AgileSafe Practices Knowledge Base List

Id	1		
Name	User Stories		
Description	*„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 customer's 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.* *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.“ (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		No
	Development		Yes
	Environment		No
	Project Management		Yes
	Requirements		Yes
	Test		Yes
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers;	

	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced;

Id	2	
Name	Frequent small releases	
Description	*"The development team needs to release iterative versions of the system to the customers often. Some teams deploy new software into production every day. At the very least you will want to get new software into production every week or two. At the end of every iteration you will have tested, working, production ready software to demonstrate to your customers. The decision to put it into production is theirs." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	Yes
	Development	Yes
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated;

	Organisational Distribution	A – Collaborative; B – Different teams; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced;

Id	3		
Name	Open Work Space		
Description	*"Communication is very important to an Extreme Programming (XP) team. You can add vital communication paths to your team by just taking down the barriers that divide people. Putting computers in a central area that no one owns makes pair programming easier and encourages people to work together with feelings of equal value and contributions. Putting a few desks around the perimeter gives people a place to work alone without becoming disconnected from the rest of the team.* *Including a large area for daily stand up meetings keeps people from missing the meeting. Adding a conference table gives you a home for group discussions that occur spontaneously throughout the day. Being able to see the discussions encourages people to listen in or join the discussion when they have a stake in the out come.* *Adding white boards for design sketches and important notes or blank walls where user story cards can be taped ads even more channels for communication. Posting a couple large charts targeting process improvement or project progress informs the team without management hounding them about progress." (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		Yes
	Development		No
	Environment		Yes
	Project Management		No
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers;	
	Geographical Distribution	A – Co-located; B – Same building;	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; E – Contractual	

	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused;

Id	4		
Name	Daily stand up meeting		
Description	*"Communication among the entire team is the purpose of the stand up meeting. A stand up meeting every morning is used to communicate problems, solutions, and promote team focus. Everyone stands up in a circle to avoid long discussions. It is more efficient to have one short meeting that every one is required to attend than many meetings with a few developers each. When you have daily stand up meetings any other meeting's attendance can be based on who will actually be needed and will contribute. Now it is possible to avoid even scheduling most meetings. With limited attendance most meetings can take place spontaneously in front of a computer, where code can be browsed and ideas actually tried out.* *During a stand up meeting developers report at least three things; what was accomplished yesterday, what will be attempted today, and what problems are causing delays. The daily stand up meeting is not another meeting to waste people's time. It will replace many other meetings giving a net savings several times its own length." (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		No
	Development		Yes
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home;	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D	

		- System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced;

Id	5		
Name	Release plan		
Description	*"After user stories have been written you can use a release planning meeting to create a release plan. The release plan specifies which user stories are going to be implemented for each system release and dates for those releases. This gives a set of user stories for customers to choose from during the iteration planning meeting to be implemented during the next iteration. These selected stories are then translated into individual programming tasks to be implemented during the iteration to complete the stories." (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		Yes
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated;	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;	

	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused;

Id	6		
Name	Measuring project velocity		
Description	*“The project velocity (or just velocity) is a measure of how much work is getting done on your project. To measure the project velocity you simply add up the estimates of the user stories that were finished during the iteration. It's just that simple. You also total up the estimates for the tasks finished during the iteration. Both of these measurements are used for iteration planning.” (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		Yes
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing;	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;	
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary;	
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused;	

Id	7
Name	System metaphor
Description	*“The system metaphor is a means of communicating about the project in terms that both developers and customers will understand, and which does not require pre-existing familiarity with the problem domain. The system metaphor guides the mental*

	models that project members have of the system, and shapes a logical architecture for the system. " (R. Khaled, P. Barr, J. Noble and R. Biddle "System Metaphor in Extreme Programming: A Semiotic Approach")	
Discipline	Architecture	Yes
	Deployment	No
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	Yes
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance;
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing;
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	8
Name	CRC Cards
Description	*"Use Class, Responsibilities, and Collaboration (CRC) Cards to design the system as a team. The biggest value of CRC cards is to allow people to break away from the procedural mode of thought and more fully appreciate object technology. CRC Cards allow entire project teams to contribute to the design. The more people who can help design the system the greater the number of good ideas incorporated.* *Individual CRC Cards are used to represent objects. The class of the object can be written at the top of the card, responsibilities listed down the left side, collaborating classes are listed to the right of each responsibility. We say "can be written" because once a CRC session is in full swing participants usually only need a few cards with the class name and virtually no cards written out in full.*

	A CRC session proceeds with someone simulating the system by talking about which objects send messages to other objects. By stepping through the process weaknesses and problems are easily uncovered. Design alternatives can be explored quickly by simulating the design being proposed." (http://www.extremeprogramming.org/)	
Discipline	Architecture	Yes
	Deployment	No
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance;
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing;
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	9	
Name	Spike solution	
Description	*"Create spike solutions to figure out answers to tough technical or design problems. A spike solution is a very simple program to explore potential solutions. Build the spike to only addresses the problem under examination and ignore all other concerns. Most spikes are not good enough to keep, so expect to throw it away. The goal is reducing the risk of a technical problem or increase the reliability of a user story's estimate." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	Yes
	Deployment	No
	Development	Yes
	Environment	No

	Project Management		No
	Requirements		No
	Test		Yes
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid	
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;	

Id	10	
Name	Refactor Mercilessly	
Description	*"When we remove redundancy, eliminate unused functionality, and rejuvenate obsolete designs we are refactoring. Refactoring throughout the entire project life cycle saves time and increases quality.* *Refactor mercilessly to keep the design simple as you go and to avoid needless clutter and complexity. Keep your code clean and concise so it is easier to understand, modify, and extend. Make sure everything is expressed once and only once. In the end it takes less time to produce a system that is well groomed.* *There is a certain amount of Zen to refactoring. It is hard at first because you must be able to let go of that perfect design you have envisioned and accept the design that was serendipitously discovered for you by refactoring. You must realize that the design you envisioned was a good guide post, but is now obsolete." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No

	Deployment	No
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	No
	Test	No

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing;
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;
	Technical Complexity	A – Homogenous; B - Multiple technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced;

Id	11
Name	The Customer is Always Available
Description	*"One of the few requirements of extreme programming (XP) is to have the customer available. Not only to help the development team, but to be a part of it as well. All phases of an XP project require communication with the customer, preferably face to face, on site. It's best to simply assign one or more customers to the development team. Beware though, this seems like a long time to keep the customer hanging and the customer's department is liable to try passing off a trainee as an expert. You need the expert. User Stories are written by the customer, with developers helping, to allow time estimates, and assign priority. The customers help make sure most of the system's desired functionality is covered by stories.* *Another Satisfied Customer During the release planning meeting the customer will need to negotiate a selection of user stories to be included in each scheduled release. The timing of the release may need to be negotiated as well. The customers must make the decisions that affect their business goals. A release planning meeting is used to define small incremental releases to allow functionality to be released to the customer early. This allows the*

	customers to try the system earlier and give the developers feedback sooner. *Because details are left off the user stories the developers will need to talk with customers to get enough detail to complete a programming task. Projects of any significant size will require a full time commitment from the customer.* *The customer will also be needed to help with functional testing. The test data will need to be created and target results computed or verified. Functional tests verify that the system is ready to be released into production. It can happen that the system will not pass all functional tests just prior to release. The customer will be needed to review the test score and allow the system to continue into production or stop it." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	Yes
	Development	No
	Environment	Yes
	Project Management	Yes
	Requirements	Yes
	Test	Yes
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused;

Id	12
Name	Unit Test First
Description	*"When you create your tests first, before the code, you will find it much easier and faster to create your code. The combined time it*

	takes to create a unit test and create some code to make it pass is about the same as just coding it up straight away. But, if you already have the unit tests you don't need to create them after the code saving you some time now and lots later. *Creating a unit test helps a developer to really consider what needs to be done. Requirements are nailed down firmly by tests. There can be no misunderstanding a specification written in the form of executable code.* *You also have immediate feedback while you work. It is often not clear when a developer has finished all the necessary functionality. Scope creep can occur as extensions and error conditions are considered. If we create our unit tests first then we know when we are done; the unit tests all run.* *There is also a benefit to system design. It is often very difficult to unit test some software systems. These systems are typically built code first and testing second, often by a different team entirely. By creating tests first your design will be influenced by a desire to test everything of value to your customer. Your design will reflect this by being easier to test.* *There is a rhythm to developing software unit test first. You create one test to define some small aspect of the problem at hand. Then you create the simplest code that will make that test pass. Then you create a second test. Now you add to the code you just created to make this new test pass, but no more! Not until you have yet a third test. You continue until there is nothing left to test. The coffee maker problem shows an example written in Java.* *The code you will create is simple and concise, implementing only the features you wanted. Other developers can see how to use this new code by browsing the tests. Input whose results are undefined will be conspicuously absent from the test suite."* *(http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	No
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	No
	Test	Yes
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging

