AGILIS HÁLÓZATI SZOLGÁLTATÁS-FEJLESZTÉS – NETWORK SERVICE DEVELOPMENT IN AGILE WAY

VITMMA01

PRESENTERS

› Adamis Gusztáv leader of the subject – IE345
adamis@tmit.bme.hu

› Csöndes Tibor Ericsson
csondes@tmit.bme.hu

› Kovács Gábor – IL106a
kovacsg@tmit.bme.hu

2

KÖVETELMÉNYEK

- Zárthelyi: 2017. április 26.
- Csoportmunka
  - 3 feladat - 3 csoport
  - Folyamatos értékelés; 3 sprint
  - Az értékelésnél elsősorban nem a kódminőséget vesszük figyelembe, hanem azt, hogy a tanult módszereket mennyire használjátok
- Aláírás megszerzése: ZH + Csoportmunka mindegyike legalább 2
- Pótlás:
  - Pótzh (május 3. + pótlási héten – csak az egyik vehető igénybe)
  - Csoportmunka – nem pótolható
- Megajánlott jegy: ZH + Csoportmunka átlaga >= 4

5

IDŐBEOSZTÁS

- Előadás – szerdánként 8:30-10:00 QBF10
- Gyakorlat – páratlan heteken pénteken 8:30-10:00 QBF11
- A valóságban nem ilyen merev
  - Csapatok megalakulása, feladatok ismertetése
  - 3 sprint: tervezés, demó, értékelés
  - ZH, PótZH
- Órabeosztás és fóliák: https://www.tmit.bme.hu/vitmma01-2017

6

REFERENCES

The Art of Agile Development, James Shore & Shane Warden

Foundations of Software Testing, Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black

Foundation Level Extension Syllabus, Agile Tester, 2014, International Software Testing Qualification Board (ISTQB)

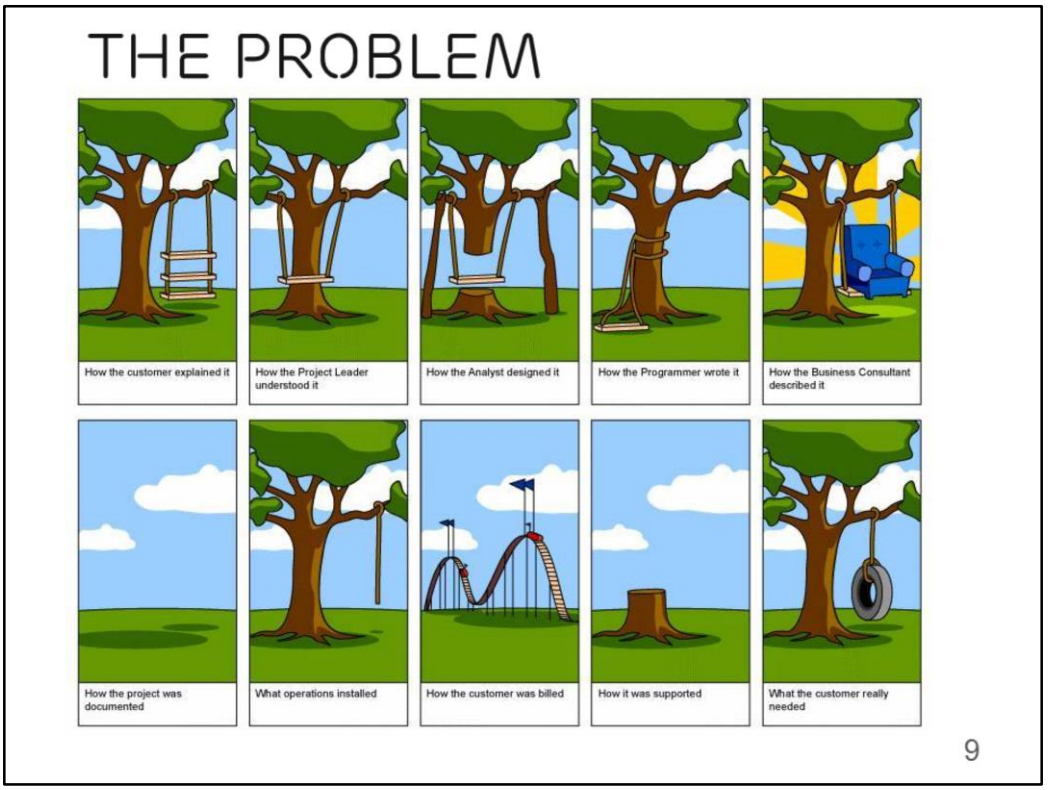
7

http://www.istqb.org/

LEAN & AGILE DEVELOPMENT

Tibor Csöndes, Honorary Associate Professor

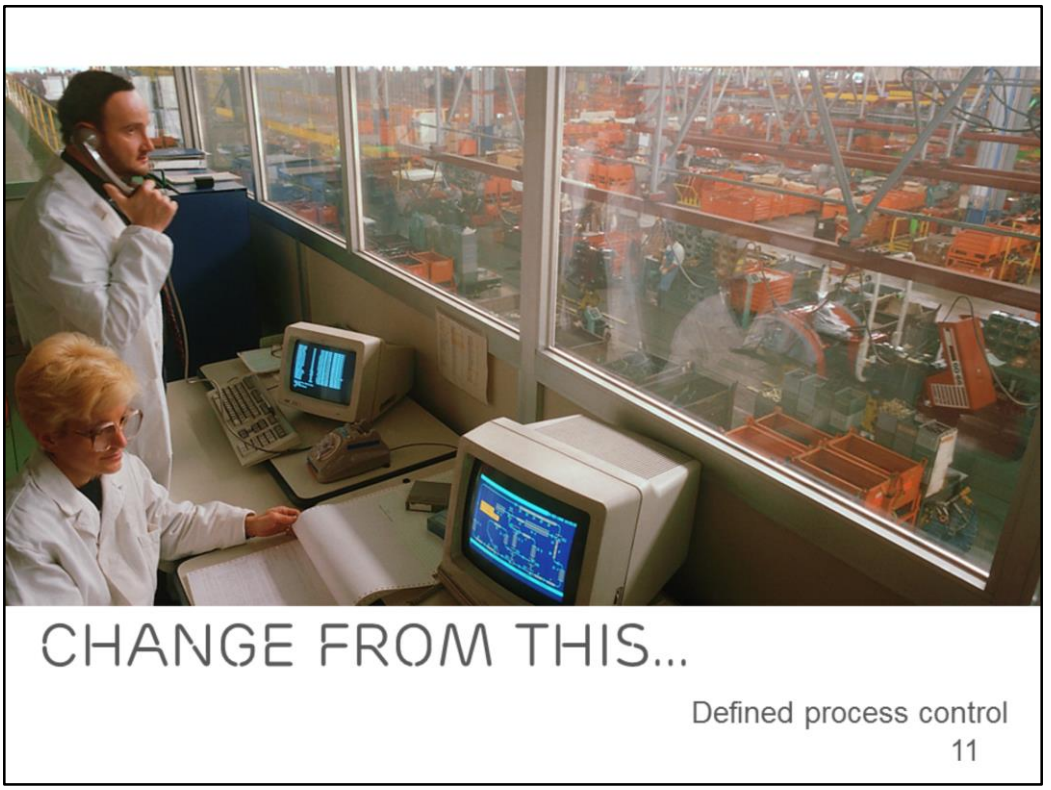
csondes@tmit.bme.hu


THE PROBLEM
How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed
9

INTRODUCTION

- Software development has been changed
- Need to follow **fast changes** requested by the customer
- Most software development is a chaotic activity, often characterized by the phrase "code and fix"
- Requirement engineering getting more and more important!

10



There are two types of process control: “defined” and “empirical”.

A defined process tells you exactly what to do, which artifacts you have to produce, who you’re supposed to talk with and when...

Defined processes work when you know exactly what you are building and you know exactly what inputs are required to build it.

If you have to produce the same thing day in day out, a defined process might work.

The typical example is an assembly line.

(However, please note that even assembly lines tend not to use defined process these days).

Empirical processes have an “inspect and adapt” approach.

You continuously “inspect” your problems, and “adapt” the process by consequence.

Empirical processes work best when you work in an environment that changes, and you need constant feedback.

Agile and Scrum assume that building software is inherently an empirical

process:

we are building a unique product one time only;

we can’t specify the end result exactly;

we don’t know exactly what will be required to build it.

Think of the difference between designing the Honda Civic (an empirical process) and mass producing the Honda Civic (a defined process).

SOFTWARE DEVELOPMENT PROCESS

1970s

- Structured programming since 1969

1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

1990s

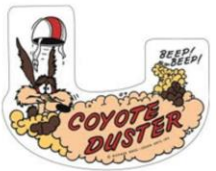
- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Scrum, since 1995
- Extreme programming (XP), since 1999

2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler

13

WHAT SPEED CAN LOOK LIKE



- "New application from concept to production: 2 months"
- "Every IT system fully regression-tested by the end of every iteration"
- "Every IT system built & fully tested at least twice a day"
- "An incident is fully resolved within 24h and the long-term improvement for addressing the root-causes handled within a week."
- "All testing of integration contracts is fully automated."
- "Development & test environments delivered within 30 minutes from initial order."
- "A new car can be produced 20 hours after receiving the customer order.."
- "A new fully automated stock trading strategy can be implemented every week."
- "A stock trade can be cleared in seconds."

14

RAPID SOFTWARE DEVELOPMENT

Rapid development and delivery is now often the most important requirement for software systems

- Businesses operate in a fast-changing requirement and it is practically impossible to produce a set of stable software requirements
- Software has to evolve quickly to reflect changing business needs.

Rapid software development

- Specification, design and implementation are interleaved
- System is developed as a series of versions with stakeholders involved in version evaluation
- User interfaces are often developed using an IDE and graphical toolset.

15

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) MODEL

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "**Software Development Process Models**"

- Waterfall model
- V-model
- Incremental model
- Iterative model
- etc.

Each process model follows a particular life cycle in order to ensure success in process of software development.

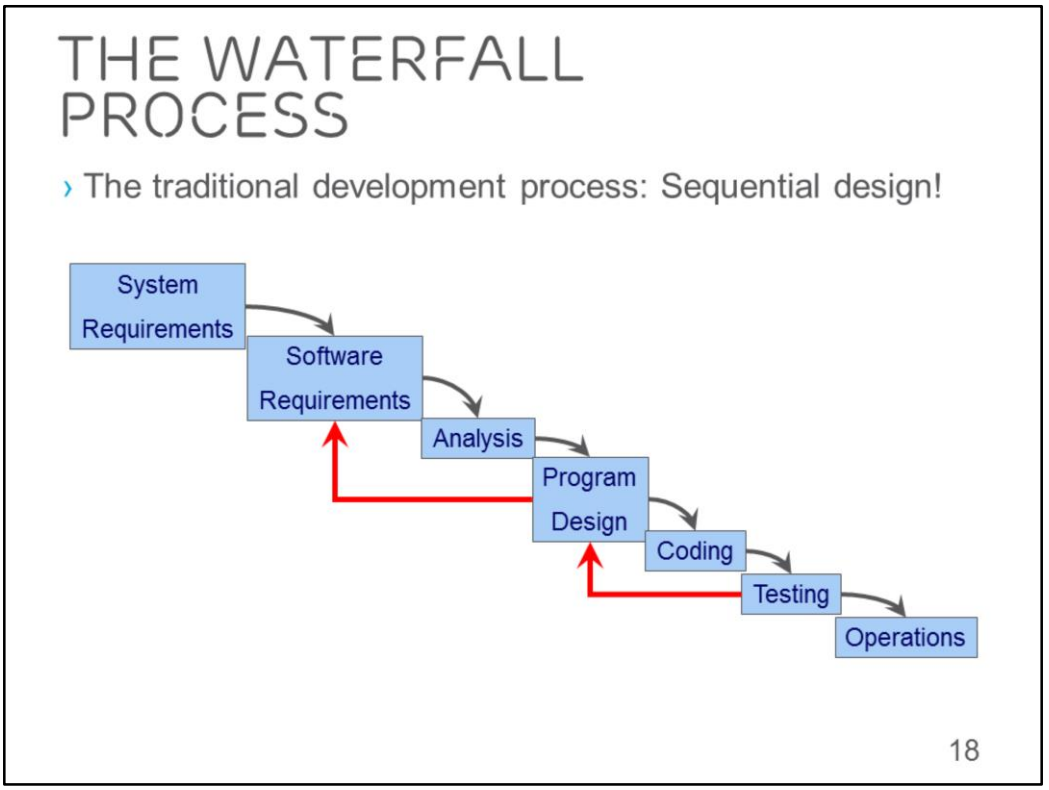
16

SOFTWARE DEVELOPMENT LIFE CYCLE MODEL PHASES

There are following six phases in every Software development life cycle model:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

17



It’s only in testing that your design is tested against reality. At this point you always learn something about your design (usually that it is wrong!), and that tells you something about your requirements (ditto!)

