Quality Assurance Techniques for Software Dev
Textbook: https://plus.pearson.com/
1.1: Product Vision
-Product Vision (starting point)
-A statement on the essence of the software you are trying to develop.
1. *What* is the product that you propose to develop? What makes this product different from competing products?
2. *Who* are the target users and customers for the product?
3. *Why* should customers buy this product?

-Moore's vision template
- FOR (target customer)
- WHO (statement of the need or opportunity)
- The (PRODUCT NAME) is a (product category)
- THAT (key benefit, compelling reason to buy)
- UNLIKE (primary competitive alternative)
- OUR PRODUCT (statement of primary differentiation)

-Ex:

> *FOR a mid-sized company's marketing and sales departments WHO need basic CRM functionality, THE CRM-Innovator is a Web-based service THAT provides sales tracking, lead generation, and sales representative support features that improve customer relationships at critical touch points. UNLIKE other services or package software products, OUR product provides very capable services at a moderate cost.*

-Usually, this vision doesn't come from a eureka moment, but lots of bruh moments of refinement.
-Table of sources that help create product visions.

Information source	Explanation
Domain experience	The product developers may work in a particular area (say, marketing and sales) and understand the software support that they need. They may be frustrated by the deficiencies in the software they use and see opportunities for an improved system.
Product experience	Users of existing software (such as word processing software) may see simpler and better ways of providing comparable functionality and propose a new system that implements this. New products can take advantage of recent technological developments such as speech interfaces.
Customer experience	The software developers may have extensive discussions with prospective customers of the product to understand the problems that they face; constraints, such as interoperability, that limit their flexibility to buy new software; and critical attributes of the software that they need.
Prototyping and "playing around"	Developers may have an idea for software but need to develop a better understanding of that idea and what might be involved in developing it into a product. They may develop a prototype system as an experiment and "play around" with ideas and variations using that prototype system as a platform.

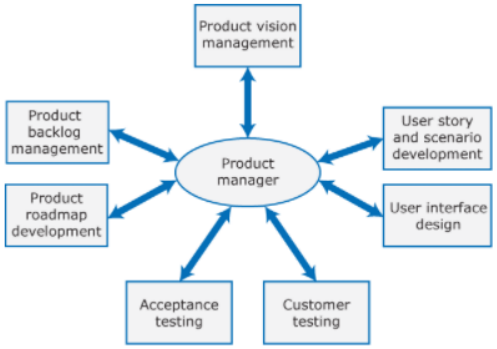
-Author gives ex of ILearn, where made had a product vision from the shortcomings of prior software.

> *FOR* teachers and educators *WHO* need a way to help students use web-based learning resources and applications, *THE iLearn system* is an open learning environment *THAT* allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves.
>
> *UNLIKE* Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process rather than the administration and management of materials, assessments, and coursework. *OUR* product enables teachers to create subject and age-specific environments for their students using any web-based resources, such as videos, simulations, and written materials that are appropriate.
>
> Schools and universities are the target customers for *the iLearn system* as it will significantly improve the learning experience of students at relatively low cost. It will collect and process learner analytics that will reduce the costs of progress tracking and reporting.

1.2: Software Product Management
-Product Managers (PM)
-Takes overall responsibility for planning, development, and marketing.
-The connection between dev team, broader organization, and customers.
-Can be a dedicated person (usually large company) or interchanged (usually small teams)

1. **Business needs** PMs have to ensure that the software being developed meets the business goals and objectives of both the software product company and its customers. They must communicate the concerns and needs of the customers and the development team to the managers of the product business. They work with senior managers and with marketing staff to plan a release schedule for the product.
2. **Technology constraints** PMs must make developers aware of technology issues that are important to customers. These may affect the schedule, cost, and functionality of the product that is being developed.
3. **Customer experience** PMs should be in regular communication with customers to understand what they are looking for in a product, the types of user and their backgrounds, and the ways in which the product may be used. Their experience of customer capabilities is a critical input to the design of the product's user interface. PMs may also involve customers in alpha and beta product testing.

-How the PM interacts with dev team



-PM & Production Vision
    -Keeps the team from drifting away from the product vision.
    -Decides if suggested changes align with the product vision.
-PM & Roadmap Development
    -Milestones & Release Date, PM must keep track of dev team work.
-PM & User story and scenario development
    -PM keeps track of what user what the software to do and tries to get dev team to implement
-PM & Product backlog management
    -The to-do list, PM must manage it.
-PM & Acceptance testing
    -PM must check if software meets the goals of the roadmap. Can be done by test scenarios.
-PM & UI design
    -PM acts like surrogate users to test UI features.

1.3: Product Prototyping

-Prototype
    -Product Vision usually isn't enough to convince backers, you need an example of it.
    -Helps know which feature needs focus and how much work it takes.
    -A prototype should be made ASAP in order to demonstrate features and see if the product vision is worth committing.
    -Should be considered a "throw-away" because the final version is better from scratch.

-2 Stages/Uses of a Prototype

1. **Feasibility demonstration** You create an executable system that demonstrates the new ideas in your product. The goals at this stage are to see whether your ideas actually work and to show funders and company management that your product features are better than those of competitors.
2. **Customer demonstration** You take an existing prototype created to demonstrate feasibility and extend it with your ideas for specific customer features and how these can be realized. Before you develop a customer prototype, you need to do some user studies and have a clear idea of your potential users and scenarios of use. I explain how to develop user personas and usage scenarios in **Chapter 3**.

Chapter 1 Key Points

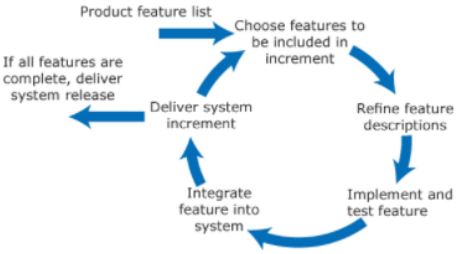
- Software products are software systems that include general functionality that is likely to be useful to a wide range of customers.
- In product-based software engineering, the same company is responsible for deciding on both the features that should be part of the product and the implementation of these features.
- Software products may be delivered as stand-alone products running on the customer's computers, hybrid products, or service-based products. In hybrid products, some features are implemented locally and others are accessed from the Internet. All features are remotely accessed in service-based products.
- A product vision succinctly describes what is to be developed, who are the target customers for the product, and why customers should buy the product you are developing.
- Domain experience, product experience, customer experience, and an experimental software prototype may all contribute to the development of the product vision.
- Key responsibilities of product managers are to own the product vision, develop a product roadmap, create user stories and scenarios, manage the product backlog, conduct customer and acceptance testing, and design the user interface.
- Product managers work at the interface between the business, the software development team, and the product customers. They facilitate communication among these groups.
- You should always develop a product prototype to refine your own ideas and to demonstrate the planned product features to potential customers.

2.1 Agile Methods

-The Agile Manifesto

> We are uncovering better ways of developing software by doing it and helping others to do it. Through this work, 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.
>
> While there is value on the items on the right, we value the items on the left more.

-Came from software devs in the 80s and 90s doing rigorous long planning and for one-off systems.
    -Software devs had to rewrite their code when the systems they were writing it for change.
-A subpart of Agile Dev. is Incremental Development, it's better to delay working on features until it is needed.
    -Don't worry about the detail and describe the feature first until you need to implement it.



Activity	Description
Choose features to be included in an increment	Using the list of features in the planned product, select those features that can be implemented in the next product increment.
Refine feature descriptions	Add detail to the feature descriptions so that the team members have a common understanding of each feature and there is sufficient detail to begin implementation.
Implement and test	Implement the feature and develop automated tests for that feature that show that its behavior is consistent with its description. I explain automated testing in Chapter 9.
Integrate feature and test	Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.
Deliver system increment	Deliver the system increment to the customer or product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

(Table explains each step of cycle)

-Agile Dev then adds the involvement of customers and dev team.

Principle	Description
Involve the customer	Involve customers closely with the software development team. Their role is to provide and prioritize new system requirements and to evaluate each increment of the system.
Embrace change	Expect the features of the product and the details of these features to change as the development team and the product manager learn more about the product. Adapt the software to cope with changes as they are made.
Develop and deliver incrementally	Always develop software products in increments. Test and evaluate each increment as it is developed and feed back required changes to the development team.
Maintain simplicity	Focus on simplicity in both the software being developed and the development process. Wherever possible, do what you can to eliminate complexity from the system.
Focus on people, not the development process	Trust the development team and do not expect everyone to always do things in the same way. Team members should be left to develop their own ways of working without being limited by prescriptive software processes.