	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	13		
Name	Pair Programming		
Description	*“All code to be sent into production is created by two people working together at a single computer. Pair programming increases software quality without impacting time to deliver. It is counter intuitive, but 2 people working at a single computer will add as much functionality as two working separately except that it will be much higher in quality. With increased quality comes big savings later in the project.* *The best way to pair program is to just sit side by side in front of the monitor. Slide the key board and mouse back and forth. Both programmers concentrate on the code being written.* *Pair programming is a social skill that takes time to learn. You are striving for a cooperative way to work that includes give and take from both partners regardless of corporate status. The best pair programmers know when to say, "let's try your idea first." Don't expect people to be good at it from the start. It helps if you have someone on your team with experience to show everyone what it should feel like.* *One thing pair programming is not is mentoring. A teacher-student relationship feels very different from two people working together as equals even if one has significantly more experience. It takes time to get used to pair programming so don't worry if it feels awkward at first.” (http://www.extremeprogramming.org/)*		
Discipline	Architecture		No
	Deployment		No
	Development		Yes
	Environment		Yes
	Project Management		No
	Requirements		No
	Test		Yes
Capability	Factor	Values	

	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home;
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused;

Id	14	
Name	Continuous integration	
Description	*"Developers should be integrating and committing code into the code repository every few hours, when ever possible. In any case never hold onto changes for more than a day. Continuous integration often avoids diverging or fragmented development efforts, where developers are not communicating with each other about what can be re-used, or what could be shared. Everyone needs to work with the latest version. Changes should not be made to obsolete code causing integration head aches.* *Each development pair is responsible for integrating their own code whenever a reasonable break presents itself. This may be when the unit tests all run at 100% or some smaller portion of the planned functionality is finished. Only one pair integrates at any given moment and after only a few hours of coding to reduce the potential problem location to almost nothing.* *Continuous integration avoids or detects compatibility problems early. Integration is a "pay me now or pay me more later" kind of activity. That is, if you integrate throughout the project in small amounts you will not find your self trying to integrate the system for weeks at the project's end while the deadline slips by. Always work in the context of the latest version of the system." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	No
	Development	Yes
	Environment	No
	Project Management	No

	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid	
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;	

Id	15
Name	Collective Code Ownership
Description	*"Collective Ownership encourages everyone to contribute new ideas to all segments of the project. Any developer can change any line of code to add functionality, fix bugs, improve designs or refactor. No one person becomes a bottleneck for changes.* *This is hard to understand at first. It's almost inconceivable that an entire team can be responsible for the system's design. Not having a single chief architect that keeps some visionary flame alive seems like it couldn't possibly work.* *But it is not uncommon to ask a chief architect a question and get an answer that is just plain wrong. It is not a failing of your lead programmers. Any non-trivial system cannot be held in one person's mind. Other programmers are hard at work changing the system without benefit of the architect's vision. Whether you realize it or not your design is already distributed among your team. If the entire team already has some responsibility for design decisions, shouldn't they receive the authority as well?*

	The way this works is for each developer to create unit tests for their code as it is developed. All code that is released into the source code repository includes unit tests that run at 100%. Code that is added, bugs as they are fixed, and old functionality as it is changed will be covered by automated testing. Now you can rely on the test suite to watch dog your entire code repository. Before any code is released it must pass the entire test suite at 100%. *Once this is in place anyone can make a change to any method of any class and release it to the code repository as needed. When combined with frequent integration developers rarely even notice a class has been extended or repaired.* *In practice collective ownership is actually more reliable than putting a single person in charge of watching specific classes. Especially since a person may leave the project at any time." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	Yes
	Development	Yes
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	Yes

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly Enterprise Focus

Id	16
Name	Unit Tests

Description	*"Unit tests are one of the corner stones of Extreme Programming (XP). But unit tests XP style is a little different. First you should create or download a unit test framework to be able to create automated unit tests suites. Second you should test all classes in the system. Trivial getter and setter methods are usually omitted. You will also create your tests first, before the code.* *Unit tests are released into the code repository along with the code they test. Code without tests may not be released. If a unit test is discovered to be missing it must be created at that time.* *The biggest resistance to dedicating this amount of time to unit tests is a fast approaching deadline. But during the life of a project an automated test can save you a hundred times the cost to create it by finding and guarding against bugs. The harder the test is to write the more you need it because the greater your savings will be. Automated unit tests offer a pay back far greater than the cost of creation.* *Another common misconception is that unit tests can be written in the last three months of the project. Unfortunately, without the unit tests development drags on and eats up those last three months and then some. Even if the time is available good unit test suites take time to evolve. Discovering all the problems that can occur takes time. In order to have a complete unit test suite when you need it you must begin creating the tests today when you don't.* *Unit tests enable collective ownership. When you create unit tests you guard your functionality from being accidentally harmed. Requiring all code to pass all unit tests before it can be released ensures all functionality always works. Individual code ownership is not required if all classes are guarded by unit tests.* *Unit tests enable refactoring as well. After each small change the unit tests can verify that a change in structure did not introduce a change in functionality.* *Building a single universal unit test suite for validation and regression testing enables frequent integration. It is possible to integrate any recent changes quickly then run your own latest version of the test suite. When a test fails your latest versions are incompatible with the team's latest versions. Fixing small problems every few hours takes less time than fixing huge problems just before the deadline. With automated unit tests it is possible to merge a set of changes with the latest released version and release in a short time.* *Often adding new functionality will require changing the unit tests to reflect the functionality.* *While it is possible to introduce a bug in both the code and test it rarely happens in actual practice. It does occasionally happen that the test is wrong, but the code is right. This is revealed when the problem is investigated and is fixed. Creating tests independent of code, hopefully before code, sets up checks and balances and greatly improves the chances of getting it right the first time.* *Unit Tests provide a safety net of regression tests and validation tests so that you can refactor and integrate effectively. As they say at the circus; never work without a net! Creating the unit test before the code helps even further by solidifying the requirements, improving developer focus, and avoid creeping elegance." (http://www.extremeprogramming.org/)*

Discipline	Architecture		No
	Deployment		No
	Development		Yes
	Environment		No
	Project Management		No
	Requirements		No
	Test		Yes
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid	
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;	

Id	17
Name	Acceptance tests
Description	*"Acceptance tests are created from user stories. During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests. The customer specifies scenarios to test when a user story has been correctly implemented. A story can have one or many acceptance tests, what ever it takes to ensure the functionality works.* *Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of*

	highest priority. Acceptance tests are also used as regression tests prior to a production release. *A user story is not considered complete until it has passed its acceptance tests. This means that new acceptance tests must be created each iteration or the development team will report zero progress.* *Quality assurance (QA) is an essential part of the XP process. On some projects QA is done by a separate group, while on others QA will be an integrated into the development team itself. In either case XP requires development to have much closer relationship with QA.* *Acceptance tests should be automated so they can be run often. The acceptance test score is published to the team. It is the team's responsibility to schedule time each iteration to fix any failed tests." (http://www.extremeprogramming.org/)*	
Discipline	Architecture	No
	Deployment	Yes
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	No
	Test	Yes

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	18	
Name	**Hazard Stories**	
Description	*Hazard stories are scenarios for possible safety violations, written in natural language much like User Stories.* *Procedure: Hazard Stories should be created at the planning stage and supplemented through the whole project development process. User Stories or Product Backlog features should be prepared before the Hazard Stories in order to outline the main objectives of the system. Methods for creating Hazard Stories can be similar to the methods known from writing User Stories; brainstorming should be useful. The team can assign roles, including "the devil's advocate". Where applicable Hazard Story should be linked to User Stories/Features it can violate or have impact on. This means that User Stories/Features ought to include non-functional requirements, especially safety-related. Based on this, priorities should be given to each Hazard Story – priorities can be taken straight from the linked User Stories/Features which means that the more important User Story/Feature it disrupts is, the more distress it can cause and should be dealt with sooner. They can provide a starting point for the safety related tests.*	
Discipline	Architecture	**No**
	Deployment	**No**
	Development	**No**
	Environment	**No**
	Project Management	**Yes**
	Requirements	**Yes**
	Test	**Yes**
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home;
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary;
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	19
Name	Iterative development

