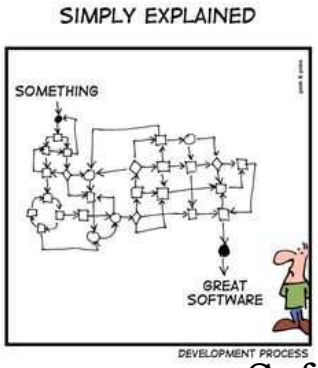


Software Development Process

www.cs.uoi.gr/~zarras/http://www.cs.uoi.gr/~zarras/se.htm

Slides material sources:
Software Engineering - Theory & Practice, S. L. Pfleeger
Introduction to Software Engineering, I. Sommerville
SWEBOK v3: IEEE Software Engineering Body of Knowledge

Software engineering fundamentals

What is software?

What is software?

Computer programs and associated documentation.

Software products may be developed for a particular customer or may be developed for a general market.

What is software engineering?

What is software engineering?

ISO/IEC/IEEE Systems and Software Engineering Vocabulary (SEVOCAB) defines **software engineering** as

"the application of a systematic, disciplined, quantifiable **approach** to the development, operation, and maintenance of software;

that is, **the application of engineering to software**"

Development process fundamentals

What is a process?

What is a process ?

A **process** is a series of steps involving activities that take some input, constraints, and resources to produce an intended output of some kind.

A process involves a set of tools and techniques.

What is a software development process?

What is a software development process model ?

A **structured set of activities** required to develop a software system.

There are many different software processes but all involve these key activities:

Requirements specification – defining what the system should do.

Design and implementation – defining the organization of the system and implementing the system.

Verification & Validation – checking if it works correctly and if it does what the customer wants.

Evolution – changing the system in response to changing customer needs.

What is a software development process model?

What is a development process model?

A software **process model** is a abstract representation of a process.

It presents a description of a process from some particular perspective.

A process description may include:

- Activities that constitute the process.
- Products, which are the outcomes of a process activity;
- Roles, which reflect the responsibilities of the people involved in the process;
- Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

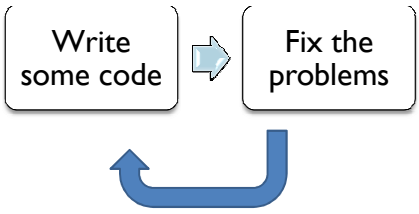
It is usually given in some notation like a simple directed graph with nodes & edges, a UML activity diagram, a BPMN process model, a Gantt chart.... and many more ...

Well known software development processes … a historical retrospection

How do we develop small programs for internal operation/personal use?

The code & fix model

In the introduction of the Spiral model Boehm refers to the basic model used in the earliest days of software.

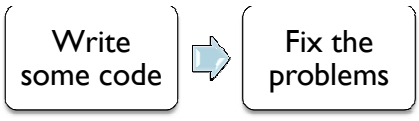


What is wrong with that?

The code & fix model

After a number of fixes, subsequent fixes may become very expensive.

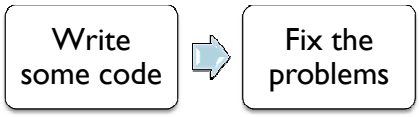
Frequently, the software becomes a poor match to users' needs that it is rejected or redeveloped.



The code & fix model

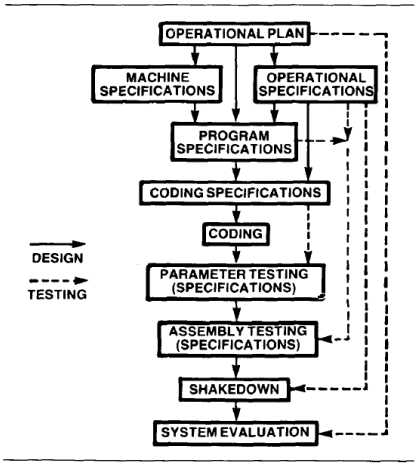
Code is expensive to fix because of poor preparation for testing and modification.