(Table are practices used by devs now)

2.2 Extreme Programming

-Extreme Programming XP

-A predecessor/subpart of Agile programming

-XP focused on new development techniques that were geared to rapid, incremental software development, change, and delivery.

Practice	Description
Incremental planning/ user stories	There is no "grand plan" for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative. The requirements are written as user stories. The stories to be included in a release are determined by the time available and their relative priority.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the previous release.
Test-driven development	Instead of writing code and then tests for that code, developers write the tests first. This helps clarify what the code should actually do and that there is always a "tested" version of the code available. An automated unit test framework is used to run the tests after every change. New code should not "break" code that has already been implemented.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system and a new version of the system is created. All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.
Refactoring	Refactoring means improving the structure, readability, efficiency, and security of a program. All developers are expected to refactor the code as soon as potential code improvements are found. This keeps the code simple and maintainable.

(Table are practices used by devs now)

-Simple Design can backfire as customers most likely don't know the complexity of the implementation.

2.3 Scrum

-What happens when things go according to plan?

-Like delays and losing a developer.

-Scrum is a proactive approach.

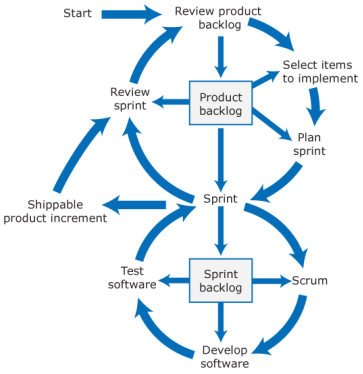
-Scrum Terms

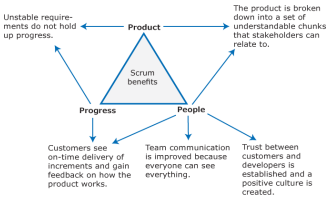
Scrum term	Explanation
Product	The software product that is being developed by the Scrum team.
Product Owner	A team member who is responsible for identifying product features and attributes. The Product Owner reviews work done and helps to test the product.
Product backlog	A to-do list of items such as bugs, features, and product improvements that the Scrum team has not yet completed.
Development team	A small self-organizing team of five to eight people who are responsible for developing the product.
Sprint	A short period, typically two to four weeks, when a product increment is developed.
Scrum	A daily team meeting where progress is reviewed and work to be done that day is discussed and agreed.
ScrumMaster	A team coach who guides the team in the effective use of Scrum.
Potentially shippable product increment	The output of a sprint that is of high enough quality to be deployed for customer use.
Velocity	An estimate of how much work a team can do in a single sprint.

1. The *Product Owner* is responsible for ensuring that the development team always focuses on the product they are building rather than diverted to technically interesting but less relevant work. In product development, the product manager should normally take on the Product Owner role.
2. The *ScrumMaster* is a Scrum expert whose job is to guide the team in the effective use of the Scrum method. The developers of Scrum emphasize that the ScrumMaster is not a conventional project manager but is a coach for the team. The ScrumMaster has the authority within the team on how Scrum is used. However, in many companies that use Scrum, the ScrumMaster also has some project management responsibilities.

-Hopefully, each sprint should produce new ready to ship code, all done and tested.

-Scrum Cycle





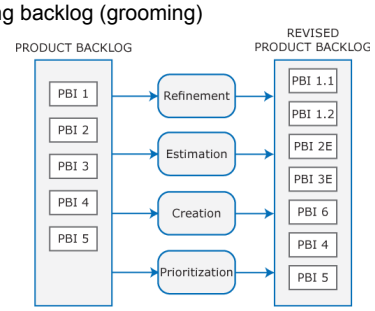
-Sprint has a todo backlog, that creates an incremental piece of the product, repeat until product is good.
-Sprint don't extend, incompleted features is placed back into product backlog

-Product Backlog
-List of to-do items

1. As a teacher, I want to be able to configure the group of tools that are available to individual classes. (feature)
2. As a parent, I want to be able to view my children's work and the assessments made by their teachers. (feature)
3. As a teacher of young children, I want a pictorial interface for children with limited reading ability. (user request)
4. Establish criteria for the assessment of open source software that might be used as a basis for parts of this system. (development activity)
5. Refactor user interface code to improve understandability and performance. (engineering improvement)
6. Implement encryption for all personal user data. (engineering improvement)

Heading	Description
Ready for consideration	These are high-level ideas and feature descriptions that will be considered for inclusion in the product. They are tentative so may radically change or may not be included in the final product.
Ready for refinement	The team has agreed that this is an important item that should be implemented as part of the current development. There is a reasonably clear definition of what is required. However, work is needed to understand and refine the item.
Ready for implementation	The PBI has enough detail for the team to estimate the effort involved and to implement the item. Dependencies on other items have been identified.

-Modifying backlog (grooming)

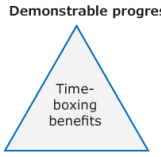


1. **Refinement** Existing PBIs are analyzed and refined to create more detailed PBIs. This may also lead to the creation of new backlog items.
2. **Estimation** The team estimates the amount of work required to implement a PBI and adds this assessment to each analyzed PBI.
3. **Creation** New items are added to the backlog. These may be new features suggested by the product manager, required feature changes, engineering improvements, or process activities such as the assessment of development tools that might be used.
4. **Prioritization** The PBIs are reordered to take new information and changed circumstances into account.

1. **Effort required** The amount of effort may be expressed in person-hours or person-days—that is, the number of hours or days it would take one person to implement that PBI. This is not the same as calendar time. Several people may work on an item, which may shorten the calendar time required. Alternatively, a developer may have other responsibilities that prevent full-time work on a project. Then the calendar time required is longer than the effort estimate.
2. **Story points** Story points are an arbitrary estimate of the effort involved in implementing a PBI, taking into account the size of the task, its complexity, the technology that may be required, and the "unknown" characteristics of the work. Story points were derived originally by comparing user stories, but they can be used for estimating any kind of PBI. Story points are estimated relatively. The team agrees on the story points for a baseline task. Other tasks are then estimated by comparison with this baseline—for example, more or less complex, larger or smaller, and so on. The advantage of story points is that they are more abstract than effort required because all story points should be the same, irrespective of individual abilities.

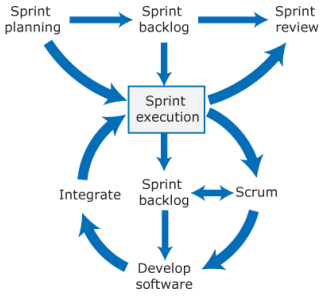
-Timeboxed Sprints
-Benefits: tangible code, problem discovery, planning insight

There is a tangible output (usually a software demonstrator) that can be delivered at the end of every sprint.



Problem discovery
If errors and omissions are discovered, the rework required is limited to the duration of a sprint.

Work planning
The team develops an understanding of how much work they can do in a fixed time period.



1. **Sprint planning** Work items to be completed during that sprint are selected and, if necessary, refined to create a sprint backlog. This should not last more than a day at the beginning of the sprint.
2. **Sprint execution** The team works to implement the sprint backlog items that have been chosen for that sprint. If it is impossible to complete all of the sprint backlog items, the time for the sprint is not extended. Rather, the unfinished items are returned to the product backlog and queued for a future sprint.
3. **Sprint reviewing** The work done during the sprint is reviewed by the team and (possibly) external stakeholders. The team reflects on what went well and what went wrong during the sprint, with a view to improving the work process.

-Sprint Planning steps

- agree on a sprint goal;
- decide on the list of items from the product backlog that should be implemented;
- create a sprint backlog, a more detailed version of the product backlog that records the work to be done during the sprint.

-Sprint Goals

Implement user roles so that users can select their role when they log in to the system.

Support
Develop analytics that maintain information about the time users spend using each feature of the system.

Performance and reliability
Ensure that the login response time is less than 10 seconds for all users where there are up to 2000 simultaneous login connections.

1. **Test automation** As far as possible, product testing should be automated. You should develop a suite of executable tests that can be run at any time. I explain how to do this in **Chapter 9**.
2. **Continuous integration** Whenever anyone makes changes to the software components they are developing, these components should be immediately integrated with other components to create a system. This system should then be tested to check for unanticipated component interaction problems. I explain continuous integration in **Chapter 10**.