<table>
<tr><td>Description</td><td colspan="3">“The word "iterative" means that it involves repetition.
Iterative Development is a development approach that "cycles" through the development phases, from gathering requirements to delivering functionality in a working release.
Contrast this with the WaterfallModel, where you gather all the requirements up front, do all necessary design, down to a detailed level, then hand the specs to the coders, who write the code; then you do testing (possibly with a side trip to IntegrationHell) and deliver the whole thing in one big end-all release. Everything is big including the risk of failure.
Consider also IncrementalDelivery; (an XP page that may actually be talking about IterativeDevelopment) IncrementalDelivery also delivers functionality to users in cycles, but is historically less focused on reworking existing functionality. So a traditional "IncrementalDelivery" project will deliver one subsystem at a time to the end users, with "as little change as possible" to each subsystem, after it's delivered.
A key differentiation between the traditional WaterFall and the Iterative processes is how the project tasks are boxed in the plan. If they are functionally boxed you are probably WaterFalling, if they time-boxed you are probably Iterating.
The mantras of IterativeDevelopment are:
Phases are Time-Boxed not Functionally-Boxed.
Test early, Test often.
Deliver early, Deliver often.
Production Quality.
Iterative Development processes grew out of ObjectOrientedDevelopment where it quick appreciated that a Class could be considered a mini-project and developed in isolation, the task was naturally boxed by its responsibilities.
Some examples of Iterative Development Processes.
Agile (XP).
Iconix
OPEN.
RationalUnifiedProcess.
SelectPerspective.“
(http://wiki.c2.com/?IterativeDevelopment)</td></tr>
<tr><td>Discipline</td><td colspan="2">Architecture</td><td>No</td></tr>
<tr><td></td><td colspan="2">Deployment</td><td>Yes</td></tr>
<tr><td></td><td colspan="2">Development</td><td>Yes</td></tr>
<tr><td></td><td colspan="2">Environment</td><td>No</td></tr>
<tr><td></td><td colspan="2">Project Management</td><td>No</td></tr>
<tr><td></td><td colspan="2">Requirements</td><td>Yes</td></tr>
<tr><td></td><td colspan="2">Test</td><td>No</td></tr>
<tr><td>Capability</td><td>Factor</td><td colspan="2">Values</td></tr>
<tr><td></td><td>Team Size</td><td colspan="2">A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100’s of developers; E – 1000’s of developers</td></tr>
<tr><td></td><td>Geographical Distribution</td><td colspan="2">A – Co-located; B – Same building; C – Some working from home; D –</td></tr>
</table>

		Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused;

Id	20	
Name	Incremental development	
Description	*"Nearly all Agile teams favor an incremental development strategy; in an Agile context, this means that each successive version of the product is usable, and each builds upon the previous version by adding user-visible functionality. These are called "vertical" increments (that is, difference between successive product versions), as opposed to the opposite strategy which successively delivers complete technical components: for instance, creating a database schema, then building business rules on top of that, and only then implementing a UI. (This article offers a typical illustration of the distinction. It echoes the "layered cake" metaphor of software architecture: one can either cut along the horizontal layers, or vertically across them.) It is difficult to imagine an incremental approach in the Agile sense which is not also iterative, at least to some extent, but the two concepts are not identical. (They also prove surprisingly difficult to pin down, and are often the subject of heated semantic debates.)" (https://www.agilealliance.org/glossary/incremental-development/)*	
Discipline	Architecture	No
	Deployment	Yes
	Development	Yes
	Environment	No
	Project Management	No
	Requirements	Yes
	Test	Yes

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused;

Id	21
Name	Milestones
Description	*"Milestone* *(Alias: management gate)* *At 15 I set my heart upon learning.* *At 30 I established myself (in accordance with ritual).* *At 40 I no longer had perplexities.* *At 50 I knew the mandate of Heaven.* *At 60 I was at ease with what ever I heard.* *At 70 I could follow my heart's desire without transgressing the boundaries of right.* *- Confucius* *A milestone is a significant event in the course of a project that is used to give visibility of progress in terms of achievement of predefined milestone goals. Failure to meet a milestone indicates that a project is not proceeding to plan and usually triggers corrective action by management.* ***Setting Milestones:***

	Milestones have no duration, they represent instantaneous events that occur throughout the project. Typical project events that are marked with milestones are: • *The completion of a project phase* • *The approval of a deliverable* • *The completion of a scheduled review* • *The completion of an activity* • *The commencement of an activity.* ***Describing a Milestone*** ⇧ *In defining your Software Development Methodology (SDM) you should describe your milestones in terms of the following attributes:* • ***Milestone name.*** *A short catchy phrase or acronym that describes the milestone (Example: Preliminary Design Review - PDR)* • ***Project event.*** *A description of the project event to which the milestone is tied (Example: formal approval of the Preliminary Design by the PDR team)* • ***Milestone goals.*** *The criteria that will be applied to determine if the milestone has been reached - the milestone goal(s) (Example: Preliminary Design approval).* *When you apply the SDM to a specific project all your milestones should also be defined in terms of an absolute date or an elapsed time period from a previous milestone such as project commencement.* ***Setting Milestone Goals*** *Make sure your milestone goals are verifiable. You should be able to verify a goal with a yes or no answer. For example, the Software Requirements Specification is either complete or not complete, the Software Design Description has either passed or not passed peer review. The software product is or is not in commercial operation. Beware of fuzzy, unverifiable milestone goals that allow one phase of a project to merge into another resulting in loss of visibility and project control. For example, "high level design complete" is a fuzzy goal, "Software Architecture Design Description Document approved", is more verifiable.* *Typical milestone goals are:* • *Designated deliverables approved* • *Phase completion review complete* • *Tests complete and all anomalies rectified.* *The verification that a milestone has been reached usually occurs in the context of a management review. Milestones are therefore often named after major project review meetings. For example, Critical Design Review.* *In some situations the commencement of work can be a valid milestone, the principle being that you can't meet your end date if you haven't met your start date. Canny project managers often use this type of milestone when supervising remote sub-contractors. They prefer to know about late starts when action can be taken to lessen the impact of a late delivery on the project.“ (http://www.chambers.com.au/glossary/milestone.php)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No

	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; E – Heterogeneous/Legacy	
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;	
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;	

Id	22	
Name	Reflection workshop	
Description	*"Involves sitting down together, after any given milestone (e.g. after each iteration) or after critical events (key person leaving) and asking fundamentally 3 questions:* • *What did we like about what we were doing? (I.e., if we make changes, we should protect these)* • *What did we not like? (I.e., we'll nominate ways to avoid these)* • *What do we want to try differently this next time? (http://wiki.c2.com/?ReflectionWorkshop)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes

	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	23
Name	Burndown chart
Description	*"Burn-downs charts are among the most common sprint tracking mechanisms used by Agile practitioners. Though their application and usage varies (some plot a burn-down chart using story points, whereas others use task count), plotting burn-down using effort remaining is the most effective and efficient way of using burn-down charts. This article looks at creating and updating a burn-down chart using the effort-remaining approach, interpreting burn-down under different scenarios, and examining common mistakes to avoid while using burn-downs. We conclude by looking at some of the benefits of using this innovative tool.* ***How to create a burn-down chart*** *The first step is to have a task breakdown in place. This is generally done during the sprint planning meeting. Each task should have associated hours (ideally not more than 12, roughly two days' work at six per day), which the team decides on during the planning meeting.*

	Once the task breakdown is in place, the ideal burn-down chart is plotted. The ideal reflects progress assuming that all tasks will be completed within the sprint at a uniform rate (refer to the red line in Figure 1 below). *Many Agile tools (Rally, RTC, Version One, Mingle, etc.) have built-in capability for burn-down charts. However, in its simplest form, a burn-down chart can be maintained in a spreadsheet. Dates in the sprint are plotted on the X axis, while remaining efforts are plotted on the Y axis."* *(https://www.scrumalliance.org/community/articles/2013/august/burn-down-chart-%E2%80%93-an-effective-planning-and-tracki)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly

		enterprise focused; 5 – Enterprise focus;