WATERFALL MODEL

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model.

It is very simple to understand and use.

In a waterfall model, each phase must be completed fully before the next phase can begin.

This type of model is basically used for the for the **project** which **is small** and there are no uncertain requirements.

At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project.

In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

19

WATERFALL MODEL

Advantages of waterfall model:

- This model is **simple and easy to understand** and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

20

WHERE DOES WATERFALL WORK WELL?

Organizations where requirements don't change

- Stable requirements leads to stable design…
- Stable design leads to "no surprise" implementation

Extremely high-reliability systems (product or custom projects), where functions are very well understood and no changes in requirements during a project are desired

Who has requirements like this?

- NASA

Embedded products with hardware constraints that cannot be easily changed

Complex projects (product or custom) where parts of design & coding are outsourced, off-shored, or done in multiple sites, AND there are weak mechanisms to synchronize and manage distributed teams

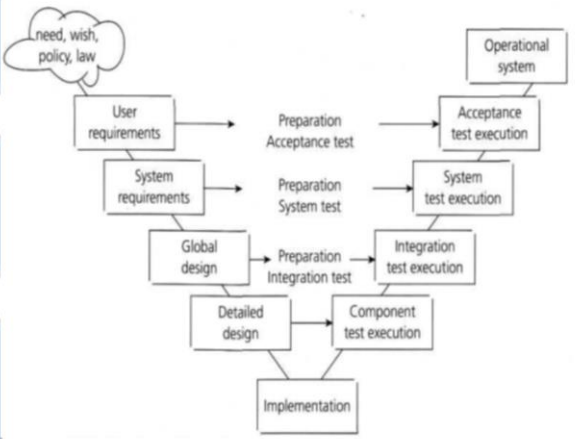
21

V-MODEL
(VERIFICATION AND VALIDATION)

The V-model is an extension of the waterfall model.

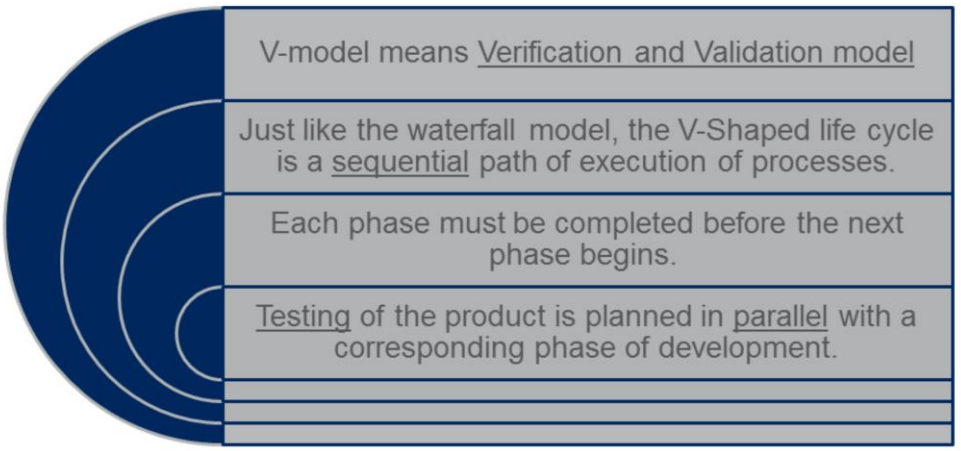
Show the relationships between development phases and test phases

Time and project completeness vs. level of abstraction



22

V-MODEL


V-model means Verification and Validation model
Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes.
Each phase must be completed before the next phase begins.
Testing of the product is planned in parallel with a corresponding phase of development.

23

V-MODEL

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

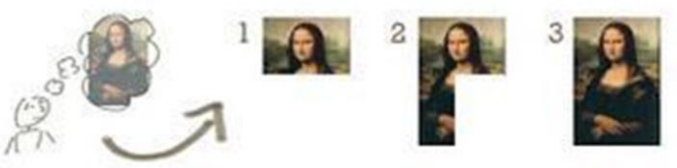
24

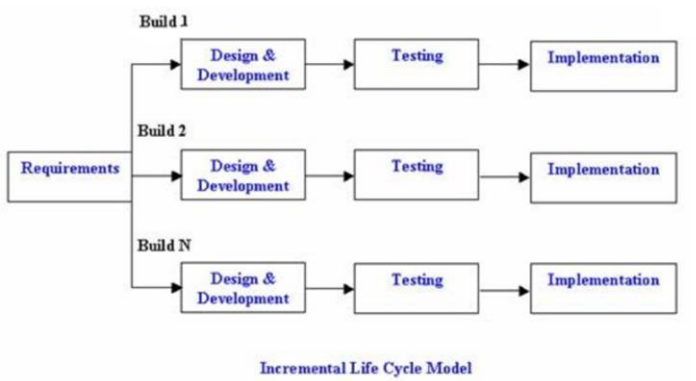
INCREMENTAL MODEL

- The whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a “multi-waterfall” cycle.
- Cycles are divided up into smaller, more easily managed modules.
- Each module passes through the requirements, design, implementation and testing phases.
- A working version of software is produced during the first module, so you have working software early on during the software life cycle.
- Each subsequent release of the module adds function to the previous release.
- The process continues till the complete system is achieved.

25

INCREMENTAL MODEL





Incremental Life Cycle Model

26

INCREMENTAL MODEL

Advantages of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

27

WHEN TO USE THE INCREMENTAL MODEL

- This model can be used when the **requirements** of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

28

AGILE MODEL

Type of Incremental model

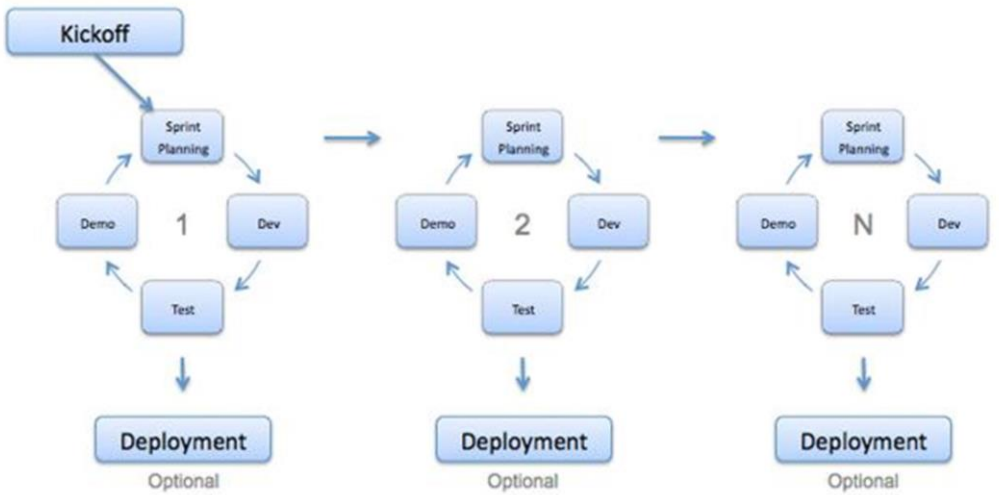
Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality

Each release is thoroughly tested to ensure software quality is maintained

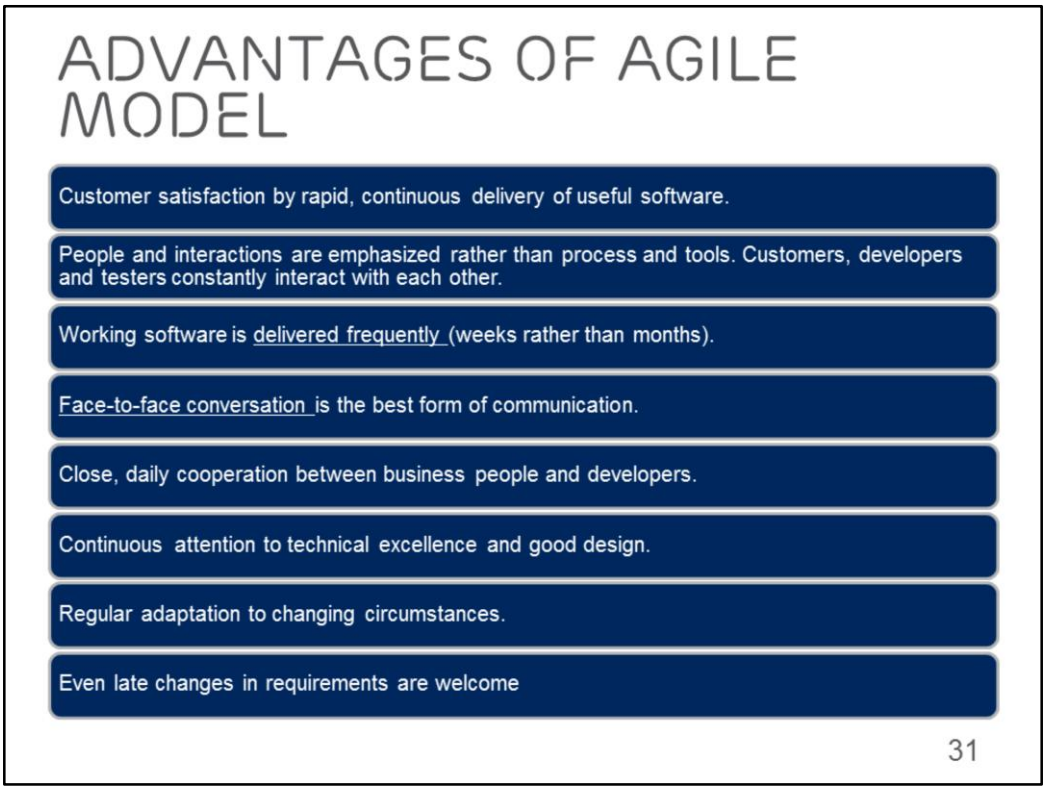
It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model