-Scrum (aka meeting)

A scrum is a short, daily meeting that is usually held at the beginning of the day. During a scrum, all team members share information, describe their progress since the previous day's scrum, and present problems that have arisen and plans for the coming day. This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

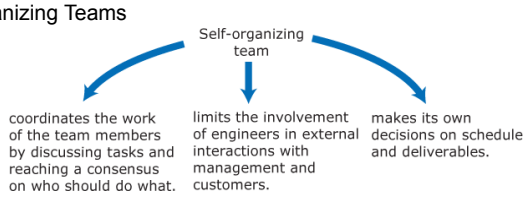
Scrum meetings should be short and focused. To dissuade team members from getting involved in long discussions, scrums are sometimes organized as "stand-up" meetings where there are no chairs in the meeting room.

During a scrum, the sprint backlog is reviewed. Completed items are removed from it. New items may be added to the backlog as new information emerges. The team then decides who should work on sprint backlog items that day.

-Levels of Code Completeness

State	Description
Reviewed	The code has been reviewed by another team member who has checked that it meets agreed coding standards, is understandable, includes appropriate comments, and has been refactored if necessary.
Unit tested	All unit tests have been run automatically and all tests have executed successfully.
Integrated	The code has been integrated with the project codebase and no integration errors have been reported.
Integration tested	All integration tests have been run automatically and all tests have been executed successfully.
Accepted	Acceptance tests have been run if appropriate and the Product Owner or the development team has confirmed that the product backlog item has been completed.

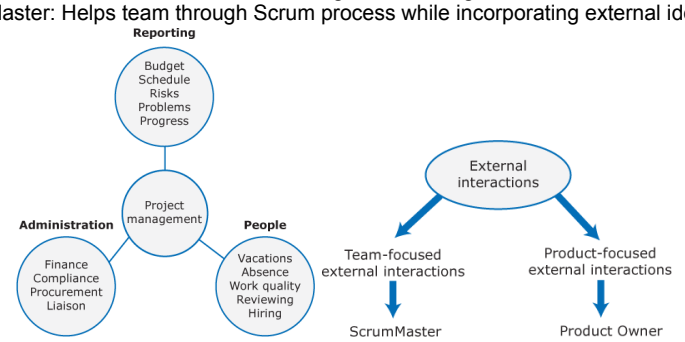
-Self-organizing Teams



1. Small companies may not have the resources to support dedicated project managers.
2. Many project management tasks require detailed knowledge of a team's work. If a project manager is working across several teams, it may be impossible to know the work of each team in detail.
3. Self-organizing teams are cohesive and tend to resent being told what to do by people outside of the team. Members are liable to obstruct, rather than support, an external project manager.

1. Scrum assumes that the team is made up of full-time workers who share a workspace. In reality, team members may be part-time and may work in different places. For a student project, team members may take different classes at different times, so it may be difficult to find a time slot where all team members can meet.
2. Scrum assumes that all team members can attend a morning meeting to coordinate the work for the day. This does not take into account that team members may work flexible hours (for example, because of child care responsibilities) or may work part-time on several projects. They are, therefore, not available every morning.

-Ez to adapt to change
-Need a team with a diverse skillset, meaning hard to find good teammates.
-ScrumMaster: Helps team through Scrum process while incorporating external ideas.

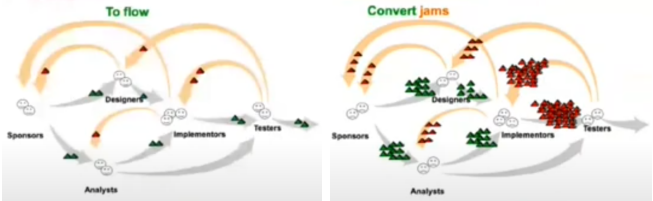


Heart of Agile (Alistair C. Lecture Vid)

-What is Agile?
-Ability to change direction quickly and ease.
-It's not about working faster, but getting more value.
-Back then, planning was heavily focused. When that plan fails, bruh moment.
-Not a framework, more like a compass. What should dev team focus on next?

-Collaborate, Deliver, Reflect, Improve

-Deliver
-Smaller pieces are easier to manage.
-Doing too much will make it way harder to know priorities



-Find mistakes and learn from them ASAP, opposed to later where there will be too much to handle.
-Handling mistakes later makes the higher up more confused on your dev team's ability to make something.

-Collaborate
-Just talk and listen to people 4head

-Improve & Reflect
-Small improvement is best. It's easier to do small increments

-Reflect on how to improve better for next increment, project, etc.

Discussion 1: Continuous Integration

https://martinfowler.com/articles/continuousIntegration.html

-Wat the continuous integration?

-Use Source Control & unit tests, where on every commit, you have to pass the tests.

-Contribute frequently to the main branch (ie daily).

-This drives dev from making their builds take too long to compile.

-When you have conflicts when committing your version, it's your job to fix it because the other person passed the unit tests already.

-Since frequent commits, there shouldn't be that much to change and it's easier to be flexible with teammates.

-Automate creating the build, and testing server side so that there is less chance to create errors through local machine differences.

Chapter 2: Key Points

- The best way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.
- Agile methods are based on iterative development and the minimization of overheads during the development process.
- Extreme Programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development, and continuous integration. These are now mainstream software development activities.
- Scrum is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used. The development team may use any technical practices they consider appropriate for the product being developed.
- In Scrum, work to be done is maintained in a product backlog, a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.
- Sprints are fixed-time activities (usually two to four weeks) in which a product increment is developed. Increments should be potentially shippable; that is, they should not need further work before they are delivered.
- A self-organizing team is a development team that organizes the work to be done by discussion and agreement among team members.
- Scrum practices, such as the product backlog, sprints, and self-organizing teams, can be used in any agile development process, even if other aspects of Scrum are not used.

Class: Wat the Agile? (Chapter 1 & 2)

-Why Software Quality Assurance?

-No one wants trash broken software

-Software's expectation, reputation, reviews.

-SAFETY (is our number priority)

-Marketing, customer cost vs quality of software.

-A way for the dev team to measure their software's quality during development.

-Is the quality wanted within time (also maintenance after shipping) and within budget.

-Ultimate Answer: Dev teams need a way to reliability makes a good program/software for the customer.

-Why does software fail?

-Poor security (malware, vulnerability)

-Bugs

-HW fails

-Missing dependencies

-User (devs didn't understand what user wanted to do)

-Over budget/time

-Compatibility

-Bad team composition, collab skills & communication

-2 Approaches to Software Projects

-1. Design in conversative manner, making a software that is done before using tried and true ways.

-2. Making something new, without historically tried methods.

-Taking on a project that is of type 2 but thinking it's type 1 leads to issues.

-User Manifesto

-We want product teams to

-solve my problems

-evolve as my needs evolve

-work the way I work

-lets me see my job getting done

-feel a joy in using

-Methods and Methodologies for Software Projects

-Activities (discussion)

-1. Goal/Vision/Brainstorm: What should it do, how much will it need (time&cost)

-2. Design UI, Network, Platform, Data

-3. Scheduling Milestones, Pacing, Delegation

-4. Develop/Code/Implement

-5. Test, QA

-Jebait, this is called Waterfall Model of development, it has been phased out now.

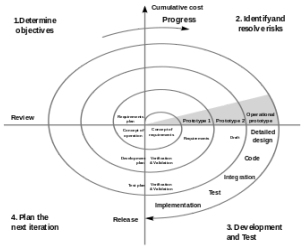
-Feasibility -> Analysis -> Authorization from Backers -> Implementation -> Test -> Deploy & Maintenance -> [back to start]

-US Department of Defense standardize this in 1985, but the people who made this model warned that there are pitfalls like

-Errors found early on tend to be pushed to later in development

-Errors will make other stages be less prioritized, room for even more mistakes.

-Spiral Model: Alternative look on Waterfall method.



-Agile

-Incremental Development. Do small increments and learn from that to push forward to the next increment.

-Feature focused, what features does the user want?

-Prioritize the most important feature, make a prototype to get feedback ASAP

-From a prioritized list of features, pick a few of the most important, plan & develop it with Scrum type beat, deliver what you got to get feedback, again.