Id	24
Name	Regression testing
Description	*"Whenever developers change or modify their software, even a small tweak can have unexpected consequences. Regression testing is testing existing software applications to make sure that a change or addition hasn't broken any existing functionality. Its purpose is to catch bugs that may have been accidentally introduced into a new build or release candidate, and to ensure that previously eradicated bugs continue to stay dead. By re-running testing scenarios that were originally scripted when known problems were first fixed, you can make sure that any new changes to an application haven't resulted in a regression, or caused components that formerly worked to fail. Such tests can be performed manually on small projects, but in most cases repeating a suite of tests each time an update is made is too time-consuming and complicated to consider, so an automated testing tool is typically required.* *Understanding Regression Tests* *Some software development teams try to get by without performing regular regression tests, opting to test essential functions just once to make sure they work and, if they check out, proceeding with the hopeful assumption that those functions will still work unless they're directly modified again. In a way, this makes sense: it's natural to want to simply make a change, test it, and move on. Performing functional tests or highly specific unit tests to determine that a new software component works as it should has been called "non-regression testing" by Doug Hoffman and others. But it can be relatively easy to find a specific problem when you're looking for it; what's harder is catching all the ones you don't expect.* *Again, it's important for developers and testers to always bear in mind that even small, seemingly insignificant alterations to an application's source code can ripple outward in surprising ways, breaking functions that seem completely unrelated to the new modification. When you run regression tests, you're checking to make sure that your modification not only behaves as you want it to, but that it also hasn't inadvertently caused problems in functions that had otherwise worked correctly when previously tested. Fortunately for the would-be regression tester, on any given project your regression test libraries can be built from the existing test cases developed from day one. Functional tests, unit tests,*

	integration tests, and build verification tests—anything that has successfully verified, throughout the development process, that various components work as intended—can all be incorporated into a regression testing suite, and "regression tests," per se, don't necessarily need to be written. Each time you modify your source code, you can simply re-run the potentially relevant tests to ensure that they continue to pass. Naturally, over the course of a complex development project, those test cases—and the various functions and processes that they attempt to check—can number in the thousands, making the use of automated testing software mandatory for full-scale regression tests. TestComplete can help you reuse your previous automated tests to easily create continuous regression tests." *(https://smartbear.com/learn/automated-testing/what-is-regression-testing/)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	No
	Requirements	No
	Test	Yes

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	25
Name	Sprint Planning
Description	*"The work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team.* *Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.* *Sprint Planning answers the following:* • *What can be delivered in the Increment resulting from the upcoming Sprint?* • *How will the work needed to deliver the Increment be achieved?* *Topic One: What can be done this Sprint?* *The Development Team works to forecast the functionality that will be developed during the Sprint. The Product Owner discusses the objective that the Sprint should achieve and the Product Backlog items that, if completed in the Sprint, would achieve the Sprint Goal. The entire Scrum Team collaborates on understanding the work of the Sprint.* *The input to this meeting is the Product Backlog, the latest product Increment, projected capacity of the Development Team during the Sprint, and past performance of the Development Team. The number of items selected from the Product Backlog for the Sprint is solely up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.* *After the Development Team forecasts the Product Backlog items it will deliver in the Sprint, the Scrum Team crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment.* *Topic Two: how will the chosen work get done?* *Having set the Sprint Goal and selected the Product Backlog items for the Sprint, the Development Team decides how it will build this functionality into a "Done" product Increment during the Sprint. The Product Backlog items selected for this Sprint plus the plan for delivering them is called the Sprint Backlog.* *The Development Team usually starts by designing the system and the work needed to convert the Product Backlog into a working product Increment. Work may be of varying size, or estimated effort. However, enough work is planned during Sprint Planning for the Development Team to forecast what it believes it can do in the upcoming Sprint. Work planned for the first days of the Sprint by the Development Team is decomposed by the end of this meeting, often to units of one day or less. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during Sprint Planning and as needed throughout the Sprint.* *The Product Owner can help to clarify the selected Product Backlog items and make trade-offs. If the Development Team*

	determines it has too much or too little work, it may renegotiate the selected Product Backlog items with the Product Owner. The Development Team may also invite other people to attend in order to provide technical or domain advice. *By the end of the Sprint Planning, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment.* *Sprint Goal* *The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting. The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint. The selected Product Backlog items deliver one coherent function, which can be the Sprint Goal. The Sprint Goal can be any other coherence that causes the Development Team to work together rather than on separate initiatives.* *As the Development Team works, it keeps the Sprint Goal in mind. In order to satisfy the Sprint Goal, it implements the functionality and technology. If the work turns out to be different than the Development Team expected, they collaborate with the Product Owner to negotiate the scope of Sprint Backlog within the Sprint." (Scrum Guide, https://www.scrumguides.org/scrum-guide.html#events-planning)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance;
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy

	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	26
Name	Sprint Review
Description	"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 a four-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Sprint Review includes the following elements: • Attendees include the Scrum Team and key stakeholders invited by the Product Owner; • The Product Owner explains what Product Backlog items have been "Done" and what has not been "Done"; • The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved; • The Development Team demonstrates the work that it has "Done" and answers questions about the Increment; • The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date (if needed); • The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning; • Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and, • Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product. The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities."

	(Scrum Guide, https://www.scrumguides.org/scrum-guide.html#events-planning)		
Discipline	Architecture		No
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		Yes
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance;	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies;	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;	
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;	

Id	27
Name	Sprint Retrospective
Description	"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. The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is a three-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Scrum Master participates as a peer team member in the meeting from the accountability over the Scrum process.

	The purpose of the Sprint Retrospective is to: • Inspect how the last Sprint went with regards to people, relationships, process, and tools; • Identify and order the major items that went well and potential improvements; and, • Create a plan for implementing improvements to the way the Scrum Team does its work. The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by adapting the definition of "Done" as appropriate. By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation." *(Scrum Guide, https://www.scrumguides.org/scrum-guide.html#events-planning)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance;
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies;
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;

	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	28		
Name	**SafeScrum Backlog splitting**		
Description	"*In SafeScrum, all requirements are split into safety critical requirements and other requirements and inserted into separate product backlogs. Alternatively, the safety requirements are tagged.* Adding a second backlog is an extension of the original Scrum process and is needed to separate the frequently changed functional requirements from the more stable safety requirements. With two backlogs we can keep track of how each item in the functional product backlog relates to the items in the safety product backlog, i.e. which safety requirements that are affected by which functional requirements. This can be done by using simple cross-references in the two backlogs and can also be supported with an explanation of how the requirements are related if this is needed to fully understand a requirement. The staffing of the *Sprint team* and the duration of the sprint (30 days is common), together with the estimates of each item decides which items that can be selected for development. Sometimes also e.g. the Safety responsible or the RAMS responsible takes part in the selection of which items have to be prioritized."		
Discipline	Architecture		No
	Deployment		No
	Development		Yes
	Environment		No
	Project Management		Yes
	Requirements		Yes
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	

	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned;
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused;

Id	29
Name	SafeScrum Quality Assurance Role
Description	*"The traditional approach in this and similar organizations would be to place the QA role as a specialized function in the line organization, outside the project. We decided, however, to add the QA role to the Scrum team to be close to the activities and to the information needed to execute quality assurance. This adds to the principle of cross functional teams [12] in Scrum. We are also considering making this a rotating role to make it a shared responsibility and to share the workload. We need a QA log to trace findings, decisions, corrective actions, and the follow up/results of these. In our case, Confluence is a good tool to establish this log. We have identified four tasks:* *QA Task 1: Check Code Metric Values for New or Changed Code: Neither the IEC61508 standard nor the assessor provide directions on specific metrics and limits to monitor at the component level. We have consulted the research literature [9] and used Minitab to analyze data from code from previous projects at the case company to define the following metrics and limits: (1) number of static paths – STPTH: 250, (2) McCabe's cyclomatic number – STCYC: 15, (3) number of parameters – STPAR: 5, (4) function call count – STSUB: 13, (5) maximum nesting of control structures – STMIF: 5, (6) number of executable lines – STXLN: 70, and (7) Myer's value – (STMCC): 10.* *QAC is used to analyze new and changed code to produce values for the metrics. This is done at the end of each sprint. The metrics are displayed, together with their defined maximum values in a radar plot. It is thus easy to if there are metrics that are exceeding their defined limits. If the values are inside their limits, QA will accept the code. If one or more values are outside their limits, the code is presented by QA in the sprint review meeting where the team decides to either accept the violation or plan refactoring. If the violation is accepted, a brief explanation must be added to the log and potentially also in the code (required by the standard). If the violation is unacceptable, the team needs to define a new task in Jira to refactor the code.* *QA Task 2: Check Documentation Coverage:* *Check new/changed code to ensure proper inline documentation and documents in Doxygen. This has to be done manually. In case of missing or poor documentation, the QA-log should be updated*