29

AGILE MODEL



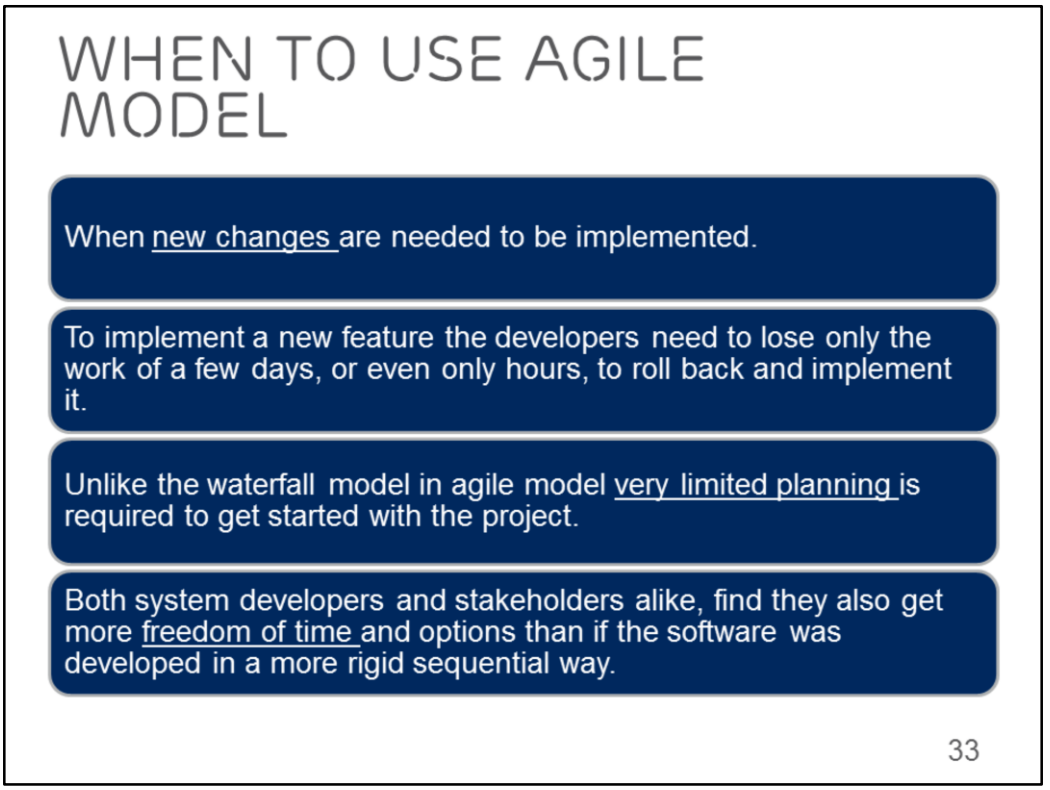
30


ADVANTAGES OF AGILE MODEL
Customer satisfaction by rapid, continuous delivery of useful software.
People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
Working software is delivered frequently (weeks rather than months).
Face-to-face conversation is the best form of communication.
Close, daily cooperation between business people and developers.
Continuous attention to technical excellence and good design.
Regular adaptation to changing circumstances.
Even late changes in requirements are welcome
31

DISADVANTAGES OF AGILE MODEL

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of **decisions** required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

32



- **When new changes are needed to be implemented.** The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
- **To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.**
- **Unlike the waterfall model in agile model very limited planning is required to get started with the project.** Agile assumes that the end users' needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.
- **Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way.** Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.

ITERATIVE LIFE CYCLE MODEL

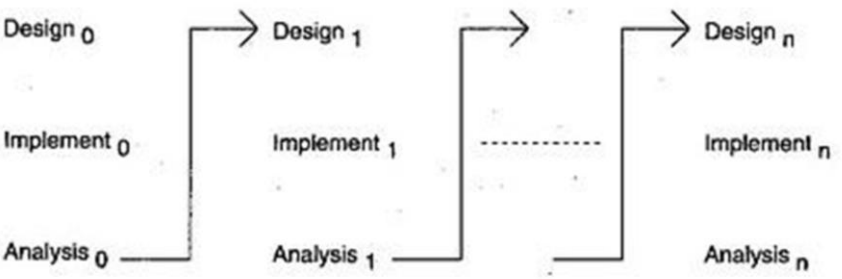
Does not attempt to start with a full specification of requirements

Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements

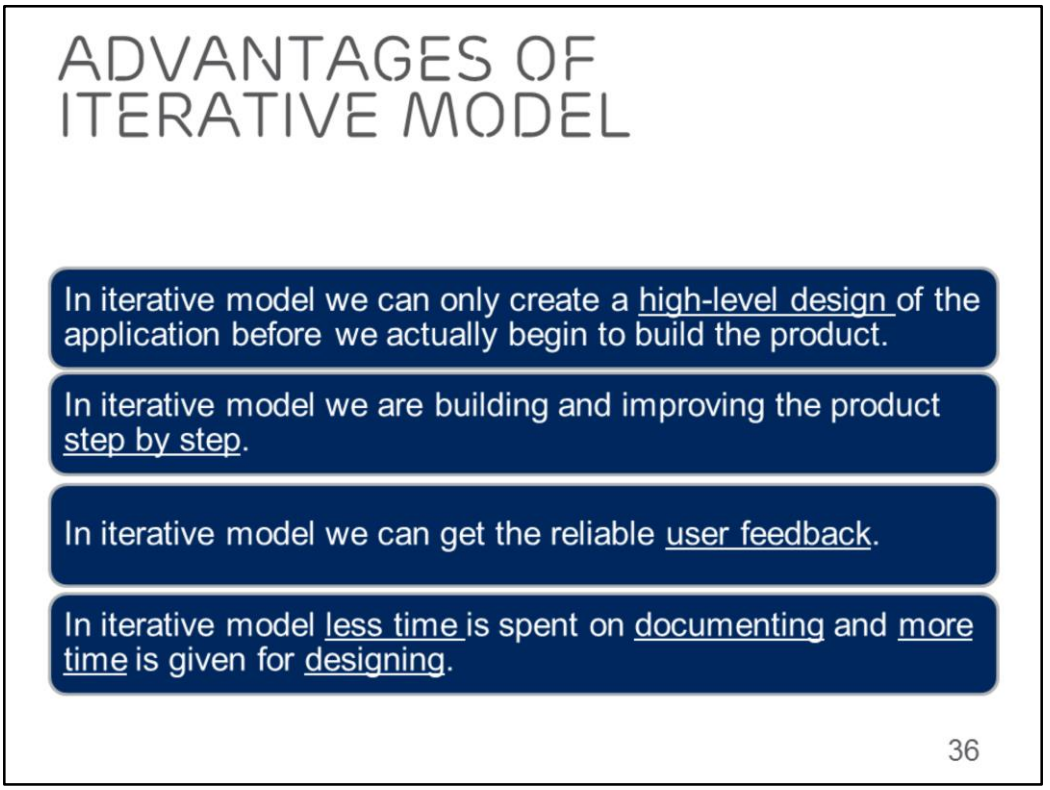
This process is then repeated, producing a new version of the software for each cycle of the model

34

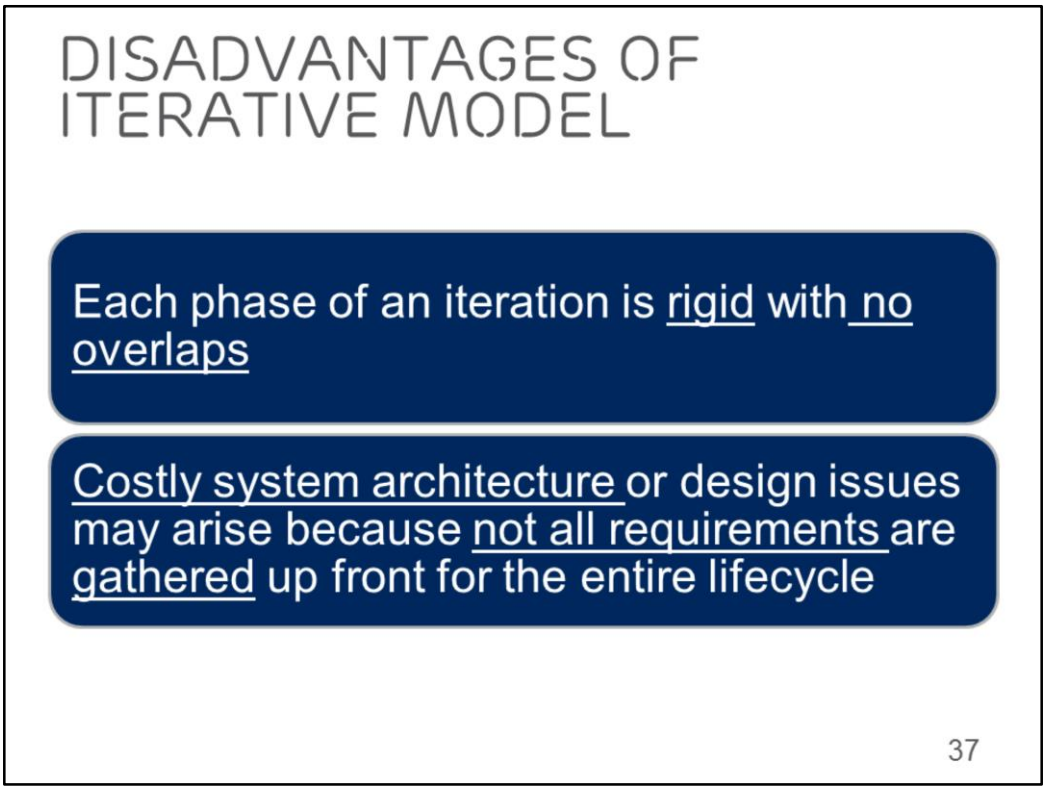
ITERATIVE LIFE CYCLE MODEL

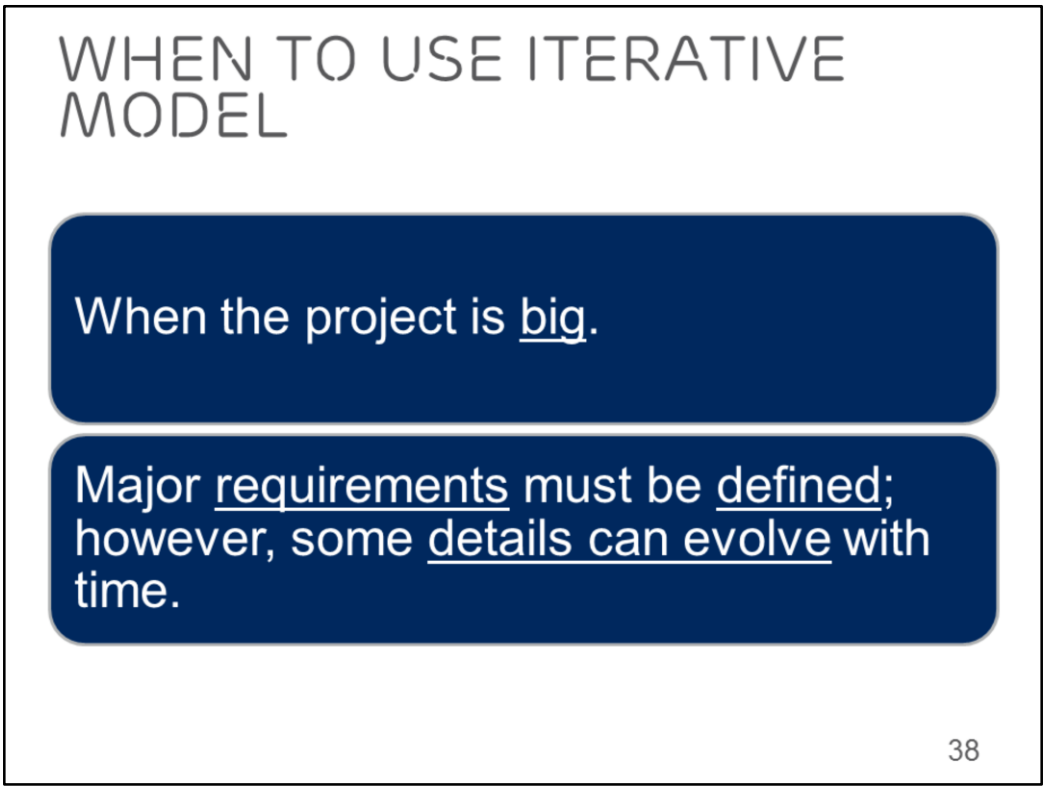


35



- **In iterative model we can only create a high-level design of the application before we actually begin to build the product** and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- **In iterative model we are building and improving the product step by step.** Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- **In iterative model we can get the reliable user feedback.** When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- **In iterative model less time is spent on documenting and more time is given for designing.**