(More at Agile Manifesto chapter)

Waterfall	Agile

Structured, 1 Big Project, Sequential, Good when change is uncommon, internal	Smaller projects, iterative, can adapt to change, good for continuous improvements, considers external feedback

-Scrum Framework
- -An Agile framework for development
- -Roles in Project Team
  - -Members of Dev Team: Devs, Designers, etc.
  - -Product Owner: Believes in the product, knowing most about the what the product should do, should be able to make decision on where project should go
  - -ScumMaster: Leads scrum process
- -Product Stakeholders
  - -The people involved & interested in the product are not making it.
  - -Without them, it's hard to tell how many people will actually want the product
- -Dev Team
  - -3-9 people
    - -teams bigger can get into each other's way more
    - -2 doesn't need scrum, but they prob dont have enough skill
    - -Bigger projects split their team to smaller teams focusing on different stuff
- -Product Owner
  - -Represents the voice of user to team, and vice versa
- -Scrum Master
  - -Like a referee to help get the work done
- -Other roles
  - -Business Owner: Mr. Money
  - -Senior Developer: Talks to Business Owner
  - -Project Manager: Responsible to guiding project through planning and
- -Scrum Events/Steps
  - -Sprint
  - -Sprint Plan Meeting
  - -Daily Scrum
  - -Sprint Review Meeting
  - -Sprint Retrospective Meeting
    - -Team needs to state what they did, what they are working on next, what they think others can do to help
    - -Team needs to reflect what they did well and how can they improve next Sprint
    - -need to be consistent.
    - -Sprint planning should not have tech get in the way of sharing ideas.
- -Definition of "Done"
  - -What is finished?
  - -You don't want someone to keep expanding the scope.
  - -Ex: No unit test errors
- -Scrum Mindset
  - -It's fine to not know everything at the start
  - -Shouldn't be afraid of stepping out of comfort zone
  - -Not afraid to ask others for help, knowing they are a part of a team
  - -Able to say "no"
  - -More than just a coder.

-Quality
- -A balance between cost/resources, scope, & time.
- -Project Success Sliders: Project Success Sliders: Agile Project ROI (mountaingoatsoftware.com)
  - -A nicer model (respec mirror be like) considering team, stakeholder, etc.
- -Myth of the Man-Month
  - -More people, better quality? No. Too many cooks in the kitchen, harder to manage.
- -Communication is important, don't want a game of telephone with what the product should be.

Class: Project Beginnings

-Gathering Requirements
- -Ask questions
  - -Fall back option is a survey.
- -Closed End - Details
- -Open End - Impression, Likes/Dislikes
- -Prototype for advice on where to go from there
- -Pain Points (what should the devs anticipates)
- -Ideal World, what would be really nice to have. Could be hard or easy to implement.
  - -"What would a perfect birdhouse look like?"

-New Systems
- -A well known thing, typically done by hand, but you are trying to make it in computers
  - -Or on a old system and needs to be revived.

-Project Artifacts
- -Forms, Printouts, Paper (things your new system have to replace/do better than)
- -Shadow (see how people are using the old stuff)

-Features
- -A way to think of them is as invitations for a conversion.

3.1 Personas:

-Personas
- -Imagined Users that you think might use your product
  - -Can be made up, just to get you into the shoes of a customer
  - -If you're making an appointment management application, personas could be doctor, dentist, patient, receptionist, etc.
  - -How good are these personas at using the application?
    - -ie doctor knows way more diction than the patient, so UI needs to accomidate.
- -Ex: ILearning, Elementary School Teacher

Table 3.1 A persona for a primary school teacher

Jack, a primary school teacher
Jack, age 32, is a primary school (elementary school) teacher in Ullapool, a large coastal village in the Scottish Highlands. He teaches children from ages 9 to 12. He was born in a fishing community north of Ullapool, where his father runs a marine fuels supply business and his mother is a community nurse. He has a degree in English from Glasgow University and retrained as a teacher after several years working as a web content author for a large leisure group. Jack's experience as a web developer means that he is confident in all aspects of digital technology. He passionately believes that the effective use of digital technologies, blended with face-to-face teaching, can enhance the learning experience for children. He is particularly interested in using the iLearn system for project-based teaching, where students work together across subject areas on a challenging topic.

Table 3.3 A persona for a history teacher

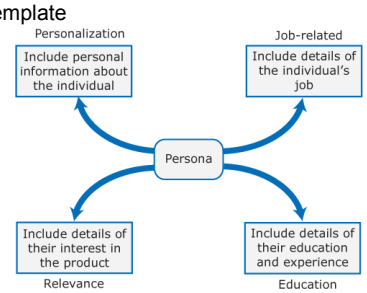
Emma, a history teacher
Emma, age 41, is a history teacher in a secondary school (high school) in Edinburgh. She teaches students from ages 12 to 18. She was born in Cardiff in Wales, where both her father and her mother were teachers. After completing a degree in history from Newcastle University, she moved to Edinburgh to be with her partner and trained as a teacher. She has two children, aged 6 and 8, who both attend the local primary school. She likes to get home as early as she can to see her children, so often does lesson preparation, administration, and marking from home. Emma uses social media and the usual productivity applications to prepare her lessons, but is not particularly interested in digital technologies. She hates the virtual learning environment that is currently used in her school and avoids using it if she can. She believes that face-to-face teaching is most effective. She might use the iLearn system for administration and access to historical films and documents. However, she is not interested in a blended digital/face-to-face approach to teaching.

Table 3.4 A persona for an IT technician

Elena, a school IT technician
Elena, age 28, is a senior IT technician in a large secondary school (high school) in Glasgow with over 2000 students. Originally from Poland, she has a diploma in electronics from Potsdam University. She moved to Scotland in 2011 after being unemployed for a year after graduation. She has a Scottish partner, no children, and hopes to develop her career in Scotland. She was originally appointed as a junior technician but was promoted, in 2014, to a senior post responsible for all the school computers. Although not involved directly in teaching, Elena is often called on to help in computer science classes. She is a competent Python programmer and is a "power user" of digital technologies. She has a long-term career goal of becoming a technical expert in digital learning technologies and being involved in their development. She wants to become an expert in the iLearn system and sees it as an experimental platform for supporting new uses for digital learning.

(more)

-Good Template



Personalization	You should give them a name and say something about their personal circumstances. It is sometimes helpful to use an appropriate stock photograph to represent the person in the persona. Some studies suggest that this helps project teams use personas more effectively.
Job-related	If your product is targeted at business, you should say something about their job and (if necessary) what that job involves. For some jobs, such as a teacher where readers are likely to be familiar with the job, this may not be necessary.
Education	You should describe their educational background and their level of technical skills and experience. This is important, especially for interface design.
Relevance	If you can, you should say why they might be interested in using the product and what they might want to do with it.

-Proto-personas

-Personas of the dev team and what they want out of the product.

-If you can't make personas because the product is too unique, or you don't know people that are interested in your product, fallback to proto-persona.

3.2 Scenarios

-Scenario

-A small narrative that describes a situation in which a user is using your product's features to do something that they want to do.

-Shouldn't include implementation details.

-User can get involved writing, but they can be too detailed

-Better to get their feedback on a scenario you've written

-Ex: Jack & ILearn

Fishing in Ullapool

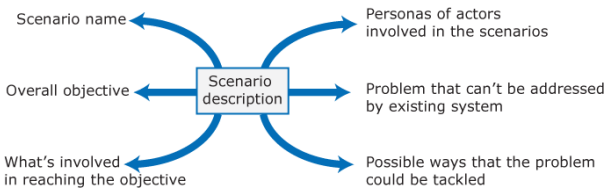
Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development, and economic impact of fishing.

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives, and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history archive site) to access newspaper archives and photographs. However, Jack also needs a photo-sharing site as he wants students to take and comment on each others' photos and to upload scans of old photographs that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack sends an email to a primary school teachers' group to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he use KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that, when they log in, they can immediately use the system to upload photos from their phones and class computers.

-Aspect of scenario



-Ex: Jack's Scenario

1. A brief statement of the overall objective. In Jack's scenario, shown in **Table 3.5**, this is to support a class project on the fishing industry.
2. References to the persona involved (Jack) so that you can get information about the capabilities and motivation of that user.
3. Information about what is involved in doing the activity. For example, in Jack's scenario, this involves gathering reminiscences from relatives, accessing newspaper archives, and so on.
4. If appropriate, an explanation of problems that can't be readily addressed using the existing system. Young children don't understand issues such as copyright and privacy, so photo sharing requires a site that a teacher can moderate to make sure that published images are legal and acceptable.
5. A description of one way that the identified problem might be addressed. This may not always be included especially if technical knowledge is needed to solve the problem. In Jack's scenario, the preferred approach is to use an external tool designed for school students.

-Structured Scenario

-Can maybe help, they are a description of a user navigating the program to do whatever they need.

-Things that should go well
-Things that can go wrong

-Ex: Emma field trip & ILearn

Table 3.6 Using the iLearn system for administration

Emma is teaching the history of World War I to a class of 14-year-olds (S3). A group of S3 students are visiting the historic World War I battlefields in northern France. She wants to set up a "battlefields group" where the students who are attending the trip can share their research about the places they are visiting as well as their pictures and thoughts about the visit.