Lack of requirements analysis and design.



How do we develop larger programs for delivery to a customer?

The stage wise model



Program production. Production of a large-program system proceeds from a general operational plan through system evaluation; for example, assembly testing verifies operational and program specifications.

H.D. Benington, Production of Large Computer Programs, In ONR Symp. Advanced Programming Methods for Digital Computers, pp. 15-27, 1956.

Also available in Annals of the History of Computing, pp. 350-361, 1983 and in Proc. 9th Int'l Conf. Software Engineering (ICSE), 1987.

The waterfall model

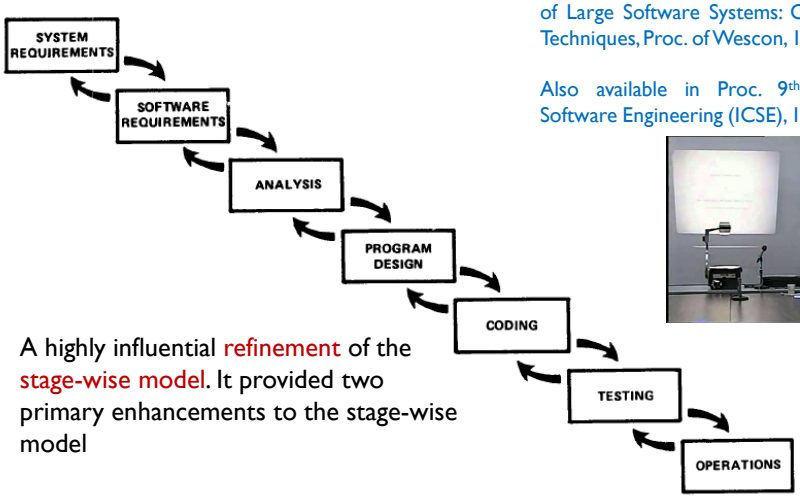


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

A highly influential refinement of the stage-wise model. It provided two primary enhancements to the stage-wise model

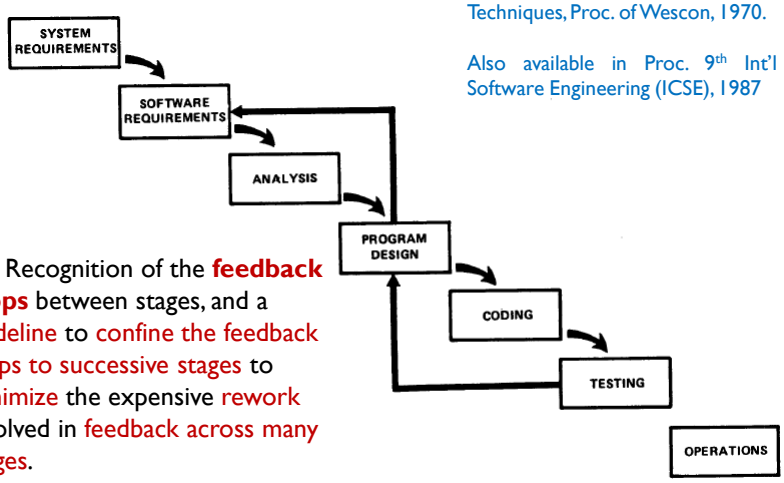
W. W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proc. of Wescon, 1970.