DISADVANTAGES OF ITERATIVE MODEL
Each phase of an iteration is rigid with no overlaps
Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle
37

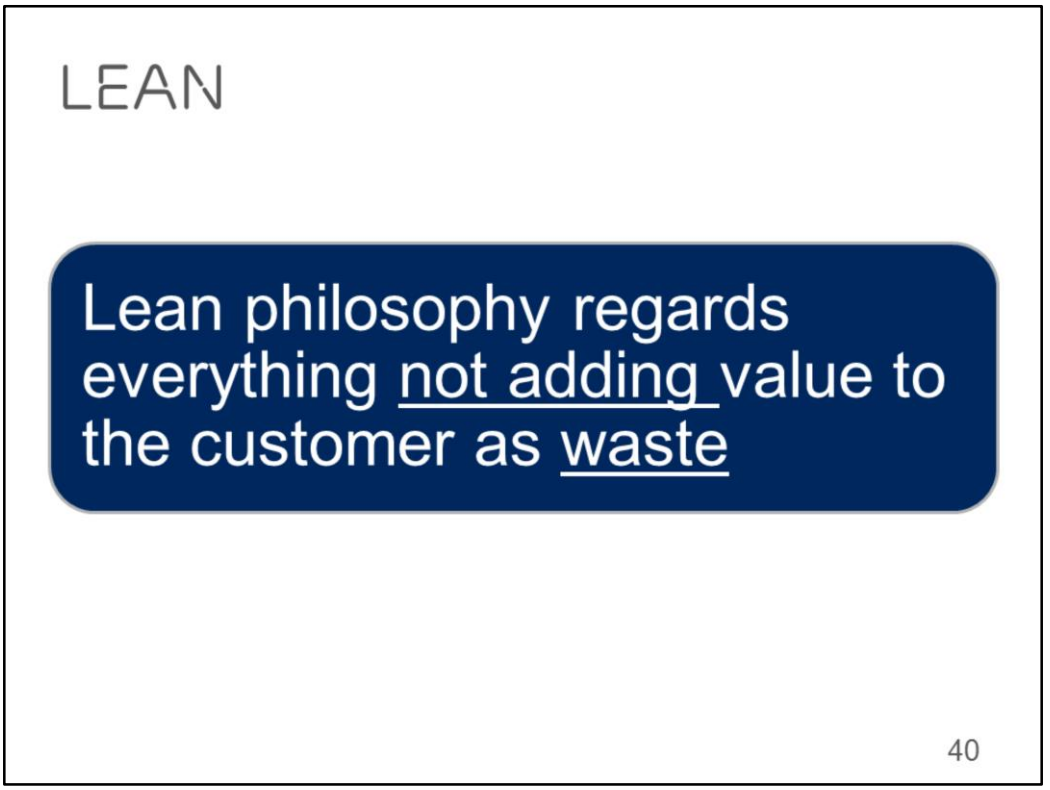

WHEN TO USE ITERATIVE MODEL
When the project is big.
Major requirements must be defined; however, some details can evolve with time.
38

INCREMENTAL VS. ITERATIVE

Incremental

Iterative

39


LEAN
Lean philosophy regards everything not adding value to the customer as waste
40

LEAN HISTORY

- 1999 — Enterprise-level Lean multi-company value stream optimizations
- 1997 — Enterprise-level Lean roll-out — *Lean Thinking published*
- 1990s — "Lean" application to non-automotive
- 1990 — MIT International Motor Vehicle Research program
- 1985 — US auto industry crisis — *Dr. Jim Womack, Research Director*
- Mid 1980s — Japanese auto makers emerge as quality leaders
- Early 1980s — Second oil shock
- 1978 — Toyota Production System (TPS) developed in Japan by Taiichi Ohno
- 1970's
- 1950's

Lean developed in a Manufacturing background

- 1945 — Mass production thrives
- 1927 — Ford opens River Rouge complex

- 1913 — First automobile assembly line at Highland Park

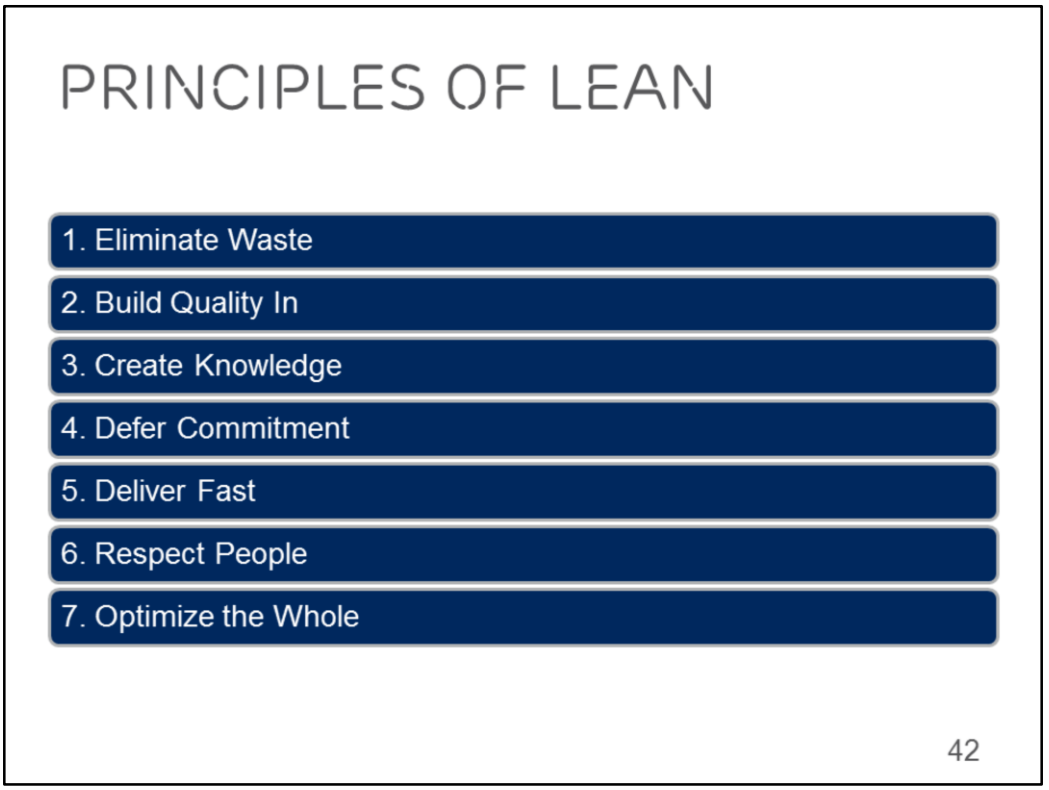
- 1850 — Eli Whitney builds muskets with interchangeable parts

- 1750 — Industrial Revolution begins mass production

- Pre-1700's — Cottage industries, crafts, and guilds

Dates back to Mass Production days

41


PRINCIPLES OF LEAN
1. Eliminate Waste
2. Build Quality In
3. Create Knowledge
4. Defer Commitment
5. Deliver Fast
6. Respect People
7. Optimize the Whole
42

We want to maximize time spent on activities that add value to the product, and eliminate waste activities.

What is waste? It is any activity or product that does not provide value to customers – anything a customer would not pay for.

There are 7 wastes in software:

- inventory (features not yet deployed, or incomplete features)

-extra processes (such as unnecessary documents)

-extra or unused features

-task switching

-Waiting

-Motion (for example walking across the building to ask someone a question)

-defects

What type of waste does this image represent?

The 7 types of software waste can be compared the types of waste in manufacturing:

Rework, Overproduction, Conveyance, Waiting, Inventory, Motion, Overprocessing

Latent Skill, Danger, Poor Information, Material, Breakdown

Wikipedia

Lean philosophy regards everything not adding value to the customer as waste (*muda*). Such waste may include:

unnecessary code and functionality

delay in the software development process

unclear requirements

avoidable process repetition (often caused by insufficient testing)

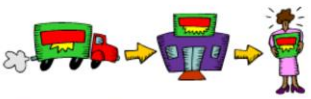
bureaucracy

slow internal communication

In order to eliminate waste, one should be able to recognize it. If some activity could be bypassed or the result could be achieved without it, it is waste. Partially done coding eventually abandoned during the development process is waste. Extra processes and features not often used by customers are waste. Waiting for other activities, teams, processes is waste. Defects and lower quality are waste. Managerial overhead not producing real value is waste.

A value stream mapping technique is used to identify waste. The second step is to point out sources of waste and to eliminate them. Waste-removal should take place iteratively until even seemingly essential processes and procedures are liquidated.

7 WASTES IN SW DEVELOPMENT



Extra Processes

Extra features

Defects

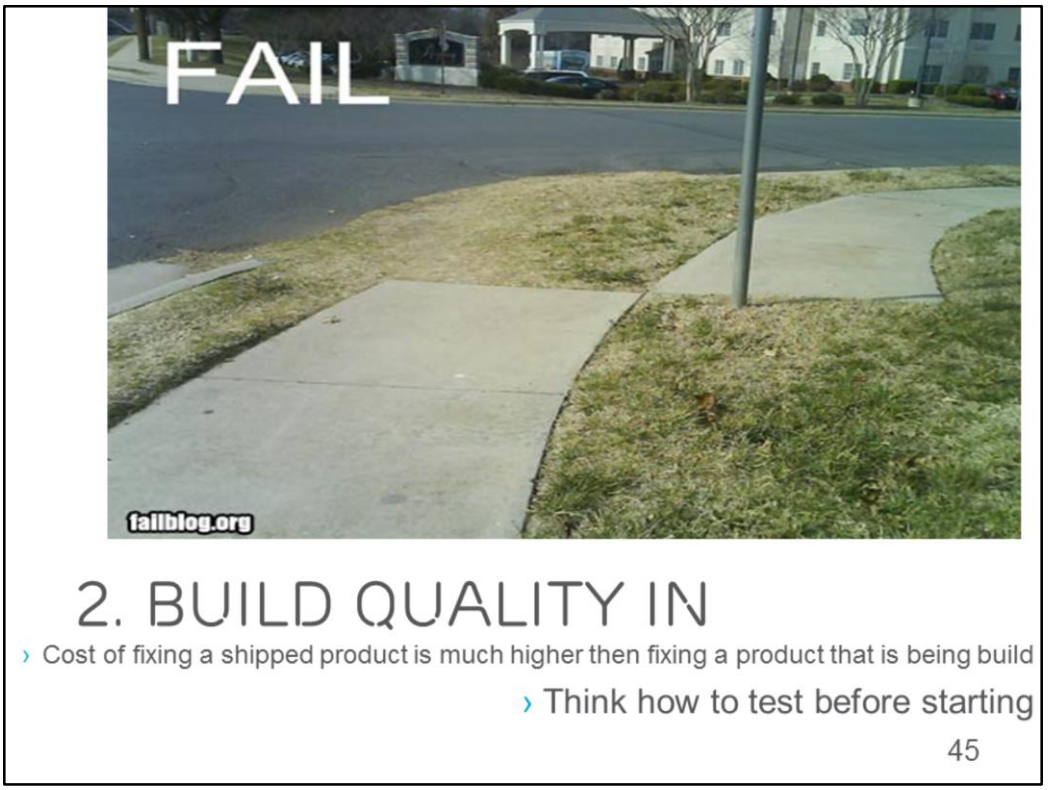
Partially done work

Task switching

Motion and handover

Waiting & Delays

44



The people who built this path didn't build quality in. The cost of fixing a shipped product is much higher then fixing a product that is being build.

