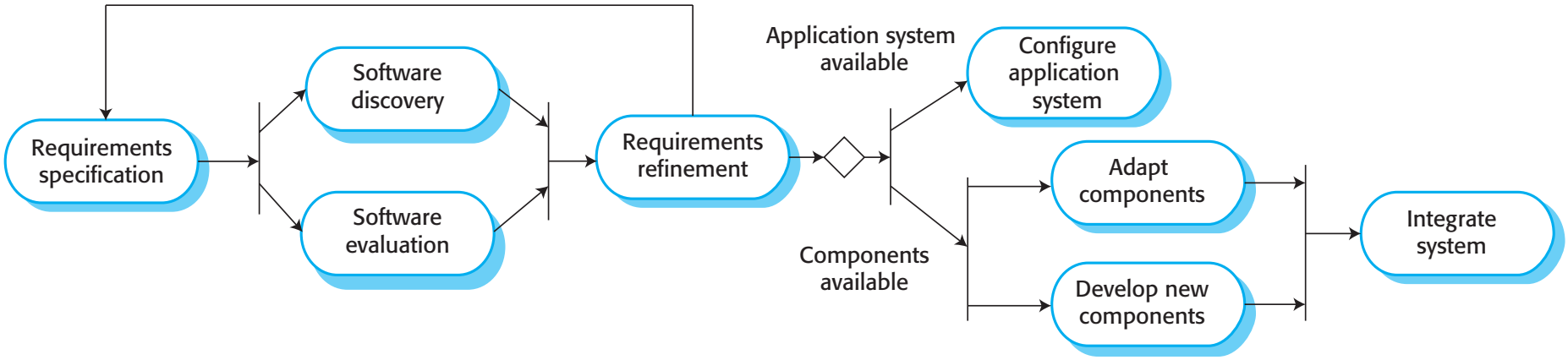
- Keeping in mind that scaling up means:
  - A completely incremental approach to requirements engineering is impossible.
  - There cannot be a single product owner or customer representative.
  - For large systems development, it is not possible to focus only on the code of the system.
  - Cross-team communication mechanisms have to be designed and used.
  - Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system.
- To scale agile up for use development large software systems, some plan-based practices must be integrated:
  - Up-front requirements
  - Multiple customer representatives
  - More documentation
  - Common tooling across development teams
  - Alignment of releases across teams

One solution: Multi-team Scrum

- Role replication
  - Each team has a Product Owner for their work component and a ScrumMaster.
- Product architects
  - Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture.
- Release alignment
  - The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
- Scrum of Scrums
  - There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.

Remember that many practical development methods are a mixture of plan-based and agile development.

Reuse-oriented software engineering



- Systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems).
- Reused elements may be configured to adapt their behavior and functionality to a user's requirement
- Open sourcing encourages re-use of capabilities and code
- Reuse heavily used in business systems

Advantages and disadvantages of reuse

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- Containers for software on virtual machines (a la Docker) promote re-use
- But requirements compromises are inevitable so system may not meet real needs of users
- When reusable software does only part of the job, effort to understand how to extend it can be considerable
- Loss of control over evolution of reused system elements