Also available in Proc. 9th Int'l Conf. Software Engineering (ICSE), 1987

The waterfall model

W. W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proc. of Wescon, 1970.

Also available in Proc. 9th Int'l Conf. Software Engineering (ICSE), 1987



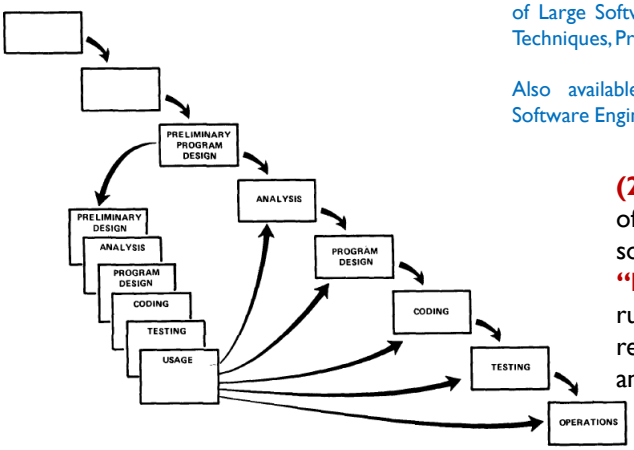
(1) Recognition of the **feedback loops** between stages, and a guideline to confine the feedback loops to successive stages to minimize the expensive rework involved in feedback across many stages.

Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps.

The waterfall model

W. W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proc. of Wescon, 1970.

Also available in Proc. 9th Int'l Conf. Software Engineering (ICSE), 1987



(2) An initial incorporation of **prototyping** in the software life cycle, via a **"build it twice"** step running in parallel with requirements analysis and design.

The waterfall model

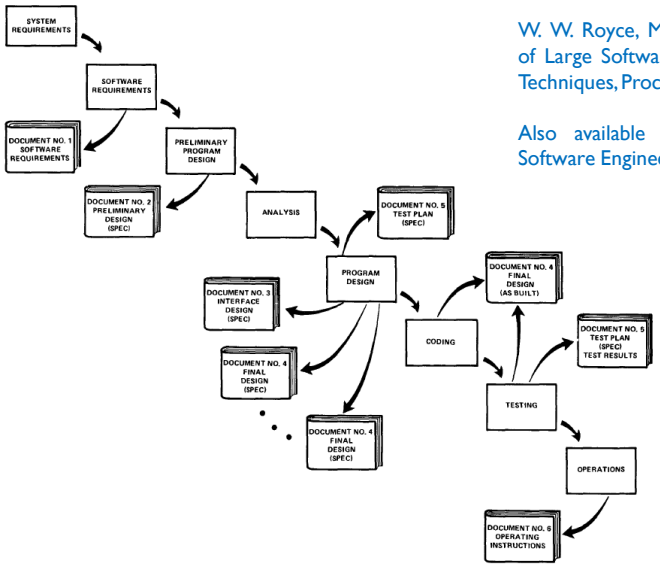


Figure 6. Step 2: Insure that documentation is current and complete – at least six uniquely different documents are required.

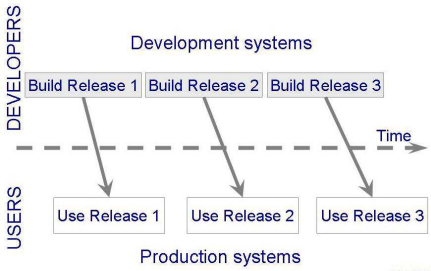
W. W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proc. of Wescon, 1970.

Also available in Proc. 9th Int'l Conf. Software Engineering (ICSE), 1987

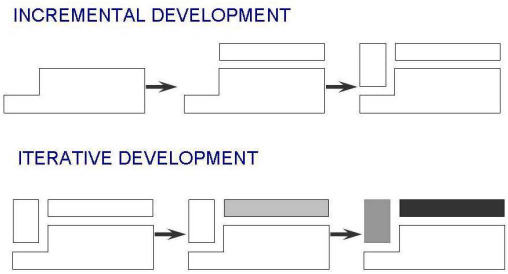
Document the design – At least 6 documents maintained during the whole process.

Ok, but how do we deal with software evolution?

The evolutionary development model



McCracken and Jackson, Life-Cycle Concept Considered Harmful, ACM Software Engineering Notes, pp. 29-32, 1982.



Ok, but how about risks and alternatives?