Piling up code on top of a shaky foundation will result in a very expensive re-factoring effort.

Continuous refactoring, continuous testing, and continuous integration are agile ways to build quality in.

Wikipedia

Build integrity in

The customer needs to have an overall experience of the System – this is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, price and how well it solves problems.

Conceptual integrity means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness. This could be achieved by understanding the problem domain and solving it at the same time, not sequentially. The needed information is received in small batch pieces – not in one vast chunk with preferable face-to-face communication and not any written documentation. The information flow should be constant in both directions – from customer to developers and back, thus avoiding the large stressful amount of information

after long development in isolation.

One of the healthy ways towards integral architecture is refactoring. As more features are added to the original code base, the harder it becomes to add further improvements. Refactoring is about keeping simplicity, clarity, minimum amount of features in the code. Repetitions in the code are signs for bad code designs and should be avoided. The complete and automated building process should be accompanied by a complete and automated suite of developer and customer tests, having the same versioning, synchronization and semantics as the current state of the System. At the end the integrity should be verified with thorough testing, thus ensuring the System does what the customer expects it to. Automated tests are also considered part of the production process, and therefore if they do not add value they should be considered waste. Automated testing should not be a goal, but rather a means to an end, specifically the reduction of defects.

JBTDT: Just Build The Damn Thing - learn from actually building it.

Engage as a Team: let the Specialist dissect the problem, let the rest of the Team build the solution.

Share knowledge gained with the whole Team.

Share knowledge across multiple Teams.

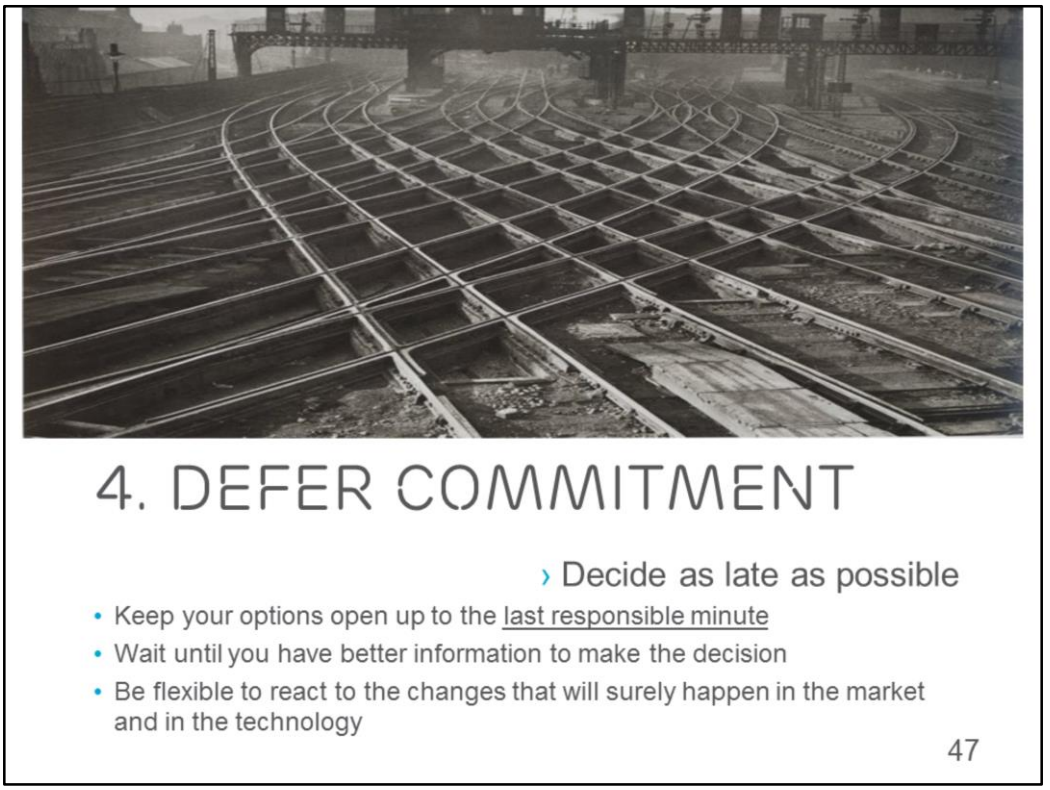
Wikipedia

Amplify learning

Software development is a continuous learning process with the additional challenge of development teams and end product sizes. The best approach for improving a software development environment is to amplify learning. The accumulation of defects should be prevented by running tests as soon as the code is written. Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building. The process of user requirements gathering could be simplified by presenting screens to the end-users and getting their input.

The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with customers helps when determining the current phase of development and adjusting efforts for future improvements. During those short sessions both customer representatives and the development team

learn more about the domain problem and figure out possible solutions for further development. Thus the customers better understand their needs, based on the existing result of development efforts, and the developers learn how to better satisfy those needs. Another idea in the communication and learning process with a customer is set-based development – this concentrates on communicating the constraints of the future solution and not the possible solutions, thus promoting the birth of the solution via dialogue with the customer.



Keep your options open up to the last responsible minute.

Wait until you have better information to make the decision.

Be flexible to react to the changes that will surely happen in the market and in the technology.

[Note to presenter: very observant people may notice that this image shows a crossing, and not a switch where trains can change tracks. Change the image if you like.]

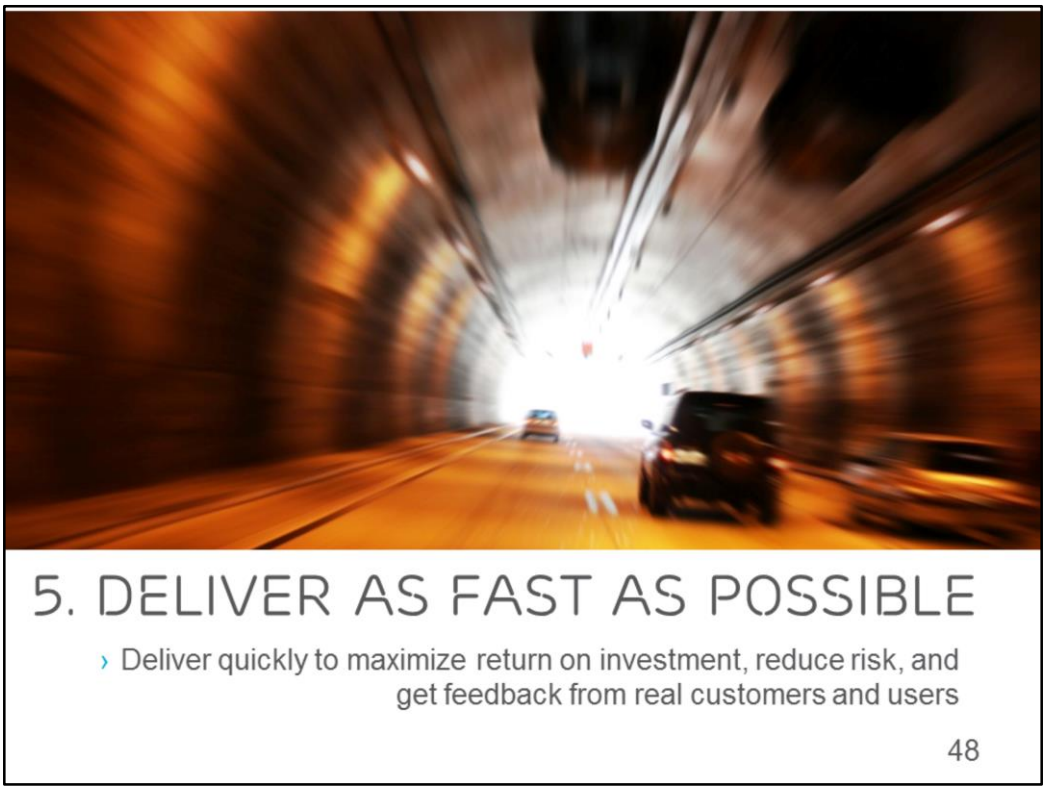
Wikipedia

Decide as late as possible

As software development is always associated with some uncertainty, better results should be achieved with an options-based approach, delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle – the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

An agile software development approach can move the building of options earlier for customers, thus delaying certain crucial decisions until customers

have realized their needs better. This also allows later adaptation to changes and the prevention of costly earlier technology-bounded decisions. This does not mean that no planning should be involved – on the contrary, planning activities should be concentrated on the different options and adapting to the current situation, as well as clarifying confusing situations by establishing patterns for rapid action. Evaluating different options is effective as soon as it is realized that they are not free, but provide the needed flexibility for late decision making.



We deliver quickly to maximize return on investment, reduce risk, and get feedback from real customers and users.

Markets and technology change quickly. Long delivery cycles increase risk that you won't deliver a product that meets customers' needs.

Wikipedia

Deliver as fast as possible

In the era of rapid technology evolution, it is not the biggest that survives, but the fastest. The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team. With speed, decisions can be delayed. Speed assures the fulfilling of the customer's present needs and not what they required yesterday. This gives them the opportunity to delay making up their minds about what they really require until they gain better knowledge. Customers value rapid delivery of a quality product.

The just-in-time production ideology could be applied to software development, recognizing its specific requirements and environment. This is achieved by presenting the needed result and letting the team organize itself and divide the tasks for accomplishing the needed result for a specific iteration. At the beginning, the customer provides the needed input. This could be simply presented in small cards or stories – the developers estimate the

time needed for the implementation of each card. Thus the work organization changes into self-pulling system – each morning during a stand-up meeting, each member of the team reviews what has been done yesterday, what is to be done today and tomorrow, and prompts for any inputs needed from colleagues or the customer. This requires transparency of the process, which is also beneficial for team communication. Another key idea in Toyota's Product Development System is set-based design. If a new brake system is needed for a car, for example, three teams may design solutions to the same problem. Each team learns about the problem space and designs a potential solution. As a solution is deemed unreasonable, it is cut. At the end of a period, the surviving designs are compared and one is chosen, perhaps with some modifications based on learning from the others - a great example of deferring commitment until the last possible moment. Software decisions could also benefit from this practice to minimize the risk brought on by big up-front design.

Leverage ALL of your people's talents and intelligence. Give them a goal and let them find the best way to accomplish it. When people have ownership over their work they are more motivated.

The army is a great example of "self-organized units".

Teams are given a "Commander's intent" and can organize themselves as they see fit.

Wikipedia

Empower the team

There has been a traditional belief in most businesses about the decision-making in the organization – the managers tell the workers how to do their own job. In a "Work-Out technique", the roles are turned – the managers are taught how to listen to the developers, so they can explain better what actions might be taken, as well as provide suggestions for improvements. The lean approach favors the aphorism "find good people and let them do their own job," encouraging progress, catching errors, and removing impediments, but not micro-managing.

Another mistaken belief has been the consideration of people as resources. People might be resources from the point of view of a statistical data sheet, but in software development, as well as any organizational business, people do need something more than just the list of tasks and the assurance that

they will not be disturbed during the completion of the tasks. People need motivation and a higher purpose to work for – purpose within the reachable reality, with the assurance that the team might choose its own commitments. The developers should be given access to the customer; the team leader should provide support and help in difficult situations, as well as ensure that skepticism does not ruin the team's spirit.

Apply Systems thinking.

Find the weakest link or the biggest problem in your whole system and fix that first.