	and the findings should be discussed in the sprint review meeting to decide how to resolve it (giving a task to someone in the team). This check could be done at the end of each sprint. *QA Task 3: Check Test Coverage:* *Check for code coverage using Squish Coco. The QA log should be updated with references to uncovered code. This could be checked by the end of each sprint. Uncovered code should be discussed at the sprint review meeting and the team should define corrective actions, like defining tasks to produce tests. According to the standard, the coverage should be at least 99%.* *QA Task 4: Check Requirements-Task-Code Traceability:* *For new requirements, tasks, and code check that 1) requirements (RMsis) is linked to issues (e.g. in Jira), and 2) that code (e.g. Stash) is linked to issues (e.g. Jira). The QA role should control consistency at the end of the sprint and the team should resolve any identified issues immediately. The IEC61508 standard provides a set of explicit requirements for traceability; see table A.4 – Software design and development – detailed design, and table A.5 – Software design and development – software module testing and integration. We consulted the assessor about a definition of 'module' and its size (LOC). He referred to part 7, chapter C.2.9: "a software module should have a single well-defined task or function to fulfill." The assessor recommended 1000 LOC as an upper limit. In cases where the limit is exceeded it should be explained and documented.* *The tasks have been defined to be as simple and inexpensive as possible, partly by using the tool chain that already is in place. The goal is to not add more work to the process ceremony than strictly needed, but to use the QA role to simplify the sprint reviews so that they are not bogged down with unimportant details. So far, this is done by letting the QA close issues that have low risk and complexity on his own in advance of the sprint retrospective, where the whole team participates. This reduces the time spent on unimportant decisions and helps the team focus on difficult tasks where a joint evaluation and decision is needed.* *There should be a list of which criteria's the team members shall have done before the issue can be set as resolved and the QA needs to check that they have been fulfilled (not do them himself). The project should also create a list of which criteria's where the QA can close an issue without any further investigation. Only QA or the team can move an issue from resolved status to closed status. Closed issues cannot be changed." ("Quality Assurance in Scrum Applied to Safety Critical Software", Geir K. Hanssen, Børge Haugset, Tor Stålhane, Thor Myklebust, Ingar Kulbrandstad)*
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	Yes
Capability	Factor	Values

	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	30	
Name	SafeScrum Backlog Refinement	
Description	*"Important activities here are e.g., to improve user stories that are poorly written, estimate resources need for each backlog item, add acceptance criteria and to look deeper into the backlog to do longer-range technical planning. To these activities, we should add work needed to prepare for handling safety requirements in the next sprint, e.g. clarify safety-related requirements and decide whether we need to include customer representatives, domain experts or safety experts (depending on the backlog items) in the next sprint." (T. Stalhane, T. Myklebust, N. Lyngby, "The Agile Safety Case")*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	No

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers;
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	31
Name	PERT Analysis
Description	*"PERT is an estimating technique that uses a weighted average of three numbers (see below) to come up with a final estimate.* • *The most pessimistic (P) case when everything goes wrong* • *The most optimistic (O) case where everything goes right* • *The most likely (M) case given normal problems and opportunities* *The resulting PERT estimate is calculated as (O + 4M + P)/6. This is called a "weighted average" since the most likely estimate is weighted four times as much as the other two values. You'll notice that the final PERT estimate is moved slightly toward either the optimistic or pessimistic value - depending on which one is furthest from the most likely. Generally this ends up moving the final estimate toward the worst case, since the worst case value tends to be further out from the most likely that the optimistic number. For example, let's say you estimate a piece of work to most likely take 10 hours. The best case (everything goes right) is six hours. The worst case (everything goes wrong) is 26 hours. The PERT estimate is (6 + 4(10) + 26)/6. The answer is 72/6, or 12 hours. Notice that the number was pulled a little toward the far extreme of*

	the pessimistic estimate, but not by much, since the result is still weighted heavily toward the most likely value. *You can use the PERT estimates two ways. You can provide these three estimates for all activities in your schedule or you can only use the PERT formula for those activities that are of high risk. These are the ones where you're not really sure of the estimate so there's a wide variation between the optimistic and pessimistic values.* *Speaking of variation - if you subtract your pessimistic value from the optimistic value and divide the result by six, you would have the standard deviation, which is a measure of the volatility of the estimate. In our example above, the standard deviation would be 3.34 ((26 - 6) / 6). The larger this standard deviation is, the less confidence you have in your estimate, since it would mean you have a large range between the optimistic and pessimistic estimates. If the standard deviation was small, it would mean you were pretty confident in your estimate, since the optimistic and pessimistic estimates would be close..“* *(http://www.techrepublic.com/blog/it-consultant/use-pert-technique-for-more-accurate-estimates/)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100’s of developers; E – 1000’s of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid

	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	32		
Name	Pareto Analysis		
Description	*"Pareto analysis is a technique used to identify the few vital factors which have true impact on an outcome, result, or on measures of quality, satisfaction or performance. Separating these "vital few" from the "trivial many" is generally known as the "80/20" rule. Applying this rule to sources of problems would mean that, on average, 80 percent of the problems can be traced back to 20 percent of all key causes. Identifying the high-volume impact factors in terms of frequency of occurrence helps to determine the focus of problem solving energy. In addition, the frequency distribution of these "vital few" can set the order of priorities of action. The Pareto diagram graphically depicts these distributions in a representation similar to a bar graph or histogram, showing the frequency of occurrence in descending order from left to right.* *Purpose* *The purpose of pareto analysis technique is to provide a means for identifying the key factors that have the highest occurrences in any given situation, to enable prioritization and focus of problem solving efforts, to aid in the analysis of the current situation with respect to activity performance, service, or quality and to monitor the effectiveness of solutions once implemented.* *Benefits* *The benefit of pareto analysis is that enables attention to be focused on the "vital few" issues that have the greatest effect and it predicts the effectiveness or outcome of resolving these issues." (https://www.projectmanagement.com/process/popup.cfm?ID=23789)*		
Discipline	Architecture		No
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working	

	from home; D – Within driving distance; E – Globally distributed
Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	33
Name	Failure Mode and Effects Analysis (FMEA)
Description	*"FMEA is a qualitative and systematic tool, usually created within a spreadsheet, to help practitioners anticipate what might go wrong with a product or process. In addition to identifying how a product or process might fail and the effects of that failure, FMEA also helps find the possible causes of failures and the likelihood of failures being detected before occurrence.* *Used across many industries, FMEA is one of the best ways of analyzing potential reliability problems early in the development cycle, making it easier for manufacturers to take quick action and mitigate failure. The ability to anticipate issues early allows practitioners to design out failures and design in reliable, safe and customer-pleasing features.* ***Finding Failure Modes*** *One of the first steps to take when completing an FMEA is to determine the participants. The right people with the right experience, such as process owners and designers, should be involved in order to catch potential failure modes. Practitioners also should consider inviting customers and suppliers to gather alternative viewpoints.* *Once the participants are together, the brainstorming can begin. When completing an FMEA, it's important to remember Murphy's Law: "Anything that can go wrong, will go wrong." Participants need to identify all the components, systems, processes and functions that could potentially fail to meet the required level of quality or*

	reliability. The team should not only be able to describe the effects of the failure, but also the possible causes. *An FMEA uses three criteria to assess a problem: 1) the severity of the effect on the customer, 2) how frequently the problem is likely to occur and 3) how easily the problem can be detected.* *Participants must set and agree on a ranking between 1 and 10 (1 = low, 10 = high) for the severity, occurrence and detection level for each of the failure modes. Although FMEA is a qualitative process, it is important to use data (if available) to qualify the decisions the team makes regarding these ratings* ***Setting Priorities*** *Once all the failure modes have been assessed, the team should adjust the FMEA to list failures in descending RPN order. This highlights the areas where corrective actions can be focused. If resources are limited, practitioners must set priorities on the biggest problems first.* *There is no definitive RPN threshold to decide which areas should receive the most attention; this depends on many factors, including industry standards, legal or safety requirements, and quality control. However, a starting point for prioritization is to apply the Pareto rule: typically, 80 percent of issues are caused by 20 percent of the potential problems. As a rule of thumb, teams can focus their attention initially on the failures with the top 20 percent of the highest RPN scores.* ***Making Corrective Actions*** *When the priorities have been agreed upon, one of the team's last steps is to generate appropriate corrective actions for reducing the occurrence of failure modes, or at least for improving their detection. The FMEA leader should assign responsibility for these actions and set target completion dates.* *Once corrective actions have been completed, the team should meet again to reassess and rescore the severity, probability of occurrence and likelihood of detection for the top failure modes. This will enable them to determine the effectiveness of the corrective actions taken. These assessments may be helpful in case the team decides that it needs to enact new corrective actions.* *The FMEA is a valuable tool that can be used to realize a number of benefits, including improved reliability of products and services, prevention of costly late design changes, and increased customer satisfaction."* *(https://www.isixsigma.com/tools-templates/fmea/quick-guide-failure-mode-and-effects-analysis/)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	Yes
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to