Coping with change

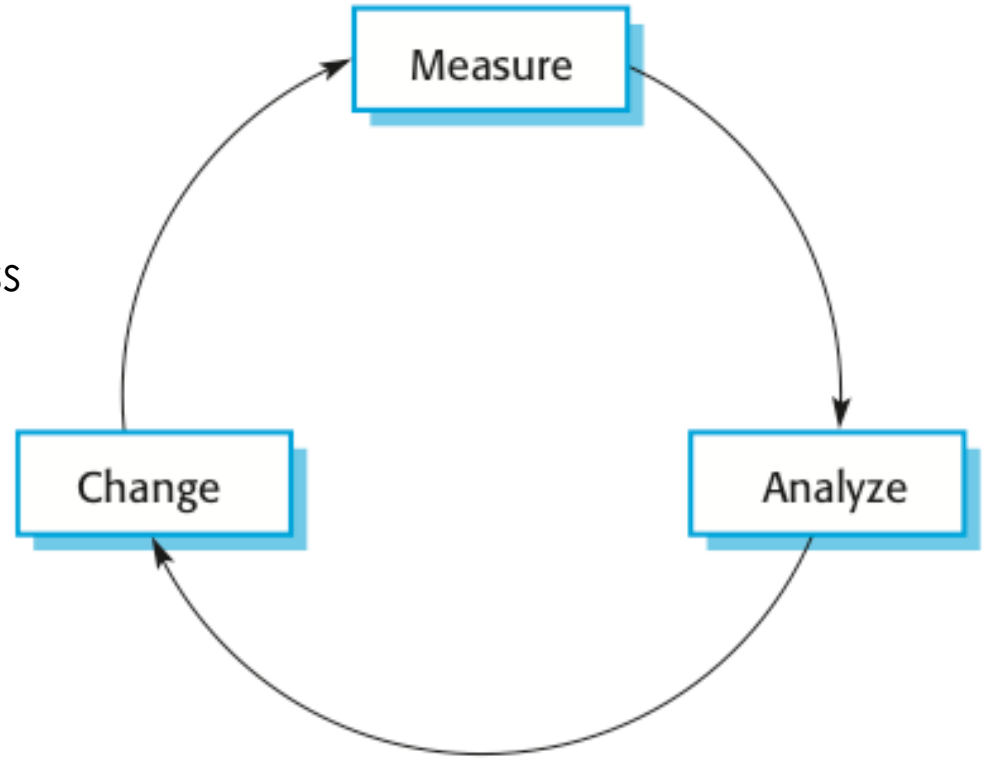
- In all large software projects (and most small ones), change is inevitable.
  - System requirements change in response to changes in the "business"
  - New technologies provide new possibilities for improved implementations
  - Software changes as platforms (hardware, lower-level software) change
- Incremental development processes can accommodate some change
- Software prototyping can help anticipate change
  - A prototype is an initial version of a system used to demonstrate concepts and try out design options.
  - A prototype can be used in:
    - The requirements engineering process to help with requirements elicitation and validation;
    - In design processes to explore options and develop a UI design;
    - In the testing process to run back-to-back tests.

Two types of software prototypes

- The *Throw-away* prototype, designed to, e.g.,
  - help with requirements elicitation and validation;
  - explore design options or develop a UI design;
  - advance the testing process by running back-to-back tests;
  - And then be discarded after development due to its lack of solid foundation, documentation and/or quality
- The *Evolutionary* prototype, designed as
  - the probable basis for the actual implementation
  - perhaps, even, the first increment in an incremental development process.
- The *throw-way* is used when the requirements are not well understood; the *evolutionary* prototype is used when the technology is yet untested, but the requirements are very well understood.

No view of processes is complete without process improvement

- Any and all processes can be improved
- Improvements to the software process can produce
  - Higher software quality
  - Lower development costs
  - Reduced development time
- The *process* of software process improvement involves
  - Measurement
  - Analysis
  - Process change



Process measurement

- Wherever possible, quantitative process data should be collected
  - Without clearly defined process standards, you don't know what to measure.
  - A process may have to be defined before any measurement is possible.
- Some process metrics:
  - Time taken for process activities to be completed; e.g. calendar time or effort to complete an activity or process.
  - Resources required for processes or activities, e.g. total effort in person-days.
  - Number of occurrences of a particular event, e.g. Number of defects discovered.
- Process measurements should be used to assess process improvements
- But measurements should not drive the improvements. The improvement driver should be the organizational objectives.

Classroom Activity – within your EMSS project teams

Activity:

Within your team, scope out a high level of view of Energy Management Software System's (EMSS) problem statement and, perhaps, a high-level concept of operation. Determine the likely fit of the system's development to one of the Software life-cycle processes discussed this week.

Readout to the class should be your problem statement (or concept of operations). Selecting the best software development process is dependent on having a good view of the problem you are solving!

Submission to complete the Canvas Assignment EMSS Week 2:

Document your problem statement and/or concept of operation as well as your choice of software life-cycle process and discuss the pros and cons your team discovered in arriving at that process choice.

Python “Pearls” from week 1

Python 3 uses **input()** instead of **raw_input()**.
Python 3 uses **print()** instead of **print**.

In Python 2.7:

```
Fname = raw_input (“Please enter your first name: “)
print “Hello”, fname
```

In Python 3:

```
Fname = input (“Please enter your first name: “)
print (“Hello”, fname)
```

Note, both deal entirely with strings and concatenation is done *automatically* by the **print** and **print()** lines.