From home, she logs onto the iLearn system using her Google account credentials. Emma has two iLearn accounts—her teacher account and a parent account associated with the local primary school. The system recognizes that she is a multiple account owner and asks her to select the account to be used. She chooses the teacher account and the system generates her personal welcome screen. As well as her selected applications, this also shows management apps that help teachers create and manage student groups.

Emma selects the "group management" app, which recognizes her role and school from her identity information and creates a new group. The system prompts for the class year (S3) and subject (history) and automatically populates the new group with all S3 students who are studying history. She selects those students going on the trip and adds her teacher colleagues, Jamie and Claire, to the group.

She names the group and confirms that it should be created. The app sets up an icon on her iLearn screen to represent the group, creates an email alias for the group, and asks Emma if she wishes to share the group. She shares access with everyone in the group, which means that they also see the icon on their screen. To avoid getting too many emails from students, she restricts sharing of the email alias to Jamie and Claire.

The group management app then asks Emma if she wishes to set up a group web page, wiki, and blog. Emma confirms that a web page should be created and she types some text to be included on that page.

She then accesses Flickr using the icon on her screen, logs in, and creates a private group to share trip photos that students and teachers have taken. She uploads some of her own photos from previous trips and emails an invitation to join the photo-sharing group to the battlefields email list. Emma uploads material from her own laptop that she has written about the trip to iLearn and shares this with the battlefields group. This action adds her documents to the web page and generates an alert to group members that new material is available.

-Writing/Creating Scenario
-Start with persona
-Don't have to cover every aspect of your program, just something to kick start.
-User POV
-Don't focus on goals. Focus on how they interact
-Write it literally and precisely what they are using (ie Google sign in), don't generalize interactions.
-You can get user feedback after writing, since it's ez to understand

-Ex: Elena configuring ILearn

Table 3.7 Elena's scenario: Configuring the iLearn system

Elena has been asked by David, the head of the art department in her school, to help set up an iLearn environment for his department. David wants an environment that includes tools for making and sharing art, access to external websites to study artworks, and "exhibition" facilities so that the students' work can be displayed.

Elena starts by talking to art teachers to discover the tools that they recommend and the art sites that they use for studies. She also discovers that the tools they use and the sites they access vary according to the age of their students. Consequently, different student groups should be presented with a toolset that is appropriate for their age and experience.

Once she has established what is required, Elena logs into the iLearn system as an administrator and starts configuring the art environment using the iLearn setup service. She creates sub-environments for three age groups plus a shared environment that includes tools and sites that may be used by all students.

She drags and drops tools that are available locally and the URLs of external websites into each of these environments. For each of the sub-environments, she assigns an art teacher as its administrator so that they can't refine the tool and website selection that has been set up. She publishes the environments in "review mode" and makes them available to the teachers in the art department.

After discussing the environments with the teachers, Elena shows them how to refine and extend the environments. Once they have agreed that the art environment is useful, it is released to all students in the school.

3.3 User "1 sentence" Stories "andy"

-User Stories
-User stories are finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system.
-Scenarios are high-level stories of system use. They should describe a sequence of interactions with the system but should not include details of these interactions.
-Aim to make them
- as a way of extending and adding detail to a scenario;
- as part of the description of the system feature that you have identified.

-Formula
-As a <role>, I <want/need> to <do something> (optional: so that <reason>)
-A way is to make a list of roles and a list for objects/entities, then make connections from both lists by an action.

-Ex: 2 examples

- *As an author I need a way to organize the book that I'm writing into chapters and sections.*

This story reflects what has become the standard format of a user story:

As a <role>, I <want / need> to <do something>

Another example of a user story taken from Emma's scenario might be:

- *As a teacher, I want to tell all members of my group when new information is available.*

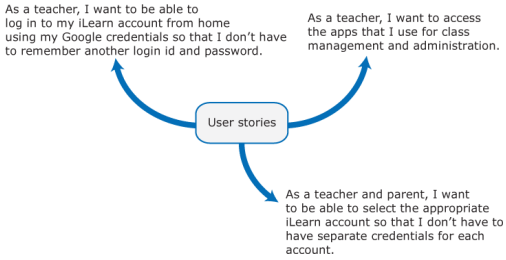
A variant of this standard format adds a justification for the action:

As a <role> I <want / need> to <do something> so that <reason>

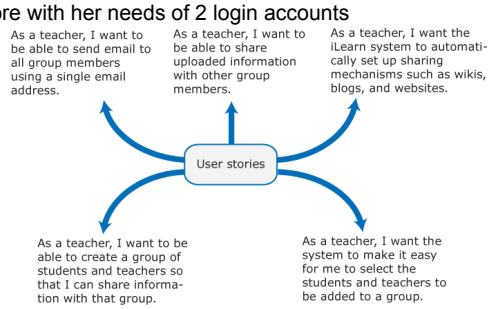
For example:

As a teacher, I need to be able to report who is attending a class trip so that the school maintains the required health and safety records.

-The variation with <reason> or justification is ok, but author thinks it is unnecessary, though can give ideas for alternative implementation.
-Emma's scenario can be broken down into 3 other (by wants/ needs)



-Even more with her needs of 2 login accounts



-Epic

-A slightly longer (2 <wants / needs>) user story

-Ex:

As a system manager, I need a way to back up the system and restore individual applications, files, directories, or the whole system.

-Negative User Stories

-For product backlog, probably not the best because you can't create tests for it.

As a user, I don't want the system to log and transmit my information to any external servers.

-Try reframing it into a positive

As a user, I want to be able to control the information that is logged and transmitted by the system to external servers so that I can ensure that my personal information is not shared.

-Scenarios are still useful

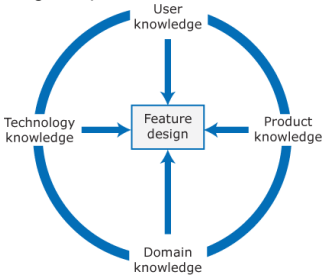
1. Scenarios read more naturally because they describe what a user of a system is actually doing with that system. People often find it easier to relate to this specific information rather than to the statement of wants or needs set out in a set of user stories.
2. When you are interviewing real users or checking a scenario with real users, they don't talk in the stylized way that is used in user stories. People relate better to the more natural narrative in scenarios.
3. Scenarios often provide more context—information about what users are trying to do and their normal ways of working. You can do this in user stories, but it means that they are no longer simple statements about the use of a system feature.

3.4 Feature Identification

-Ideally, identify product features that are independent, coherent and relevant:

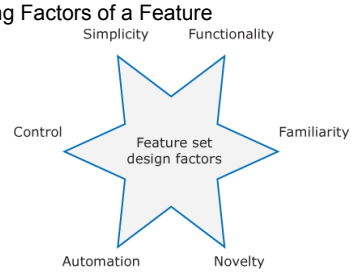
1. **Independence** A feature should not depend on how other system features are implemented and should not be affected by the order of activation of other features.
2. **Coherence** Features should be linked to a single item of functionality. They should not do more than one thing, and they should never have side effects.
3. **Relevance** System features should reflect the way users normally carry out some task. They should not offer obscure functionality that is rarely required.

-Feature Design inspiration sources



User knowledge	You can use user scenarios and user stories to inform the team of what users want and how they might use the software features.
Product knowledge	You may have experience of existing products or decide to research what these products do as part of your development process. Sometimes your features have to replicate existing features in these products because they provide fundamental functionality that is always required.
Domain knowledge	This is knowledge of the domain or work area (e.g., finance, event booking) that your product aims to support. By understanding the domain, you can think of new innovative ways of helping users do what they want to do.
Technology knowledge	New products often emerge to take advantage of technological developments since their competitors were launched. If you understand the latest technology, you can design features to make use of it.

-Balancing Factors of a Feature



1. **Simplicity and functionality** Everyone says they want software to be as simple as possible to use. At the same time, they demand functionality that helps them do what they want to do. You need to find a balance between providing a simple, easy-to-use system and including enough functionality to attract users with a variety of needs.
2. **Familiarity and novelty** Users prefer that new software should support the familiar everyday tasks that are part of their work or life. However, if you simply replicate the features of a product that they already use, there is no real motivation for them to change. To encourage users to adopt your system, you need to include new features that will convince users that your product can do more than its competitors.
3. **Automation and control** You may decide that your product can automatically do things for users that other products can't. However, users inevitably do things differently from one another. Some may like automation, where the software does things for them. Others prefer to have control. You therefore have to think carefully about what can be automated, how it is automated, and how users can configure the automation so that the system can be tailored to their preferences.