		100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	34
Name	Zurich Hazard Analysis (ZHA)
Description	*"The ZHA is straightforward and simple.* *The steps of the process are outlined below:* *Define the scope* *The scope of the ZHA describes both the content and the boundaries of the product, process or components to be analyzed. A realistically delineated scope takes into consideration the expertise and time available for the analysis.* *Choose the team and teamleader* *Ideally, the people most knowledgeable about the product, process or components to be analyzed are selected to participate as team mem bers. A teamleader is needed to facilitate the ZHA process. This is the only person on the team who really must understand the ZHA process well.* *Identify hazards, define and assess hazard scenarios* *Scenarios are developed as the team brainstorms through a 'Pathway', a defined route through the scope of the analysis. The ZHA 'Tickler List', a set of thought provoking words that encourage the team to systematically focus on each type of potential threat, is*

	used to stimulate the thought process and to ensure that hazards are revealed. Once a hazard scenario is identified and documented it is then rated for relative severity and frequency. Both elements determine the significance of the risk. *Build the risk profile, set the risk tolerance boundary and plot the risks* *The risk profile is a matrix divided into four severity categories and six probability levels. From a risk management point of view, the most important aspect of the risk profile is the risk tolerance boundary – a segmented line drawn across the risk profile. The risk tolerance boundary establishes the border between tolerable and intolerable risks. Exactly where that line is drawn on the matrix depends on your company's risk management strategy. The risks categorized in the hazard catalog are then plotted onto the corresponding coordinates of the current risk profile. The Risk Profile thus provides you with an instant graphic overview of your risk exposure.* *Develop risk improvement actions* *Improvement actions are devised to eliminate or reduce the risks above the tolerance boundary. The risk scenarios are re-ranked to take into account the improvement actions. This, then, defines the target risk Profile.* *Implement the risk improvements* *The risk improvement actions developed by the team provide concise direction as to how, when and by whom they should be implemented.* *Review the analysis* *Given that legal requirements, industrial standards, as well as consumer expectations – and indeed your own processes – continue to change, it will be necessary to periodically review the results of the analysis."* *("Methodology. Zurich Hazard Analysis: Revealing it all")*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	Yes
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed

Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	35
Name	Risk Matrix
Description	*"The Risk Matrix is also popularly known as the Probability and Impact Matrix. The Risk Matrix is used during Risk Assessment and is born during Qualitative Risk Analysis in the Risk Management process. It is a very effective tool that could be used successfully with Senior Management to raise awareness and increase visibility of risks so that sound decisions on certain risks can be made in context.* *A risk is "rated" for its Probability and Impact on a scale to understand where on the Risk Matrix it lies. Which Risks in the process move forward into the Risk Management process will depend on the industry, company, project and people. Some are by nature more risk tolerant than others. For example, we can have a project where the team agrees that any risk that is in the yellow, orange or red cell can move forward in the Risk Management process. The rest remain in the watch list and are "accepted". Another project could have different criteria.* *The Probability and Impact "Scales":* *It is recommended that companies have standard scales on their projects for how risks can be rated. This will help everyone be on the same page. A sample scale is provided below.* *Probability: A scale of 1%-100% will be used for Probability.* *(1-20)% means very low* *(21-40)% means low* *(41-60)% means medium* *(61-80)% means high* *(81-100)% means it is a fact*

	Impact: A scale of 1-5 is normally used for impact ratings where; *1 means negligible* *2 means minor* *3 means moderate* *4 means significant* *5 means severe* *The above image shows the Threat as well as the Opportunity Matrices in Vue-Matrix, our Project Risk Management Software application.* *The numbers in each cell indicates the number of risks in that cell. These cells can be drilled down into to view the risks directly from the matrix itself."* *(http://network.projectmanagers.net/profiles/blogs/what-is-a-risk-matrix)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	No

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	36	
Name	Failure Modes, Effects and Criticality Analysis (FMECA)	
Description	"**A Failure Mode, Effects, and Criticality Analysis (FMECA)** uses an inductive approach to system design and reliability. It identifies each potential failure within a system or manufacturing process and uses severity classifications to show the potential hazards associated with these failures. FMECA began with the standard developed by the United States Military, MIL-STD-1629, Procedures for Performing a Failure Mode, Effects and Criticality Analysis. This procedure was developed as a reliability technique to determine the effect of system and equipment failures. ISO 9000, and IEC 61508 methodologies are other and more modern approaches to performing this kind of analysis. There are two popular approaches to performing a FMECA: • The functional approach is applied in projects containing components that cannot be uniquely identified. In this scenario, the sub-system functions are weighed in terms of their function within the system. • The hardware/software approach is applied in projects containing components that can be uniquely identified. A FMECA is usually applied in two steps: • Identifying failure modes and their effects (FMEA). • Ranking failure modes according to the combination of severity and the probability of that failure mode occurring (Criticality Analysis). FMECA can be performed at any stage of system design. The results from a FMECA are maximized if the analysis is implemented during the early development stages and updated throughout the development. This approach also helps to educate system engineers about the system. Performing FMECA near the end of the design process minimizes the influence on the system design. FMECAs can take many forms, but at the core, these analyses are used to study a particular system and determine how that system can be modified to improve overall reliability and to avoid failures. For example, consider a simple FMECA that contains a computer monitor which has a capacitor as a component. By analyzing the circuit design, you determine that if that capacitor was open (one failure mode), the display would appear with wavy lines (the failure effect). If the capacitor was shorted (a second failure mode), the monitor would go blank. The second failure would be ranked as more critical than the first because the monitor becomes completely unusable. Once FMECA has identified failures, you can explore ways to prevent the failure or to lessen their criticality. Design and Process FMEAs can also be constructed if the analysis is centered around a process or other non-hardware system under consideration." (http://www.reliabilityeducation.com/intro_fmeca.html) Good source also here: www2.warwick.ac.uk/fac/sci/wmg/ftmsc/modules/.../section_12a_fmeca_notes.pdf	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	Yes

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	37
Name	Fault Tree Analysis (FTA)
Description	"The fault tree analysis (FTA) was first introduced by Bell Laboratories and is one of the most widely used methods in system reliability, maintainability and safety analysis. It is a deductive procedure used to determine the various combinations of hardware and software failures and human errors that could cause undesired events (referred to as top events) at the system level. The deductive analysis begins with a general conclusion, then attempts to determine the specific causes of the conclusion by constructing a logic diagram called a fault tree. This is also known as taking a top-down approach.

	The main purpose of the fault tree analysis is to help identify potential causes of system failures before the failures actually occur. It can also be used to evaluate the probability of the top event using analytical or statistical methods. These calculations involve system quantitative reliability and maintainability information, such as failure probability, failure rate and repair rate. After completing an FTA, you can focus your efforts on improving system safety and reliability. **FTA logic diagram** The basic symbols used in an FTA logic diagram are called logic gates and are similar to the symbols used by electronic circuit designers. The partial FTA logic diagram in Figure 1 uses the "and" and "or" gates' symbols to analyze hazard to the patient. Inputs to the "or" gate at the top identify the four reasons this failure can occur. One of the reasons, electrical shock, is then broken down because it results from simultaneously grounding the patient and creating a pathway to a current source (an "and" gate). The analysis continues on, using the same technique, until the lowest levels such as operator error or open ground pin are identified. When you perform an FTA, you systematically determine what happens to the system when the status of a part or another factor changes. In some applications, the minimum criterion for success is that no single failure can cause injury or an undetected loss of control over the process. In others, where extreme hazards exist or when high value product is being processed, the criteria may be increased to require toleration of multiple failures. **Fault tree construction** To do a comprehensive FTA, follow these steps: 1. Define the fault condition, and write down the top level failure. 2. Using technical information and professional judgments, determine the possible reasons for the failure to occur. Remember, these are level two elements because they fall just below the top level failure in the tree. 3. Continue to break down each element with additional gates to lower levels. Consider the relationships between the elements to help you decide whether to use an "and" or an "or" logic gate. 4. Finalize and review the complete diagram. The chain can only be terminated in a basic fault: human, hardware or software. 5. If possible, evaluate the probability of occurrence for each of the lowest level elements and calculate the statistical probabilities from the bottom up.“ *(http://asq.org/quality-progress/2002/03/problem-solving/what-is-a-fault-tree-analysis.html)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No