The spiral model

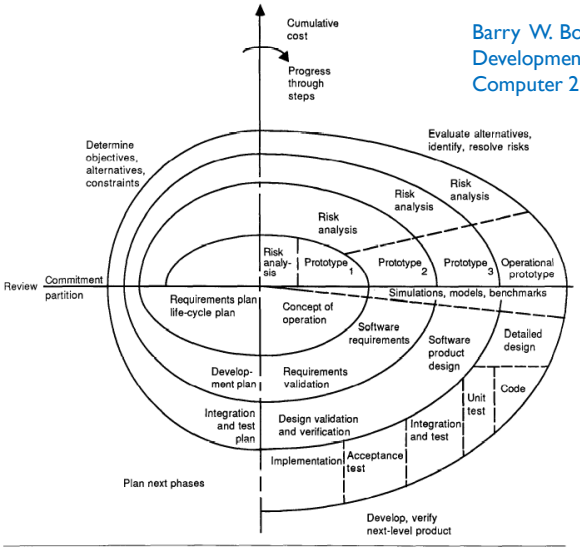


Figure 2. Spiral model of the software process.

Barry W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21(5), pp. 61-72, 1988.

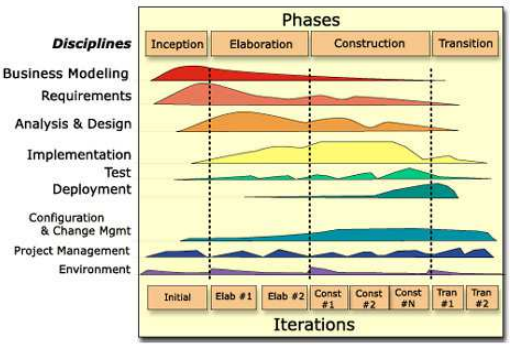
Table 4. A prioritized top-ten list of software risk items.

Risk item	Risk management techniques
1. Personnel shortfalls	Staffing with top talent, job matching; teambuilding; morale building; cross-training; pre-scheduling key people
2. Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)
5. Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
10. Straining computer-science capabilities	Technical analysis; cost-benefit analysis; prototyping; reference checking

Barry W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21(5), pp. 61-72, 1988.

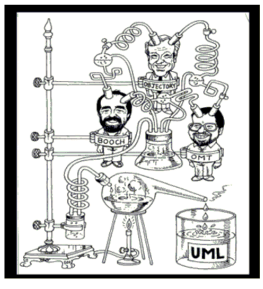
How about more recent approaches?

The Rational Unified Process (RUP)

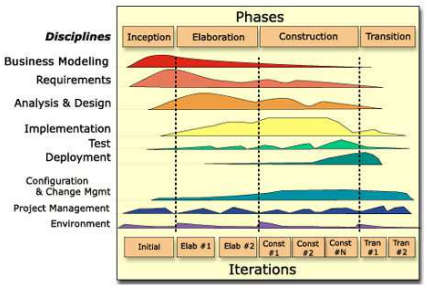


A modern iterative process derived from the work on the UML and associated process.

P. Kruchten, The Rational Unified Process- An Introduction, Addison-Wesley, 1998; 3rd ed. in 2003



The Rational Unified Process (RUP)

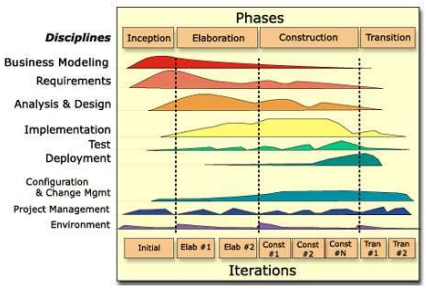


During the **inception phase**, you establish the business case for the system and delimit the project scope.

The outcome is:

- An initial business case, which includes business context, values propositions, target customers, success criteria, partner network, customer network, revenue model, cost model….
- A vision document: core requirements, key features, and main constraints.
- A initial use-case model (10% -20%) complete).
- An initial project glossary (may optionally be partially expressed as a domain model).
- An initial risk assessment.
- A project plan, showing phases and iterations.
- One or several prototypes.