-Another trade off: Security vs ease of Access

-Feature Creep

-Implementing too many features from more user knowledge

-ie Office or Photoshop, most normal users don't come close to using everything

-3 Reasons for Feature Creep

1. Product managers and marketing executives discuss the functionality they need with a range of different product users. Different users have slightly different needs or may do the same thing but in slightly different ways. There is a natural reluctance to say no to important users, so functionality to meet all of the users' demands ends up in the product.
2. Competitive products are introduced with slightly different functionality to your product. There is marketing pressure to include comparable functionality so that market share is not lost to these competitors. This can lead to "feature wars," where competing products become more and more bloated as they replicate the features of their competitors.
3. The product tries to support both experienced and inexperienced users. Easy ways of implementing common actions are added for inexperienced users and the more complex features to accomplish the same thing are retained because experienced users prefer to work that way.

-Feature Derivation

-By Product/System Vision, Scenarios, User Stories, you derive features from them.

-Ex: Vision

Table 3.9 The iLearn system vision

FOR teachers and educators WHO need a way to *help students use web-based learning resources and applications*, THE iLearn system is an open learning environment THAT *allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves.*

UNLIKE Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process rather than the administration and management of materials, assessments, and coursework. OUR product *enables teachers to create subject and age-specific environments for their students* using any web-based resources, such as videos, simulations, and written materials that are appropriate

- a feature that allows users to access and use existing web-based resources;
- a feature that allows the system to exist in multiple different configurations;
- a feature that allows user configuration of the system to create a specific environment.

-Ex: Scenario

Table 3.10 Jack's scenario with highlighted phrases

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development, and economic impact of fishing.

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives, and collect old photographs related to fishing and fishing communities in the area. *Students use an iLearn wiki* to gather together fishing stories and *SCRAN (a history archive) to access newspaper archives and photographs.* However, Jack also needs a photo-sharing site as he wants *pupils to take and comment on each others' photos and to upload scans of old photographs* that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack *sends an email to a primary school teachers' group* to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he use KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics *is not integrated with the iLearn authentication service,* he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that when they log in, they can immediately use the system to upload photos from their phones and class computers.

- a wiki for group writing;
- access to the SCRAN history archive, which is a shared national resource that provides access to historical newspaper and magazine articles for schools and universities;
- the ability to set up and access an email group;
- the ability to integrate some applications with the iLearn authentication service.

-Ex: User Story

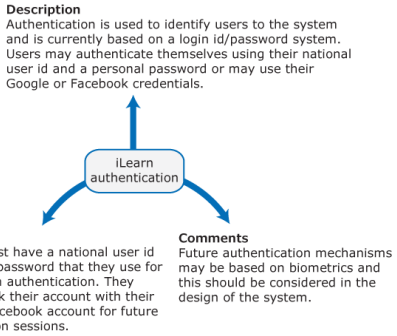
As a teacher and a parent, I want to be able to select the appropriate iLearn account so that I don't have to have separate credentials for each account.

- This story states that the account feature of the iLearn system has to accommodate the idea that a single user may have multiple accounts. Each account is associated with a particular role the user may adopt when using the system. When logging in, users should be able to select the account they wish to use.

-Feature List

-Description, Constraint, Comment Technique

-Ex: ILearn Authentication



-Ex: Simple description

*Print **the** document **to** a selected printer **or to** PDF.*

-Ex: From User Stories

Table 3.11 Feature description using user stories

iLearn system configuration
Description
As a system manager, I want to create and configure an iLearn environment by adding and removing services to/from that environment so that I can create environments for specific purposes.
As a system manager, I want to set up sub-environments that include a subset of services that are included in another environment.
As a system manager, I want to assign administrators to created environments.
As a system manager, I want to limit the rights of environment administrators so that they cannot accidentally or deliberately disrupt the operation of key services.
As a teacher, I want to be able to add services that are not integrated with the iLearn authentication system.
Constraints
The use of some tools may be limited for license reasons so there may be a need to access license management tools during configuration.
Comments
Based on Elena's and Jack's scenarios

Chapter 3: Key Points

- A software product feature is a fragment of functionality that implements something a user may need or want when using the product.
- The first stage of product development is to identify the list of product features in which you name each feature and give a brief description of its functionality.
- Personas are "imagined users"— character portraits of types of users you think might use your product.
- A persona description should paint a picture of a typical product user. It should describe the user's educational background, technology experience, and why they might want to use your product.
- A scenario is a narrative that describes a situation where a user is accessing product features to do something that they want to do.
- Scenarios should always be written from the user's perspective and should be based on identified personas or real users.
- User stories are finer-grain narratives that set out, in a structured way, something that a user wants from a software system.
- User stories may be used to extend and add detail to a scenario or as part of the description of system features.
- The key influences in feature identification and design are user research, domain knowledge, product knowledge, and technology knowledge.
- You can identify features from scenarios and stories by highlighting user actions in these narratives and thinking about the features that you need to support these actions.

Chapter 2 External Reading:

-User Stories Intro Thing

1. **Stakeholders write user stories.** An important concept is that your stakeholders write the user stories, not the developers. User stories are simple enough that people can learn to write them in a few minutes, so it makes sense that the domain experts (the stakeholders) write them.

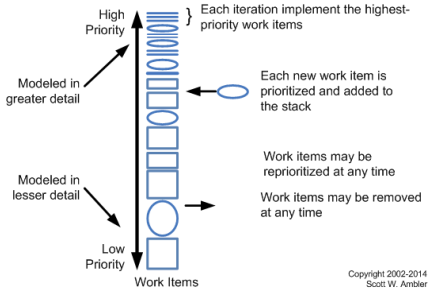
2. **Use the simplest tool**. User stories are often written on index cards as you see in Figure 2 (at least when your project team is co-located). Index cards are very easy to work with and are therefore an inclusive modeling technique.

3. **Remember non-functional requirements**. Stories can be used to describe a wide variety of requirements types. For example in Figure 1 the *Students can purchase parking passes online* user story is a usage requirement similar to a use case whereas the *Transcripts will be available online via a standard browser* is closer to a technical requirement.

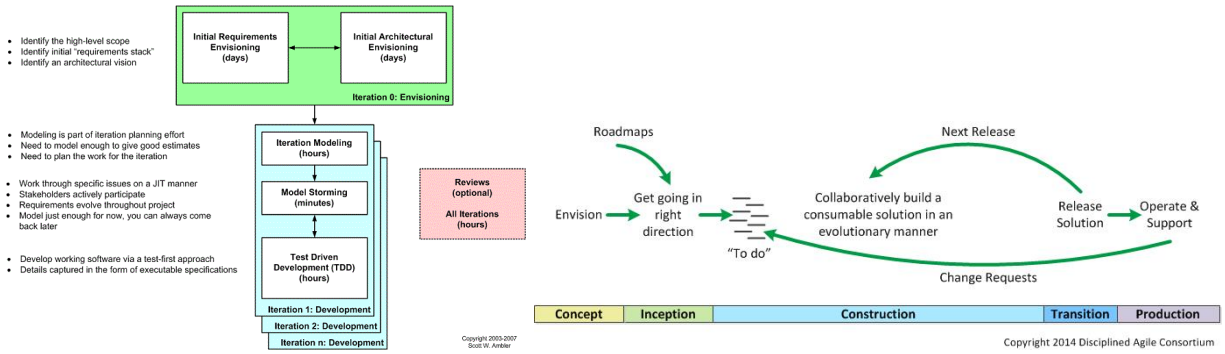
4. **Indicate the estimated size**. You can see in Figure 2 that it includes an estimate for the effort to implement the user story. One way to estimate is to assign user story points to each card, a relative indication of how long it will take a pair of programmers to implement the story. The team then knows that if it currently takes them on average 2.5 hours per point; therefore the user story in Figure 2 will take around 10 hours to implement.

5. **Indicate the priority**. Requirements, including defects identified as part of your independent parallel testing activities or by your operations and support efforts, are prioritized by your stakeholders (or representatives thereof such as product owners) and added to the stack in the appropriate place. You can easily maintain a stack of prioritized requirements by moving the cards around in the stack as appropriate. You can see that the user story card includes an indication of the priority; I often use a scale of one to ten with one being the highest priority. Other prioritization approaches are possible – priorities of High/Medium/Low are often used instead of numbers and some people will even assign each card it's own unique priority order number (e.g. 344, 345, ...). You want to indicate the priority somehow in case you drop the deck of cards, or if you're using more sophisticated electronic tooling. Pick a strategy that works well for your team. You also see that the priority changed at some point in the past, this is a normal thing, motivating the team to move the card to another point in the stack. The implication is that your prioritization strategy needs to support this sort of activity. My advice is to keep it simple.