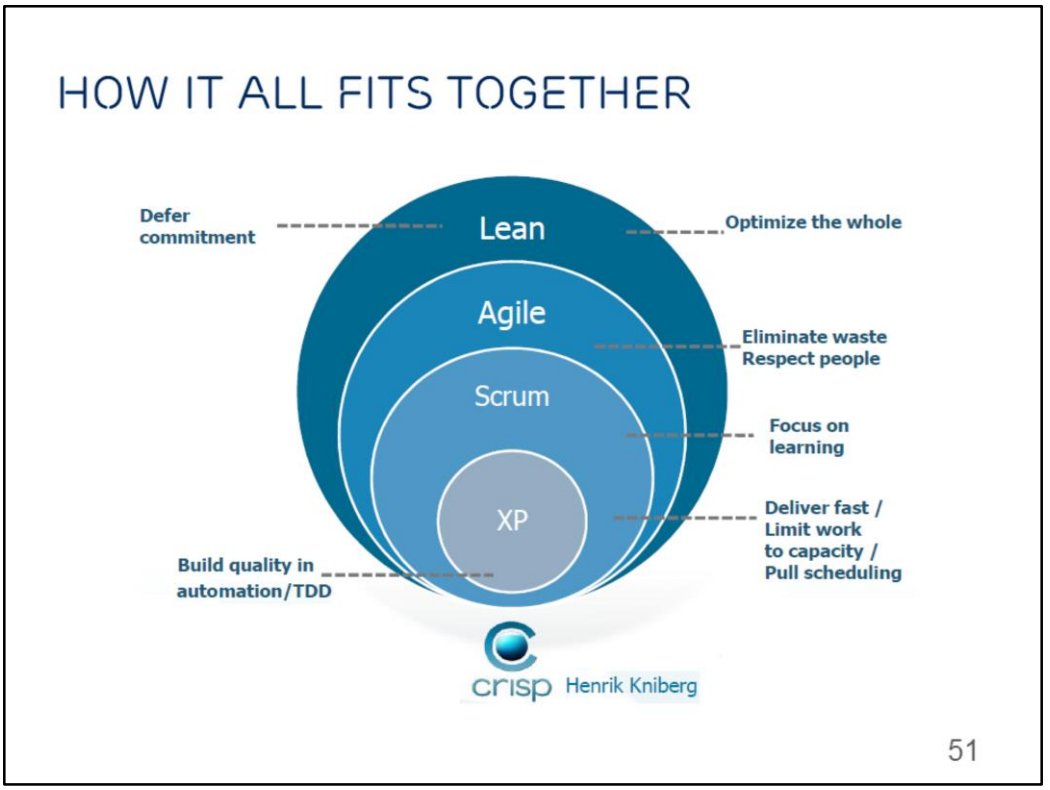
Sub-optimization is to trying to optimize some small part of the system, which may not help the whole system. Suppose you add more programmers to a team so you can create more code, but you don't have enough testers to test and deploy all the extra code; the system still cannot deliver more value.

Wikipedia

See the whole

Software systems nowadays are not simply the sum of their parts, but also the product of their interactions. Defects in software tend to accumulate during the development process – by decomposing the big tasks into smaller tasks, and by standardizing different stages of development, the root causes of defects should be found and eliminated. The larger the system, the more organizations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors, in order to produce a system with smoothly interacting components. During a longer period of development, a stronger subcontractor network is far more beneficial than short-term profit optimizing, which does not enable win-win relationships.

Lean thinking has to be understood well by all members of a project, before implementing in a concrete, real-life situation. "Think big, act small, fail fast; learn rapidly" – these slogans summarize the importance of understanding the field and the suitability of implementing lean principles along the whole software development process. Only when all of the lean principles are implemented together, combined with strong "common sense" with respect to the working environment, is there a basis for success in software development.



Speakers notes:

It begins with Lean as a concept, optimizing the whole vaule flow

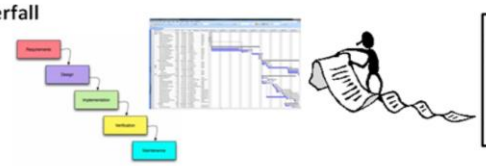
With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)

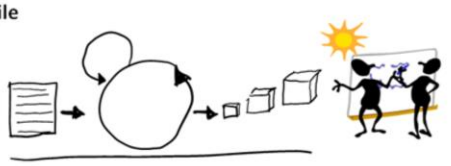
WATERFALL, AGILE, LEAN

Waterfall



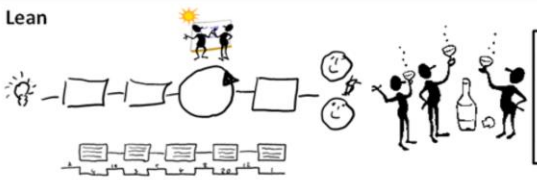
Schedule large work orders and align **people by workflow**

Agile



Schedule small work orders and align **people by schedule**

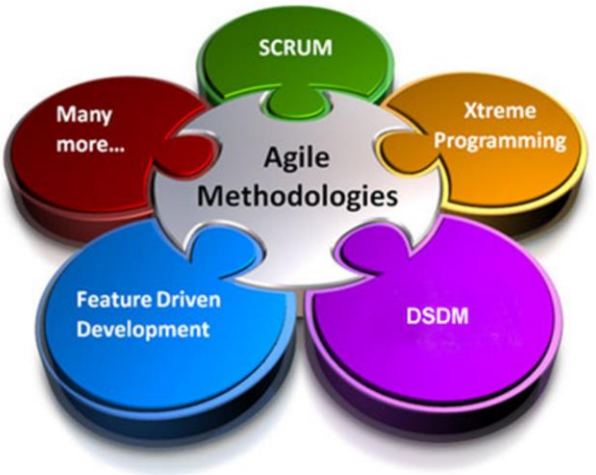
Lean



Schedule small work orders and align **people by workflow**

52

AGILE

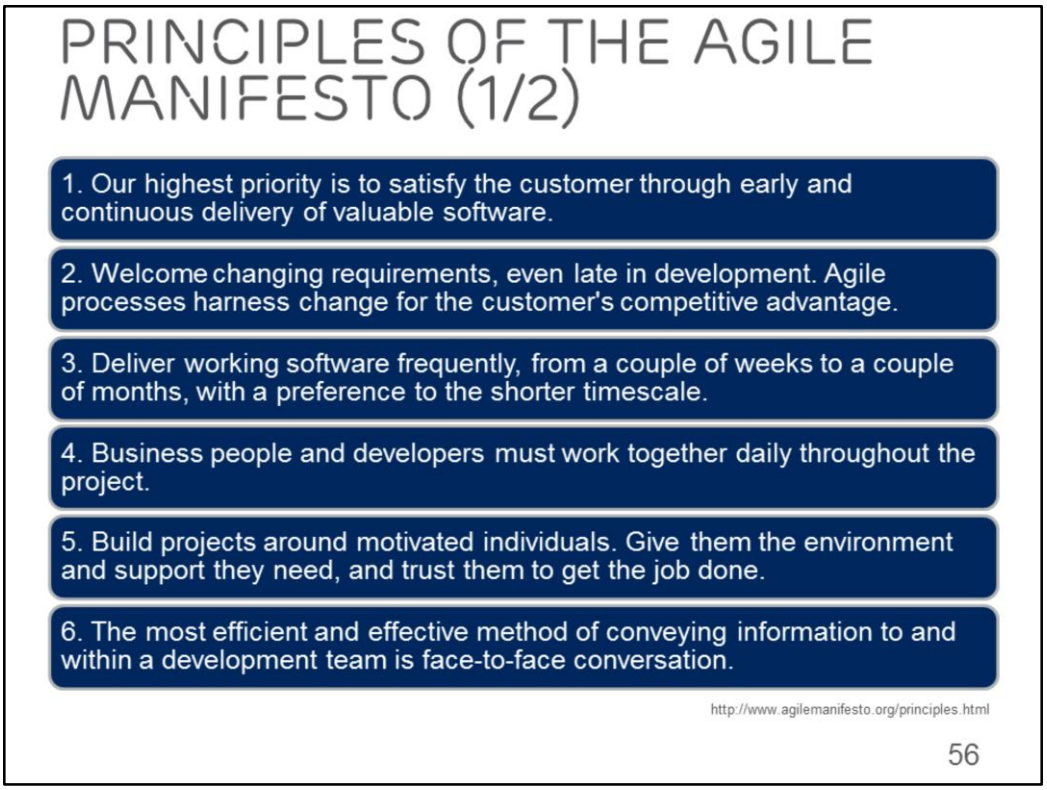

SCRUM
Many more...
Xtreme Programming
Agile Methodologies
Feature Driven Development
DSDM

53

HTTP://AGILEMANIFESTO.ORG/

HTTP://AGILEMANIFESTO.ORG/ISO/HU/MANIFESTO.HTML

55



The manifesto also includes twelve principles. Here they are.

12 principles: they are each self-explanatory.

PRINCIPLES OF THE AGILE MANIFESTO (2/2)

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

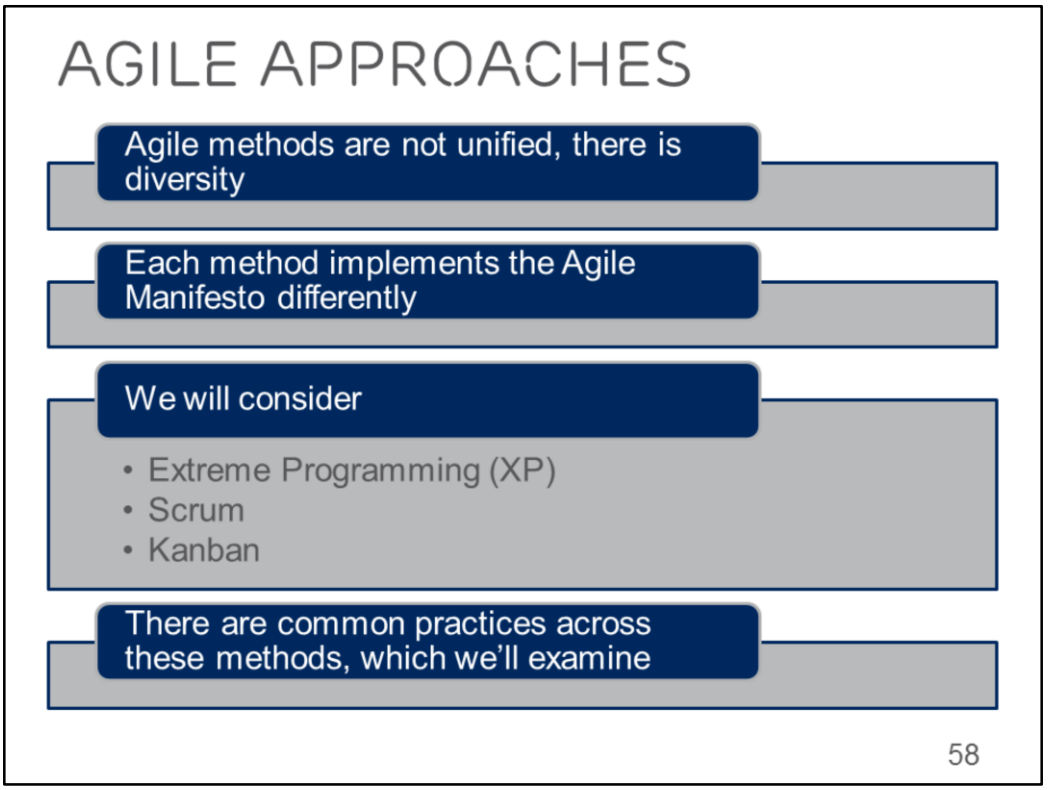
10. Simplicity--the art of maximizing the amount of work not done--is essential. (YAGNI – You Aren't Gonna Need It.)