The Rational Unified Process (RUP)

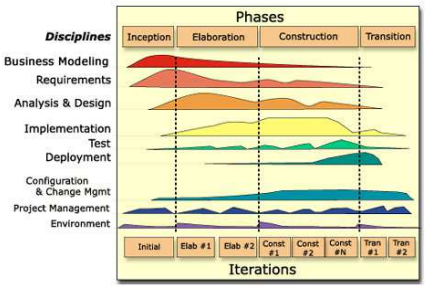


The purpose of the **elaboration phase** is to analyze the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project.

The outcome is:

- A use-case model (at least 80% complete).
- Supplementary requirements capturing the non functional requirements and any requirements that are not associated with a specific use case.
- A software architecture description and prototype.
- A revised risk list and a revised business case.
- A development plan for the overall project, including evaluation criteria for each iteration.
- A preliminary user manual (optional).

The Rational Unified Process (RUP)

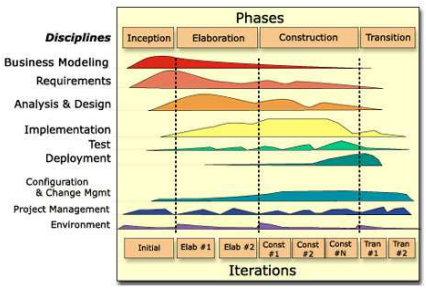


During the **construction phase**, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested.

At minimum, the outcome is:

- The software product integrated on the adequate platforms.
- The user manuals.
- A description of the current release.

The Rational Unified Process (RUP)

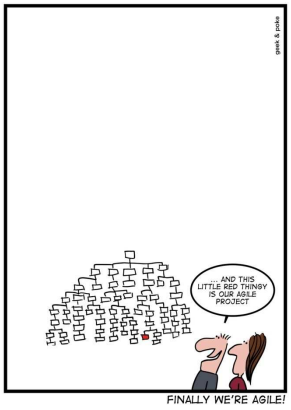


The purpose of the **transition phase** is to transition the software product to the user community.

Once the product has been given to the end user, issues usually arise that require you to develop new releases, correct some problems, or finish the features that were postponed.

How about more flexible approaches?

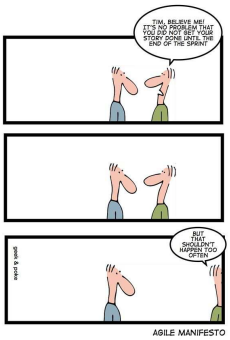
Agile methods



Dissatisfaction with the overheads involved in conventional processes of the 1980s and 1990s led to the creation of **agile methods**.

The aim of agile methods is to **reduce overheads** in the software process (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

Agile manifesto



We have come to value:

• Individuals and interactions	over	processes and tools
• Working software	over	comprehensive documentation
• Customer collaboration	over	contract negotiation
• Responding to change	over	following a plan

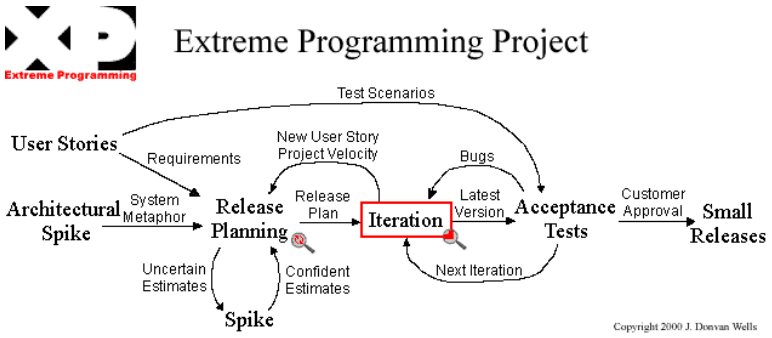
That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

What is eXtreme Programming (XP)?