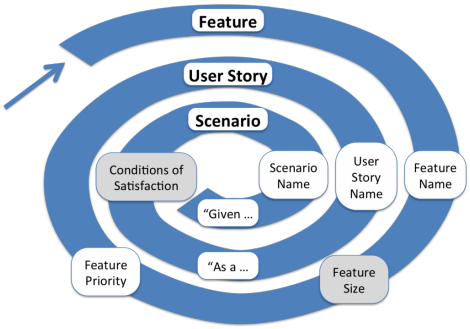
6. **Optionally include a unique identifier**. The card also includes a unique identifier for the user story, in this case 173. The only reason to do this would be to do this is if you need to maintain some sort of traceability between the user story and other artifacts, in particular acceptance tests.



1. **Inception**. You often create a stack of user stories during Inception as part of your requirements envisioning activities to identify the scope of your system.
2. **Construction**. During construction iterations you will identify new stories, split existing stories when you realize that they're too large to be implemented in single iteration, reprioritize existing stories, or remove stories that are no longer considered to be in scope. The point is that your stories evolve over time just like other types of requirements models evolve. Furthermore, enhancement requests may be identified by your support staff during the production phase and then forwarded to a development team as they are working on an upcoming release. These enhancement requests are effectively new stories (albeit in a different format).
3. **Transition**. Sometimes new stories will be identified during the Transition phase, although this isn't very common as the focus of release is on hardening the system and not on new functionality. But it does happen, and these stories would be prioritized and placed on the stack in priority order just as you normally would.



-Snail Thing



1. Feature
   - Name
   - Size [Optional]
   - Priority
2. User Story
   - Name
   - "As <user role> I want to <action> so that <goal>"
   - Conditions of Satisfaction [Optional]
3. Scenario
   - Name
   - "Given <precondition> When <trigger> Then <result>"

-Acceptance Test

-After user story, and then implementation, you get feedback from the customer who wrote the card.

-If they don't accept the feature, you goofed up.

-Iterative & Incremental Development

The word *increment* fundamentally means *add onto*.

The word *iterate* fundamentally means *re-do*.

Incremental development gives you opportunities to improve your development process, as well as adjust the requirements to the changing world.

Iterative development helps you improve your product quality. Yes, it is rework, and yes, you probably need to do some rework to make your product shine.

The development process, feature set, and product quality all need constant improvement. Use an *incremental* strategy, with reflection, to improve the first two. Use an *iterative* strategy, with reflection, to improve the third.

Discussion 2:

-After reading Chapter 2.3 on features, I wanted to know more about how to prioritize features. What should the dev team focus on first in their first sprint of the iterative and increment development process?

-https://www.productplan.com/learn/strategies-prioritize-product-features/

A broad overview of various ways to feature prioritization. There's Value VS Complexity Quadrant, saying how devs want to prioritize features with a low complexity but high business value. Charting and rating features by benefits vs cost gives you a ranking for each feature, which then can be organized to a sprint roadmap. The article also shows an example of ordering user stories from least to most importance.

-https://medium.com/@reinaldocamargo/product-backlog-prioritization-technique-kano-model-63bc5d28a1fe

An article describing the Kano Model. The model is a graph of the relationship between a feature that satisfies and its investment needed. From surveying stakeholders about their satisfaction if a feature was vs was not implemented, you can make a chart of the most important/required features.

Class: Requirements **Features, Scenarios and Stories - Engineering Software Products (iansommerville.com)**

-Gathering, Validating, Documenting Requirements

-Elicit requirements from the customers

-Reviewing Requirements to ensure quality

-Documenting requirements during design

-What is a requirement?

-Deals with objects/entities, the state they can be in,
-Focus on the customer's wants
-Customer usually don't know what they need, you have to help each other know what the product should be and agree on it
-Stakeholders
-Clients, Customers, etc.
-Types of requirements
-Functional: Required activities/behaviors
-Quality/Nonfunctional: Quality characteristics
-Business Rules: Limits on behaviors based on business/work processes.
-Environment Constraints: Resources that you don't have access to (ie Mobile is weaker than PC)
-2 Requirement Documentations
-Requirement Definition
-What the customer wants to achieve
-Broad and more narrative
-Can change with agile
-Requirement Specifications
-Behaviors of the system
-Software features
-Look at 3.4 notes
-When starting, make a mega list of features. Then weed out the ones you're not doing

Class: Increment & Iterate

-Increment
-1 part at a time
-Iterate
-Make trash (ie prototype), start over and make it better with what you learned. Could take longer than increment
-Pitfall of using Agile
-Not iterating or incrementing, backlog was too fat.

Quiz 2

Gathering requirements is a process to capture, quantify, and prioritize users' needs. We've looked at collecting User Stories as a technique for gathering functional requirements. Given that different stakeholders will have different points of view, are there other techniques that would compliment User Stories to help elicit requirements from their viewpoint? Consider that some viewpoints may not be concerned with functionality.

-Proto-personas and proto stories cheese
-Discussion 2 research
-Investors

In Chapter 3.3, Sommerville explains User Stories and primarily gives examples of stories from the perspective of various customers. Stakeholders include way more than just the customer, from the people using to the people funding, each have differing viewpoints on the product they want made. Proto-personas, and business focused techniques like the Kano model based survey are other complementary techniques can elicit requirements from different stakeholders.

Usually, user stories are based on detailed user studies, such as interviewing stakeholders. If that doesn't help generate enough requirements, like how our class project only has one stakeholder interview, a development team can create proto-personas which "represent the product users as seen by the development team, and they allow the developers to build a common understanding of the potential product users." (Sommerville, Chapter 3.1). In our requirements draft, we made user stories from proto-personas out of Bob's wife and children, describing how they might want to use our product to look up a home appliance's manual. As I'm typing this, I think I'm going to add a proto-persona of a computer building enthusiast to our draft. Tangentially related to a DIY homeowner, a possible user story is "As a computer building enthusiast, I want to keep track of information about warranty and the RMA details of a part in case something breaks". We only thought of receipt management from Bob's persona, so this is already helping elicit requirements from different viewpoints.

For a business based stakeholder, requirements that generate the most profit and can be implemented the fastest are most preferred. A 1980s model by professor Noriaki Kano provides insight to the relationship between features that yields the most customer satisfaction versus its investment (Camargo). The gist is that as the investment in a feature increases, a feature that stakeholders deems satisfactory grows exponentially more satisfying than a baseline/needed feature which stagnates in their ability to satisfy after a certain time of investment. To find requirements that are deemed satisfactory by the stakeholders, a way to do that is with a survey where a feature is rated on how satisfied they feel if a feature was vs if it wasn't implemented. Applying this, we could ask Bob this type of question on various features, features thought up from our proto-personas, eliciting requirements from a different perspective.

-Sommerville, Ian. "Engineering Software Products: An Introduction to Modern Software Engineering."
-Camargo, Reinaldo. "Product backlog prioritization technique: Kano Model." Medium, https://medium.com/@reinaldocamargo/product-backlog-prioritization-technique-kano-model-63bc5d28a1fe

Class: Git a cuh

-Stages

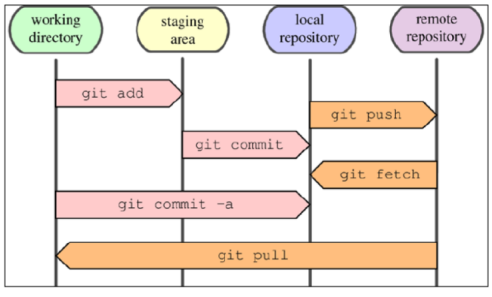
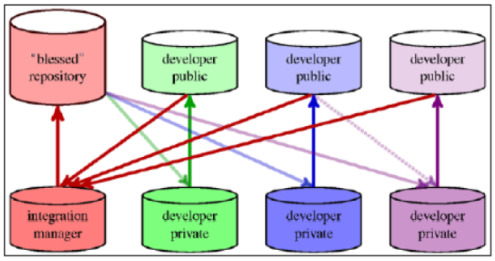
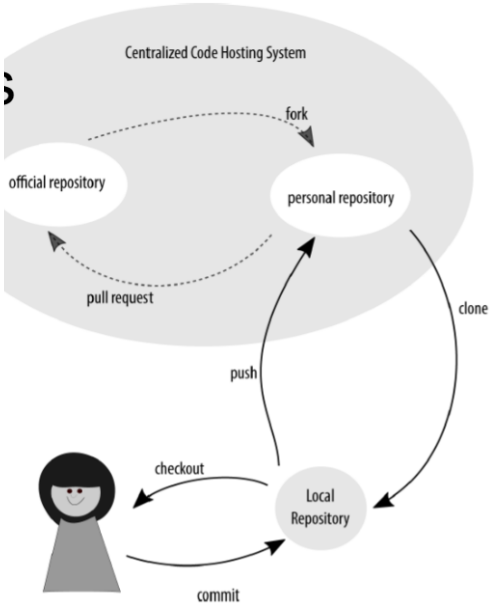


Fig 1: Creating and publishing commits.