	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	38
Name	HAzard and Operability Analysis (HAZOP)
Description	*"Hazard and Operability Analysis (HAZOP) is a structured and systematic technique for system examination and risk management. In particular, HAZOP is often used as a technique for identifying potential hazards in a system and identifying operability problems likely to lead to nonconforming products. HAZOP is based on a theory that assumes risk events are caused by deviations from design or operating intentions. Identification of such deviations is facilitated by using sets of "guide words" as a systematic list of deviation perspectives. This approach is a unique feature of the HAZOP methodology that helps stimulate the imagination of team members when exploring potential deviations.* • *As a risk assessment tool, HAZOP is often described as:* • *A brainstorming technique* • *A qualitative risk assessment tool*

	• *An inductive risk assessment tool, meaning that it is a "bottom-up" risk identification approach, where success relies on the ability of subject matter experts (SMEs) to predict deviations based on past experiences and general subject matter expertise"* (pqri.org/wp-content/uploads/2015/08/pdf/HAZOP_Training_Guide.pdf)	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	No

Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	39
Name	Cause Consequence Analysis (CCA)
Description	"Cause-consequence analysis (CCA) is a method for analyzing consequence chains and can be used individually or as a supportive method for other analysis

	methods. The objective of the analysis is to recognize consequence chains developing from failures or other unwanted events, and to estimate these consequences with their probabilities. The cause-consequence structure of the analysis is formed by combining two different types of tree structures together. To the consequence tree, built from left to right, includes the examined primary event and its follow-up events leading eventually to a failure or some other unwanted event like for example a serious injury of a person. The causes and the probabilities for the realization of the primary event and the follow-up events are defined to cause trees built from top to down. Often cause trees describe failures and are therefore called fault trees. The top level of the cause tree is at the same time a node in the consequence tree describing an event realizing or not. Cause and consequence tree together create a visual consequence chain to help illustrate the relations between causes and consequences that lead into different damages. Consequence tree shows the possible consequence chains and damages of a single event, whereas cause trees (fault trees) describe the causes and probabilities of each consequence. Cause-consequence analysis includes the following phases: 1. Recognizing damage chains 2. Recognizing the primary event (failure or some unwanted event that triggers the damage chain) 3. Recognizing the follow-up events (events between primary event and final damages) 4. Final consequence damages (damages coming from different levels of follow-up events) 5. Defining causes of primary and follow-up events to cause/fault trees 6. Inputting realization probabilities (failure data) for the causes of primary and follow-up events Cause-consequence analysis is an effective tool when confirming that the operational safety features have been taken into account already on the design phase. The method can be applied especially when examining complex event chains where there are many possible consequence damages for a single primary event. The results of cause-consequence analysis include among other things: • Visual and logical description of the consequence chain evolving from the examined primary event • Probabilities for the final consequence damages based on the cause-consequence structure • Cause-consequence relations (causalities) between events • Requirements for the safety features“ (http://www.ramentor.com/theory/cause-consequence-analysis/)	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100’s of developers; E – 1000’s of developers

	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	40
Name	Management Oversight Risk Tree (MORT)
Description	*"The Management Oversight and Risk Tree (MORT) can be used to guide an investigation or as a tool to ensure inclusion of all applicable elements. It can also be used to preclude investigation of elements that have no bearing on the subject being evaluated. The MORT Chart lists things to consider regarding hazards, targets, barriers, controls, hazard analysis, and other components of the operation of a system. It also considers risk assessment, information systems, policies, policy implementation practices and procedures, management services and attitudes, fitness for duty concerns, and all the other components of management control systems that go into making the work place safe for employees. The old MORT Chart was developed for the nuclear industry and was quite cumbersome. It took a lot of use (years) to become proficient. The new MORT Chart has section labeling, color coding, guide arrows, readable type size, and properly defined usage guidance that relegates the chart to a support function to ensure that the investigation does not overlook any applicable deficiencies in the system. The chart can be explained in about half an hour, and with about an hour of practice, most people can see how simple and powerful the chart actually is. In addition, the chart has been divided into twenty-one areas and printed in easily reproducible form as cut-sheets. The cut-sheets may be the most effective and easiest way to use the MORT chart as a graphical checklist. It may be found that*

	an area on the MORT chart is applicable to several different people, objects, or procedures. It would be inconvenient to use several complete MORT charts. When a smaller area such as Fitness for Duty is needed for more than one person, the MORT Topical Cut Sheet for Fitness for Duty can be laid out on a duplicating machine and multiple copies can be reproduced for repeated use on the different persons.“ (http://www.acc-inv.com/SSDC%20Information/what_is_this_thing_called_mort.htm)	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100’s of developers; E – 1000’s of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	41

Name	Cost Benefit Analysis	
Description	*"What is a 'Cost-Benefit Analysis'* *A cost-benefit analysis is a process by which business decisions are analyzed. The benefits of a given situation or business-related action are summed, and then the costs associated with taking that action are subtracted. Some consultants or analysts also build the model to put a dollar value on intangible items, such as the benefits and costs associated with living in a certain town, and most analysts will also factor opportunity cost into such equations.* *BREAKING DOWN 'Cost-Benefit Analysis'* *Prior to erecting a new plant or taking on a new project, prudent managers conduct a cost-benefit analysis as a means of evaluating all the potential costs and revenues that may be generated if the project is completed. The outcome of the analysis will determine whether the project is financially feasible or if another project should be pursued.* *The Cost-Benefit Analysis Process* *The first step in the process is to compile a comprehensive list of all the costs and benefits associated with the project or decision. Costs should include direct and indirect costs, intangible costs, opportunity costs and the cost of potential risks. Benefits should include all direct and indirect revenues and intangible benefits, such as increased production from improved employee safety and morale, or increased sales from customer goodwill. A common unit of monetary measurement should then be applied to all items on the list. Care should be taken to not underestimate costs or overestimate benefits. A conservative approach with a conscious effort to avoid any subjective tendencies when calculating estimates is best suited when assigning value to both costs and benefits for the purpose of a cost-benefit analysis.* *The final step is to quantitatively compare the results of the aggregate costs and benefits to determine if the benefits outweigh the costs. If so, then the rational decision is to go forward with project. In not, a review of the project is warranted to see if adjustments can be made to either increase benefits and/or decrease costs to make the project viable. If not, the project may be abandoned.* *Limitation of Cost-Benefit Analysis* *For projects that involve small to mid-level capital expenditures and are short to intermediate in terms of time to completion, an in-depth cost-benefit analysis may be sufficient enough to make a well-informed rational decision. For very large projects with a long-term time horizon, cost-benefit analysis typically fails to effectively take into account important financial concerns such as inflation, interest rates, varying cash flows and the present value of money. Alternative capital budgeting analysis methods including net present value (NPV) or internal rate of return (IRR) are more appropriate for these situations."* *(http://www.investopedia.com/terms/c/cost-benefitanalysis.asp)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes

	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid	
	Enterprise Discipline	2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;	

Id	42
Name	Safety and Risk Evaluation using Bayesian Nets (SERENE)
Description	"The SERENE method addresses the lack of quantification of safety in the current standards-based approach. The overall objective of the project was to use Bayesian Belief Nets (BBNs) to reason about the safety of Programmable Electronic Systems (PES). This objective has been addressed by developing a method, the SERENE Method, for representing a PES safety arguments using BBNs and by enhancing an existing BBN tool with features needed to support the method. In the SERENE Method, a safety argument is a prediction of one or more properties of a system relevant to safety. The prediction is based on all the relevant evidence. In addition to the overall purpose of the safety argument the SERENE method satisfies a number of secondary objectives: 1. rationally combine different sources and types of evidence in a single model 2. identify weaknesses in the argument such that it can be improved

	3. identify weaknesses in products and processes to aid process improvement 4. specify degrees of confidence associated with predictions 5. provide a sound basis for rational discussion and negotiation about the systems development and deployment." (https://www.eecs.qmul.ac.uk/~norman/papers/serene.pdf)	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	No

Capability	Factor	Values
	Team Size	B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	43
Name	Risk Assessment and Method Statements (RAMS)
Description	*"A 'Risk Assessment and Method Statement' (RAMS), is a safety management document required for activities where there a significant safety risks. A RAMS must be activity-specific and not generic.* *The '**Risk Assessment**' identifies safety control measures that form the basis of a safe system of work and the '**Method Statement**' is a written record of that safe system of work for a specific activity developed from the relevant risk assessments.*