XP model

http://www.extremeprogramming.org/



The Rules of Extreme Programming

Planning

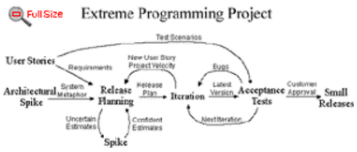
- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.



Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

Extreme Programming

User Stories

User stories serve the same purpose as use cases but are not the same. They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document. User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface. They are in the format of about three sentences of text written by the customer in the customers terminology without techno-syntax.

User stories also drive the creation of the acceptance tests. One or more automated acceptance tests must be created to verify the user story has been correctly implemented.

One of the biggest misunderstandings with user stories is how they differ from traditional requirements specifications. The biggest difference is in the level of detail. User stories should only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement. When the time comes to implement the story developers will go to the customer and receive a detailed description of the requirements face to face.

FullSize Extreme Programming Project

Developers estimate how long the stories might take to implement. Each story will get a 1, 2 or 3 week estimate in "ideal development time". This ideal development time is how long it would take to implement the story in code if there were no distractions, no other assignments, and you knew exactly what to do. Longer than 3 weeks means you need to break the story down further. Less than 1 week and you are at too detailed a level, combine some stories. About 80 user stories plus or minus 20 is a perfect number to create a release plan during release planning.

Another difference between stories and a requirements document is a focus on user needs. You should try to avoid details of specific technology, data base layout, and algorithms. You should try to keep stories focused on user needs and benefits as opposed to specifying GUI layouts.

What is Scrum?

Scrum

https://www.scrumguides.org/

Scrum is a process framework that has been used to manage work on complex products since the early 1990s.

Scrum is not a process, technique, or definitive method. Rather, it is a framework within which you can employ various processes and techniques.

Ken Schwaber and Jeff Sutherland

Scrum Team

https://www.scrumguides.org/

The Product Owner is the person responsible for managing the Product Backlog. Product Backlog management includes:

1. Clearly expressing Product Backlog items;
2. Ordering the items in the Product Backlog to best achieve goals and missions;
3. Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next;
4. Ensuring the Development Team understands items in the Product Backlog to the level needed.

Scrum Team

https://www.scrumguides.org/

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.

A "Done" increment is required at the Sprint Review. Only members of the Development Team create the Increment.

<= 3 Size of team < 9

Scrum Team

https://www.scrumguides.org/

The Scrum Master is responsible for promoting and supporting Scrum as defined in the Scrum Guide.

Scrum Masters do this by helping everyone understand Scrum theory, practices, rules, and values.

Scrum Events

https://www.scrumguides.org/

The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort.

A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint: **No changes** are made that would endanger the Sprint Goal; Quality goals do not decrease; and, Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Scrum Events

https://www.scrumguides.org/

Sprint Planning

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

Sprint Planning answers the following:

1. What can be delivered in the Increment resulting from the upcoming Sprint?
2. How will the work needed to deliver the Increment be achieved?

What => select from Program Backlog

How => designing the product

Scrum Events

https://www.scrumguides.org/

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team. The Daily Scrum is held every day of the Sprint.

Here is an example of what might be used:

What did I do yesterday that helped the Development Team meet the Sprint Goal?

What will I do today to help the Development Team meet the Sprint Goal?

Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

Scrum Events

https://www.scrumguides.org/

Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.

During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value.

This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration. This is at most a four-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter

Scrum Events

https://www.scrumguides.org/

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

1. Inspect how the last Sprint went with regards to people, relationships, process, and tools;
2. Identify and order the major items that went well and potential improvements;
3. Create a plan for implementing improvements to the way the Scrum Team does its work.

Scrum Artefacts

https://www.scrumguides.org/

Product Backlog

The Product Backlog is an ordered list of everything that is known to be needed in the product.

The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

A Product Backlog is never complete.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items often include test descriptions

Scrum Artefacts

https://www.scrumguides.org/

Sprint Backlog

The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal.

The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a "Done" Increment.

As work is performed or completed, the estimated remaining work is updated.