-Fork and integration manager



-Commit, Pull, bruh Terminology



Paper Prototypes

-What the paper prototype?

Tools for Paper Prototyping

- White poster board (11"x14")
  - For background, window frame
- Big (unlined) index cards (4"x6", 5"x8")
  - For menus, window contents, and dialog boxes
- Restickable glue
  - For keeping pieces fixed
- White correction tape
  - For text fields, checkboxes, short messages
- Overhead transparencies
  - For highlighting, user "typing"
- Photocopier
  - For making multiple blanks
- Pens & markers, scissors, tape

Tools for Paper Prototyping

- White poster board (11"x14")
  - For background, window frame
- Big (unlined) index cards (4"x6", 5"x8")
  - For menus, window contents, and dialog boxes
- Restickable glue
  - For keeping pieces fixed
- White correction tape
  - For text fields, checkboxes, short messages
- Overhead transparencies
  - For highlighting, user "typing"
- Photocopier
  - For making multiple blanks
- Pens & markers, scissors, tape

Tips for Good Paper Prototypes

- Make it larger than life
- Make it monochrome
- Replace tricky visual feedback with audible descriptions
  - Tooltips, drag & drop, animation, progress bar
- Keep pieces organized
  - Use folders & open envelopes

What You Can Learn from a Paper Prototype

- Conceptual model
  - Do users understand it?
- Functionality
  - Does it do what's needed? Missing features?
- Navigation & task flow
  - Can users find their way around?
  - Are information preconditions met?
- Terminology
  - Do users understand labels?
- Screen contents
  - What needs to go on the screen?

What You Can't Learn

- Look: color, font, whitespace, etc
- Feel: efficiency issues
- Response time
- Are small changes noticed?
  - Even the tiniest change to a paper prototype is clearly visible to user
- Exploration vs. deliberation
  - Users are more deliberate with a paper prototype; they don't explore or thrash as much

-Use paper to draw out ui.

-Let someone rp as user and another as the computer

-Unlike hi-fi demos and what, you didnt waste as much time was bad design.

-Not set in stone, you dont get too attached with the work an is willing to change it

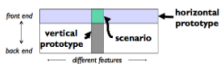
-No bugs to fix when you have to scrap it and change or creating new elements

-Software prototype good to see if design is feasible. Paper prototype is good to see interaction with user and conceptuallize design

-Fidelity

Fidelity is Multidimensional

- Breadth: % of features covered
  - Only enough features for certain tasks
- Depth: degree of functionality
  - Limited choices, canned responses, no error handling



-How to paper prototype?

-White paper,sticky notes, small note cards, colored pens/marker, glue, tape, scissors

-Hand drawn, then for rp testing, print out nicely.

Quiz 3

Alan Cooper (the "Father of Visual Basic") wrote an essay on "The Perils of Prototyping" advocating low-fidelity prototyping over high fidelity options (like VB) as it removes the temptation to keep code you worked hard to write.

Assume that argument is quelled; your development manager has mandated that all prototype code will be discarded, and the effort and cost to build multiple high fidelity prototypes is already planned and budgeted.

Use a specific user story or use case scenario to illustrate your answer to the following:

Would you still make low-fidelity prototypes?

Are there advantages to using both low and high fidelity prototypes?

-Low for brainstorming

-Literally just did it Thursday

-Advantage for High is because user interaction not marred by trash handwriting. (Youtube vid, you dont get the timings)

-With agile, you can adapt easier. When you need to draft a new part of the ui, low fidelity paper is OP. When you need some user testing, it might be better to build mid to high fidelity to get more proper user interaction.

Even with high fidelity prototypes planned, low fidelity prototypes are definitely still useful. For example, our project now: "As a DIY Home Owner person, I want to manage projects that I am working on." When we made our personas/user stories/scenarios, I don't think anyone had a clue how the UI was going to look, or at least there is no way we could come to an agreement then. If we tried to make a high fidelity prototype, we wouldn't know where to start and who was going to do what. "Paper prototyping is excellent for: Brainstorming sessions (when a team needs to outline the steps in a user flow or explore/validate a variety of layouts)." (Babich) Doing a high fidelity prototype from the get-go, especially for a program where its purpose hasn't been historically tried sounds like the best way to create delays. UI are visual elements, unless you draw it

out, it's going to be hard to prioritize which elements should be included in a higher fidelity prototype. So if I was with the project manager, I would create low fidelity user prototypes for brainstorming, in order to prioritize what needs to be implemented in a high fidelity prototype.
When I was researching for Discussion 3, I came across videos of users testing paper prototypes. One was with this blood testing machine (https://youtu.be/_g4GGtJ8NCY). It played out pretty well, the person doing the paper was quick and the paper model was sized like the potentially real machine. However, some were actually ridiculous, usually when it gets more complex. One was of this banking phone app (https://youtu.be/yafaGNFu8Eg). The phone was the size of a laptop screen, they had to provide color codes for UI elements (notification, input fields, buttons), and the timings of some actions can't be replicated by a person moving papers (holding down an icon so that it enters edit mode so you can move it around). This is the drawback of low fidelity prototypes, they can't elicit the true user reaction when testing. Bad handwriting can get in the way, and you will never know if a UI element loads in too slow or is too unresponsive for the user. The time to perfect this paper prototype testing is probably better just to make an interactive medium fidelity mock-up in wire building tools (like the PowerPoint application you showed us), or commit to a high fidelity prototype like what this project manager planned.
In conclusion, lo-fi for brainstorming and hi-fi for user testing. Doing both seems to be the best case.

Babich, Nick. "The Magic of Paper Prototyping." https://babich.biz/the-magic-of-paper-prototyping/ (Links to an external site.)

Class: Design

-Diagrams
    -Domain Model
        -Model that shows relationship between high level things like the user, gui, data structure manager, etc/
    -User Sequence Model
        -Mode that shows a flow/sequence that results from user actions. Like log in, then the account manager authentication, if successful then
    -UML

Class: Software Architecture

-Architecture VS Design
    -Architecture is strategic, the bigger picture, the framework.
        -Changing architecture probability needs approval from higher ups.
        -You design the compoment's interface/interactions instead of implementing how they interact.
    -Design is opposite, the tactical small choices.
-Key Principles of Software Architecture
    -Try to give components just the info they need to do their function.
    -Components on different layers/scope shouldn't talk to components that aren't adjacent.
-Layers
    -Think of components in layers
    -Start with what the user sees, then go down to the underlying data manipulated by components controlled by users.
        -Ex: UI -> Authentication/User Management -> Application specific functions ->Basic shared services -> transaction & basic management
    -"Model-View-Controller"

Chapter 4: Architecture

-Diagrams

Class: Check In 2

-Team TDI
    -"Add project page", we forgot and should have.
    -File Explorer when selecting files
    -Controller? Is it needed?
        -Supposed to ask the database to do actions, told by the home page. Skip the middleman then?
    -Rename our Managers to Database?
-Team One
    -They focused on a smaller user story, about managing files.
        -They have a labeling system, a pointer to a file?
    -Arrow solid vs dashed.
        -Dashed is when there is a return/message sent.
        -Solid is actions
-Project6 **(**Our Team)
    -Avoid using Manager & describe how the Projects are going to be manage.
-TPSink
    -View is code word for what the User sees.
    -Which view should?
-TeamName
    -Desktop as a thing in domain model
        -App as a part of the desktop
    -"Folder" has a lot of meaning. It's broad
-Macrosoft
    -They did what we need to do. No manager, the app/gui should have a direct relationship with the item/project objects.
        -So maybe our GUI should manage everything instead of another object to do so?
    -Keywords and Tags, I don't know why we didn't include that in our domain model.
    -Like TPSink, we need views for things like "add a new project", "settings", etc.

Class: Design Heuristics

-How to know if OO design is good?
    -Just subconscious 4head
        -We have heuristics
-Basic Heuristics
    -All data should be hidden within it's class
    -Related functions are close to each other
-Qualities of modular software
    -Decomposable
    -Composable
    -Understandable
    -Continuity
    -Protected/Safe
-Action VS OO