	Together they form the RAMS. *RAMS are only required for activities with significant risks. All managers, supervisors and team leaders with staff who are involved in activities with significant safety risks, need to ensure RAMS are developed for these activities.* *If you are a manager, supervisor or team leader with staff who are involved in activities with significant risks, you need to ensure risk assessments are completed for the tasks which form part of these activities and there needs to be* ***Risk Assessment and Method Statements*** *(RAMS) for these activities.* *Using the Risk Assessment and Method Statement templates will provide a guide for developing a RAMS. However, RAMS need to be activity-specific and consider unique aspects for the individual activity such as location and adjacent works. Using 'generic' RAMS are not acceptable.* ***› The benefits of a RAMS*** *The benefit of a RAMS is that it provides clear information and instruction to staff, provides a standard by which the work can be monitored and enables lapses in the safety standards to be identified.* *A RAMS also enables managers to ensure the resources necessary to do an activity safely are to hand and that everyone knows what to do in an emergency situation. This contributes to a safer site by ensuring everyone is aware of areas of risk where accidents may happen and the actions required to prevent them from happening."* *(http://www.ucl.ac.uk/estates/safety-and-sustainability/topics/risk-assessment-and-method-statement/index.php)*	
Discipline	Architecture	No
	Deployment	No
	Development	No
	Environment	No
	Project Management	Yes
	Requirements	Yes
	Test	Yes
Capability	Factor	Values
	Team Size	B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D

		– Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	44		
Name	UML Behaviour Diagrams		
Description	"**Behavior diagrams** show the **dynamic behavior** of the objects in a system, which can be described as a series of changes to the system over **time**." *(http://www.uml-diagrams.org/uml-25-diagrams.html#behavior-diagram)* • *Activity diagram* • *Sequence diagram* • *Use case diagram* • *State diagram* • *Communication diagram* • *Interaction overview diagram* • *Timing diagram*		
Discipline	Architecture		Yes
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		Yes
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	

	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	45		
Name	Risk checklists (PRINCE2)		
Description	*These are in-house lists of risks that have either been identified or have occurred on previous similar projects. Risk checklists are useful aids to ensure that risks identified on previous projects are not overlooked. (https://www.prince2primer.com/prince2-techniques-for-risk-assessment/)*		
Discipline	Architecture		No
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		Yes
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100’s of developers;	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	A – Straightforward; B - Predictable; D – Complicated;	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D	

		– Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	46		
Name	Risk breakdown structure (PRINCE2)		
Description	*This is a hierarchical decomposition of the project environment assembled to illustrate potential sources of risk. Each descending level represents an increasingly detailed definition of sources of risk to the project.* *The structure acts as a prompt and an aid to support the project management team in thinking through the potential sources of risk to the objectives.* *(https://www.prince2primer.com/prince2-techniques-for-risk-assessment/)*		
Discipline	Architecture		No
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	

	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	47	
Name	UML Structure Diagrams	
Description	*"**Structure diagrams** show **static structure** of the system and its parts on different abstraction and implementation levels and how those parts are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts. Structure diagrams are not utilizing **time** related concepts, do not show the details of dynamic behavior. However, they may show relationships to the behaviors of the classifiers exhibited in the structure diagrams."* *(http://www.uml-diagrams.org/uml-25-diagrams.html#behavior-diagram)* • *Class diagram* • *Package diagram* • *Object diagram* • *Component diagram* • *Composite structure diagram* • *Deployment diagram*	
Discipline	Architecture	Yes
	Deployment	No
	Development	Yes
	Environment	No
	Project Management	Yes
	Requirements	No
	Test	No
Capability	Factor	Values
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D –

		Within driving distance; E – Globally distributed
	Domain Complexity	A – Straightforward; B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy
	Organisational Complexity	1 – Flexible, intuitive; 2 – Flexible, structured; 3 – Stable, evolutionary; 4 – Stable, planned; 5 – Rigid
	Enterprise Discipline	1 – Project focus; 2 – Mostly project focused; 3 – Balanced; 4 – Mostly enterprise focused; 5 – Enterprise focus;

Id	48
Name	Threat modelling
Description	The technical steps in threat modelling involve answering questions: - What are we working on - What can go wrong - What will we do with the findings - Did we do a good job? The work to answer these questions is embedded in some sort of process, ranging from incredibly informal Kanban with Post-its on the wall to strictly structured waterfalls. The effort, work, and timeframes spent on threat modelling relate to the process in which engineering is happening and products/services are delivered. The idea that threat modelling is waterfall or 'heavyweight' is based on threat modelling approaches from the early 2000s. Modern threat modelling building blocks fit well into agile and are in wide use. *When to threat model* When the system changes, you need to consider the security impact of those changes. Sometimes those impacts are not obvious. Threat modelling integrates into Agile by asking "what are we working on, now, in this sprint/spike/feature?"; trying to answer this can be an important aspect of managing security debt, but trying to address it per-sprint is overwhelming. When the answer is that the system's architecture isn't changing, no new processes or dataflows are being introduced, and there are no changes to the data structures being transmitted, then it is unlikely that the answers to 'what can go wrong' will change. When one or more of those changes, then it's useful to examine what can go wrong as part of the current work package, and to understand designs trade-offs you can make, and to understand

	what you're going to address in this sprint and in the next one. The question of did we do a good job is split: the "did we address these threats" is part of sprint delivery or merging, while the broader question is an occasional saw-sharpening task. After a security incident, going back and checking the threat models can be an important process. *Threat modelling: engagement versus review* Threat modelling at a whiteboard can be a fluid exchange of ideas between diverse participants. Using the whiteboard to construct a model that participants can rapidly change based on identified threats is a high-return activity. The models created there (or elsewhere) can be meticulously transferred to a high-quality archival representation designed for review and presentation. Those models are useful for documenting what's been decided and sharing those decisions widely within an organization. These two activities are both threat modelling, yet quite different. (The Open Web Application Security Project, https://www.owasp.org/index.php/Main_Page)	
Discipline	Architecture		Yes
	Deployment		No
	Development		No
	Environment		No
	Project Management		No
	Requirements		No
	Test		Yes
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual	
	Technical Complexity	A – Homogenous; B - Multiple technology; C – New technology; D - System/embedded solutions; E – Heterogeneous/Legacy	
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid	

	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	49		
Name	Prototyping		
Description	*Bernhard Boar [Boar 83] has definedprototyping as a specific strategy for performingrequirements definitions wherein user needs areextracted, presented, and successively refined bybuilding a working model of the ultimate systemquickly and in its working context. A useful definition is given by Connell andShafer [Connell 89]: "A software prototype is adynamic visual model providing a communi-* *cation tool for customer and developer that is farmore effective than either narrative prose or staticvisual models for portraying functionality. It hasbeen described as:* *• functional after a minimal amount of effort* *• a means for providing users of a proposedapplication with a physical representation ofkey parts of the system before systemimplementation* *• flexible modifications require minimal effort* *• not necessarily representative of a complete system."* *This definition is deliberately general. Its purposeis to establish that experimenting with models isvery useful in the development of large softwaresystems and that the general proceeding should besimilar to that of development in other technical areas.* *(G. Pomberger, R. Weinreich, "The Role of Prototyping in Software Development")*		
Discipline	Architecture		Yes
	Deployment		No
	Development		Yes
	Environment		No
	Project Management		No
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	A – Under 10 developers; B – From 10 to 50 developers; C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	C – Quickly changing; D – Complicated; E – Intricate/Emerging	

	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies; E – Contractual
	Technical Complexity	B - Multiple technology; C – New technology; D - System/embedded solutions;
	Organisational Complexity	A – Flexible, intuitive; B – Flexible, structured; C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	A – Project focus; B – Mostly project focused; C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;

Id	50		
Name	Business analysis		
Description	***Business Analysis*** *is the practice of enabling change in an organizational context, by defining needs and recommending solutions that deliver value to stakeholders. The set of tasks and techniques that are used to perform* ***business analysis*** *are defined in A Guide to the* ***Business Analysis*** *Body of Knowledge® (BABOK®Guide)* (www.iiba.org/Careers/What-is-Business-Analysis.aspx)		
Discipline	Architecture		No
	Deployment		No
	Development		No
	Environment		No
	Project Management		Yes
	Requirements		No
	Test		No
Capability	Factor	Values	
	Team Size	C – From 50 to 100 developers; D – 100's of developers; E – 1000's of developers	
	Geographical Distribution	A – Co-located; B – Same building; C – Some working from home; D – Within driving distance; E – Globally distributed	
	Domain Complexity	B - Predictable; C – Quickly changing; D – Complicated; E – Intricate/Emerging	
	Organisational Distribution	A – Collaborative; B – Different teams; C – Different departments; D – Different partner companies;	
	Technical Complexity	B - Multiple technology; C – New technology; D - System/embedded	

		solutions; E – Heterogeneous/Legacy
	Organisational Complexity	C – Stable, evolutionary; D – Stable, planned; E – Rigid
	Enterprise Discipline	C – Balanced; D – Mostly enterprise focused; E – Enterprise focus;