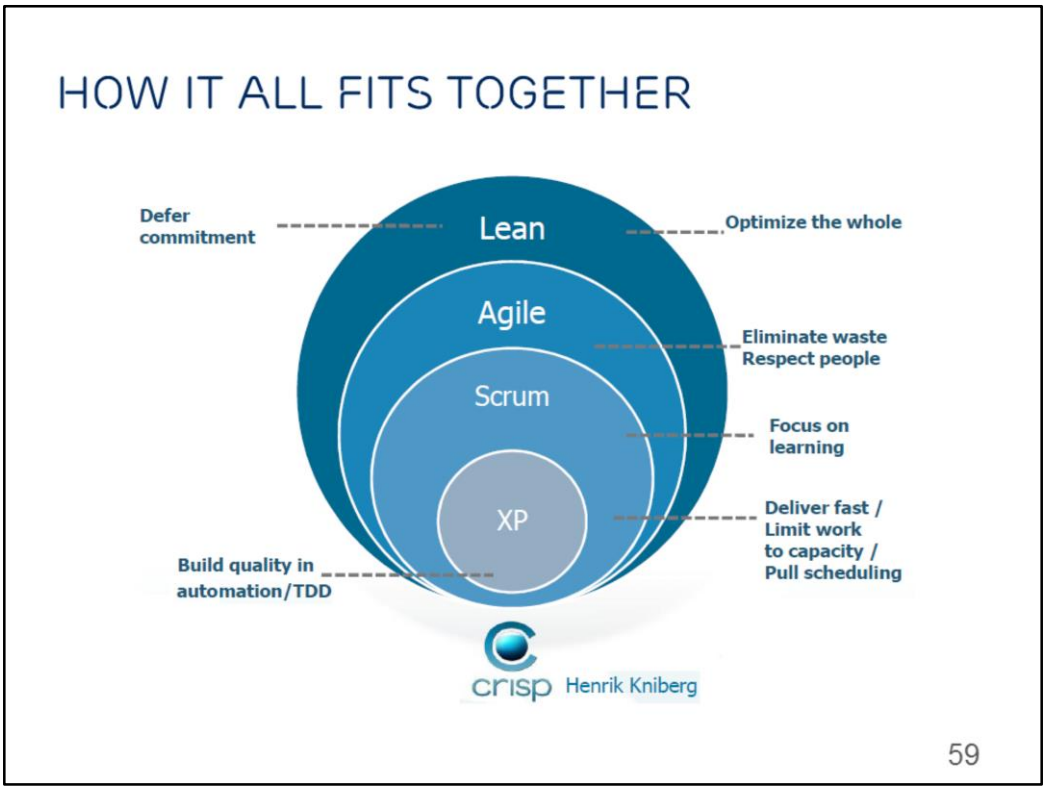
11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://www.agilemanifesto.org/principles.html

57


AGILE APPROACHES
Agile methods are not unified, there is diversity
Each method implements the Agile Manifesto differently
We will consider
• Extreme Programming (XP)
• Scrum
• Kanban
There are common practices across these methods, which we'll examine
58



Speakers notes:

It begins with Lean as a concept, optimizing the whole vaule flow

With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)

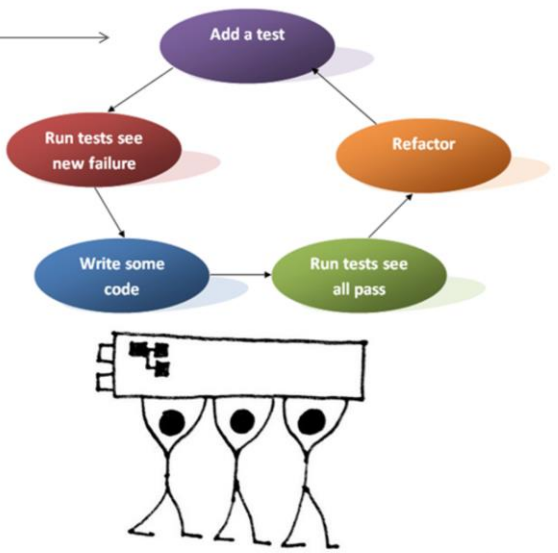
EXTREME PROGRAMMING (XP)

- Formulated in 1999 by Kent Beck, Ward Cunningham and Ron Jeffries
- Agile software development methodology (others: Scrum, DSDM, Kanban)
- Developed in reaction to high ceremony methodologies

60

EXAMPLE OF PRINCIPLES FROM XP (EXTREME PROGRAMMING)

- Test Driven Development
- Continuous Integration
- Collective Code Ownership



62

Speakers notes:

Collective code ownership doesn't mean that everyone is supposed to do everything. It means that we try learn more from each other to become less vulnerable so e g Charles can keep on working with a design task even if Edith is on sick leave on a Monday.

XP: WHY?

Previously:

- Get all the requirements before starting design
- Freeze the requirements before starting development
- Resist changes: they will lengthen schedule
- Build a change control process to ensure that proposed changes are looked at carefully and no change is made without intense scrutiny
- Deliver a product that is obsolete on release

63

XP: EMBRACE CHANGE

Recognize that:

- All requirements will not be known at the beginning
- Requirements will change

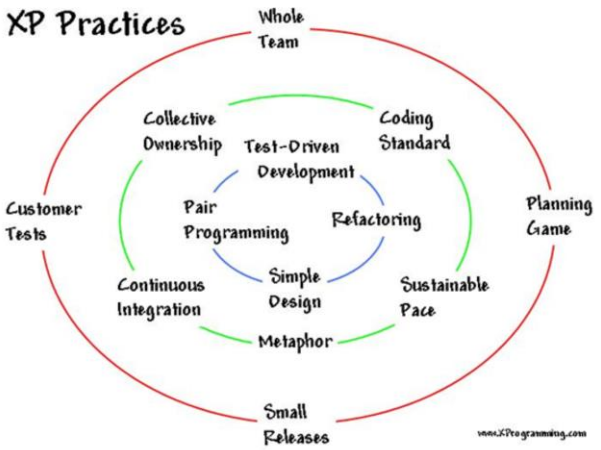
Use tools to accommodate change as a natural process

Do the simplest thing that could possibly work and refactor mercilessly

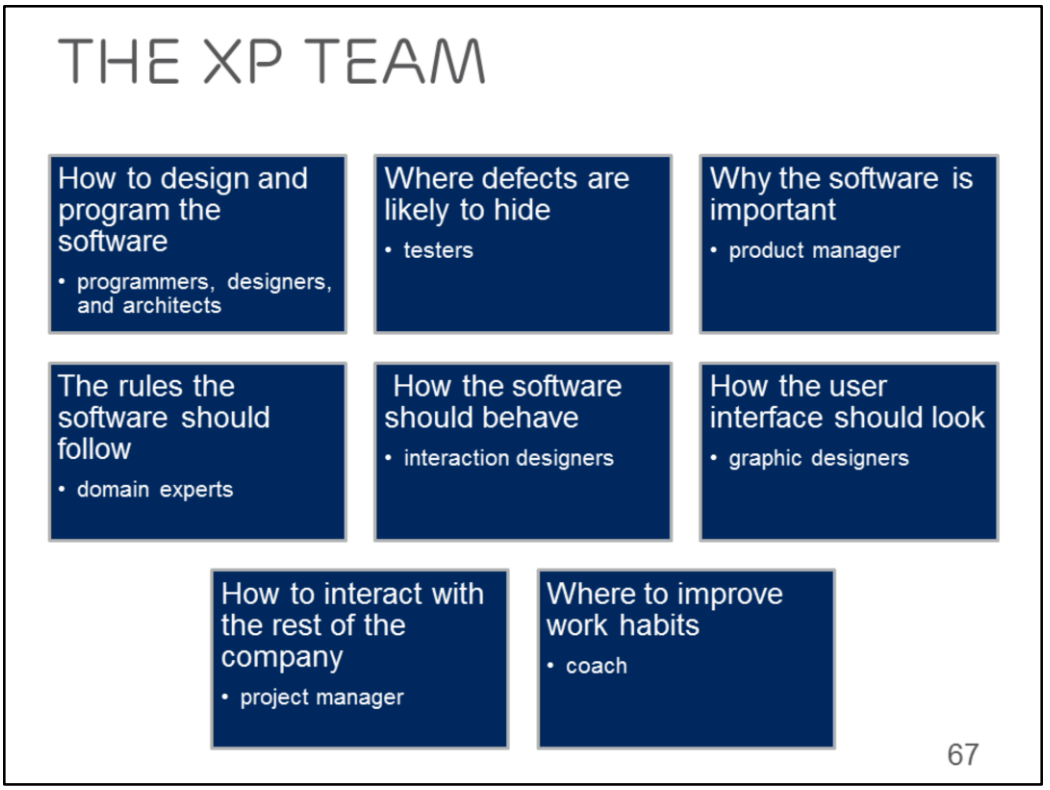
Emphasize values and principles rather than process

64

XP PRACTICES



(Source: http://www.xprogramming.com/xpmag/whatisxp.htm) 65


THE XP TEAM
How to design and program the software
• programmers, designers, and architects
Where defects are likely to hide
• testers
Why the software is important
• product manager
The rules the software should follow
• domain experts
How the software should behave
• interaction designers
How the user interface should look
• graphic designers
How to interact with the rest of the company
• project manager
Where to improve work habits
• coach
67

XP PRACTICES: WHOLE TEAM

All contributors to an XP project are one team

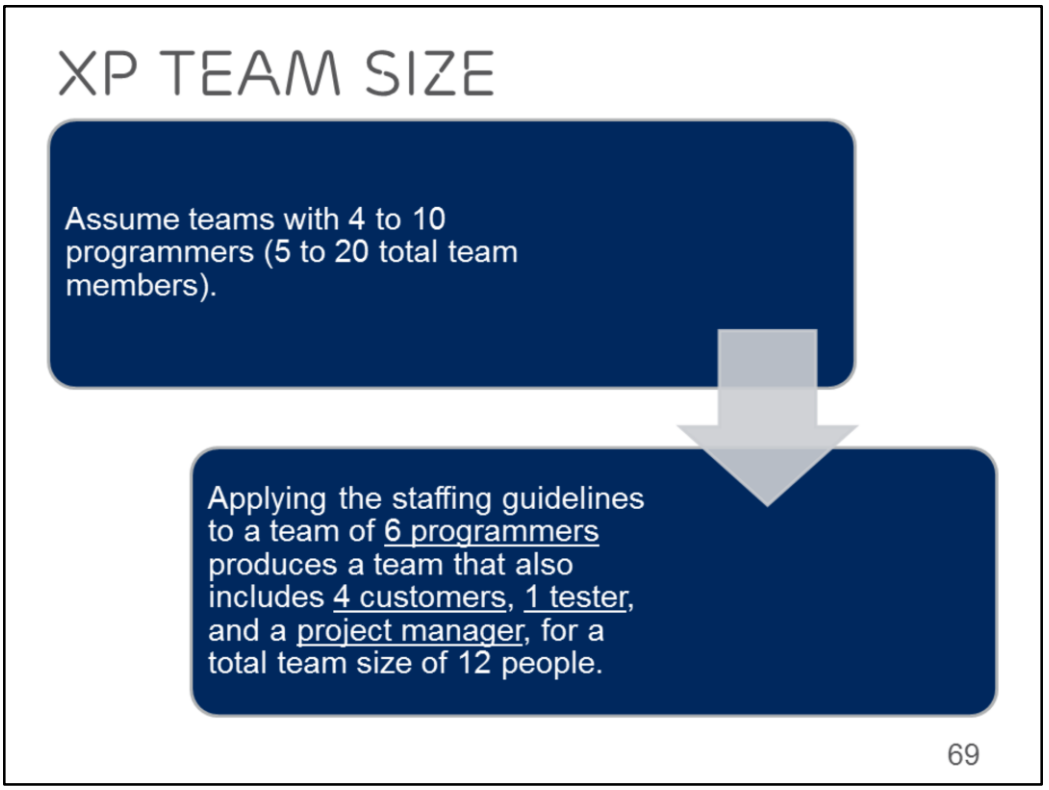
Must include a business representative: the 'Customer'

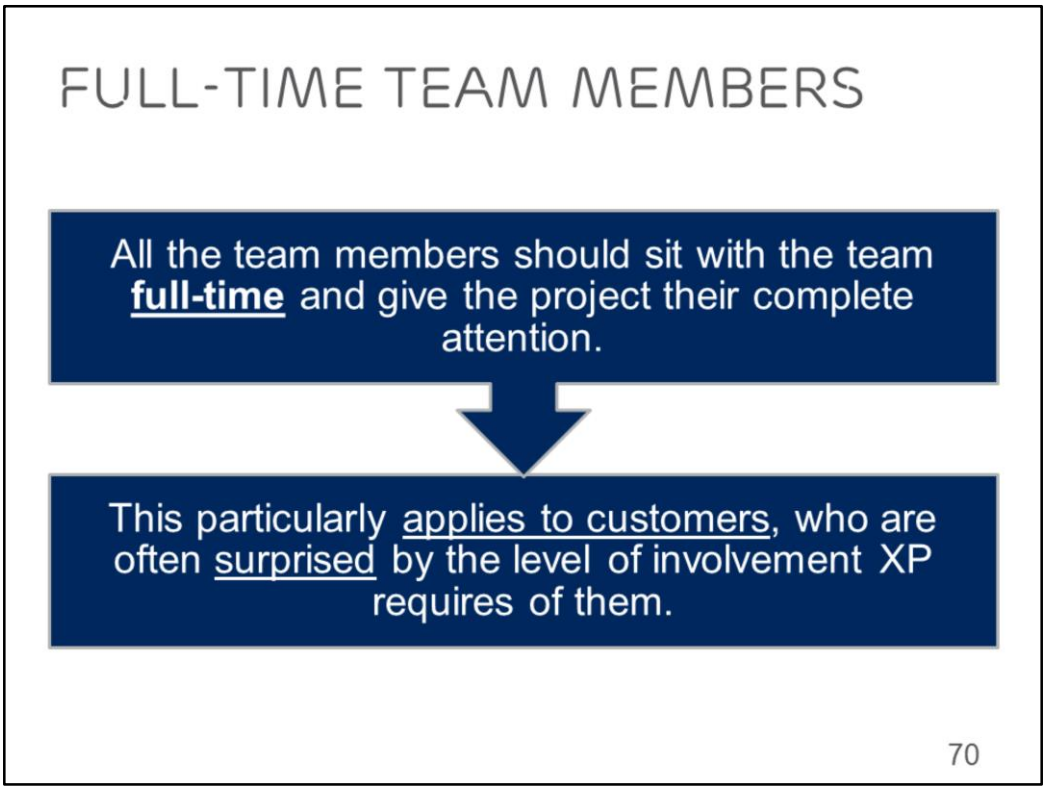
- Provides requirements
- Sets priorities
- Steers project

Team members are programmers, testers, analysts, coach, manager

Best XP teams have no specialists

68


XP TEAM SIZE
Assume teams with 4 to 10 programmers (5 to 20 total team members).
Applying the staffing guidelines to a team of 6 programmers produces a team that also includes 4 customers, 1 tester, and a project manager, for a total team size of 12 people.
69



Some organizations like to assign people to multiple projects simultaneously. This *fractional assignment* is particularly common in *matrix-managed organization*s. (If team members have two managers, one for their project and one for their function, you are probably in a matrixed organization.)

XP PRACTICES: PLANNING GAME

Two key questions in software development:

- Predict what will be accomplished by the due date
- Determine what to do next

Need is to steer the project

Exact prediction (which is difficult) is not necessary

71

XP PRACTICES: PLANNING GAME

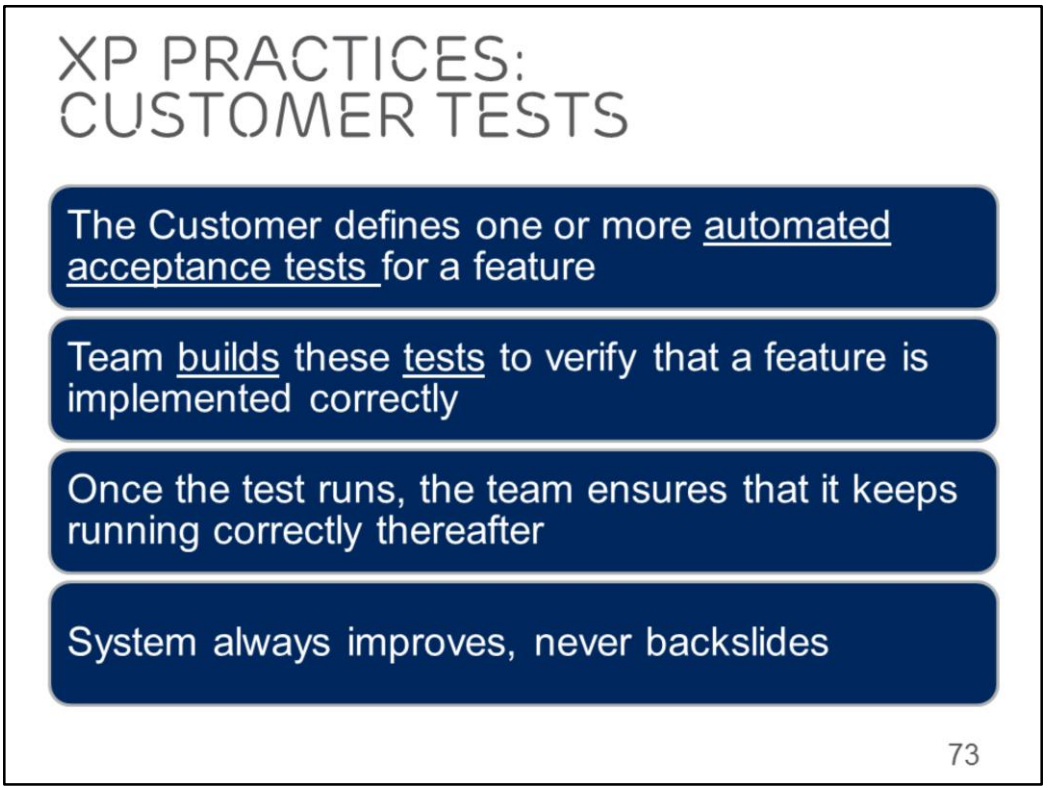
XP Release Planning

- Customer presents required features
- Programmers estimate difficulty
- Imprecise but revised regularly

XP Iteration Planning

- Two week iterations
- Customer presents features required
- Programmers break features down into tasks
- Team members sign up for tasks
- Running software at end of each iteration

72


XP PRACTICES:
CUSTOMER TESTS
The Customer defines one or more automated acceptance tests for a feature
Team builds these tests to verify that a feature is implemented correctly
Once the test runs, the team ensures that it keeps running correctly thereafter
System always improves, never backslides
73

XP PRACTICES: SMALL RELEASES

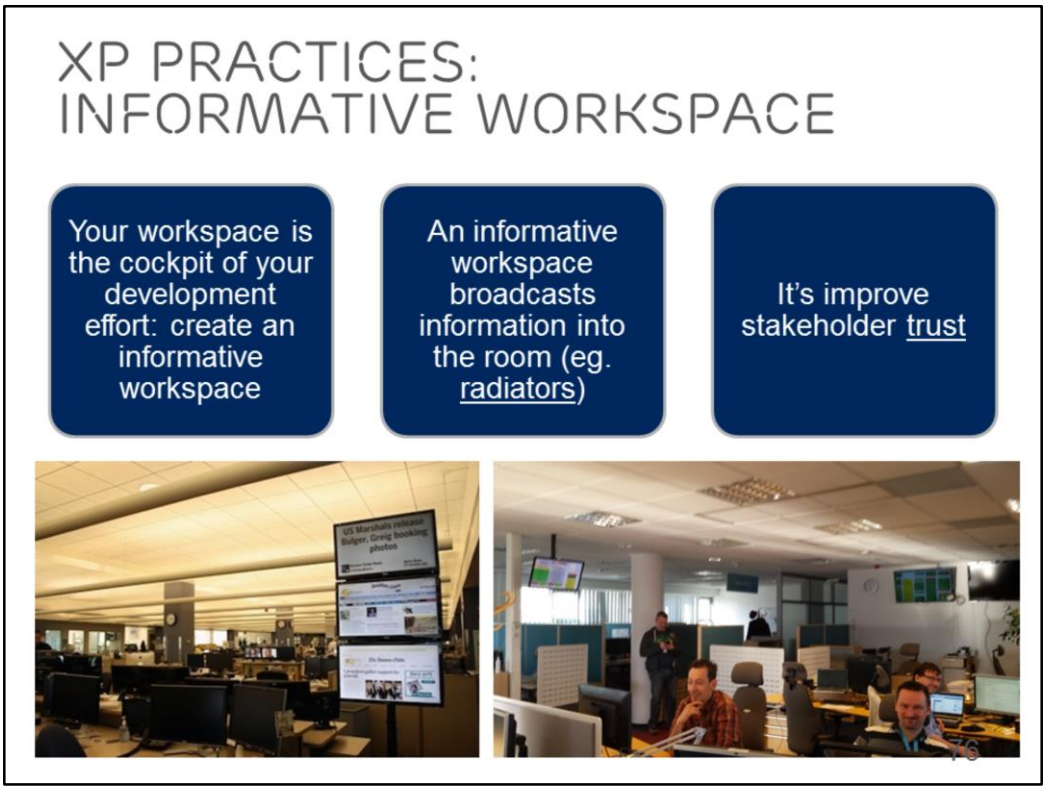
- Team releases running, tested software every iteration
- Releases are small and functional
- The Customer can evaluate or in turn, release to end users, and provide feedback
- Important thing is that the software is visible and given to the Customer at the end of every iteration

74

XP PRACTICES: SIMPLE DESIGN

- Build software to a simple design
- Through programmer testing and design improvement, keep the software simple and the design suited to current functionality
- Design steps in release planning and iteration planning
- Teams design and revise design through refactoring, through the course of the project

75



- Your workspace is the cockpit of your development effort. Just as a pilot surrounds himself with information necessary to fly a plane, arrange your workspace with information necessary to steer your project: create an informative workspace.
- An informative workspace broadcasts information into the room. When people take a break, they will sometimes wander over and stare at the information surrounding them. Sometimes, that brief zoneout will result in an aha moment of discovery.
- An informative workspace also allows people to sense the state of the project just by walking into the room. It conveys status information without interrupting team members and helps improve stakeholder trust.

XP PRACTICES: PAIR PROGRAMMING

- All production software is built by two programmers, sitting side by side, at the same machine
- All production code is therefore reviewed by at least one other programmer
- Research into pair programming shows that pairing produces better code in the same time as programmers working singly
- Pairing also communicates knowledge throughout the team

77

XP PRACTICES: TEST-DRIVEN DEVELOPMENT

- Teams practice TDD by working in short cycles of adding a test, and then making it work
- Easy to produce code with 100 percent test coverage
- These programmer tests or unit tests are all collected together
- Each time a pair releases code to the repository, every test must run correctly

78

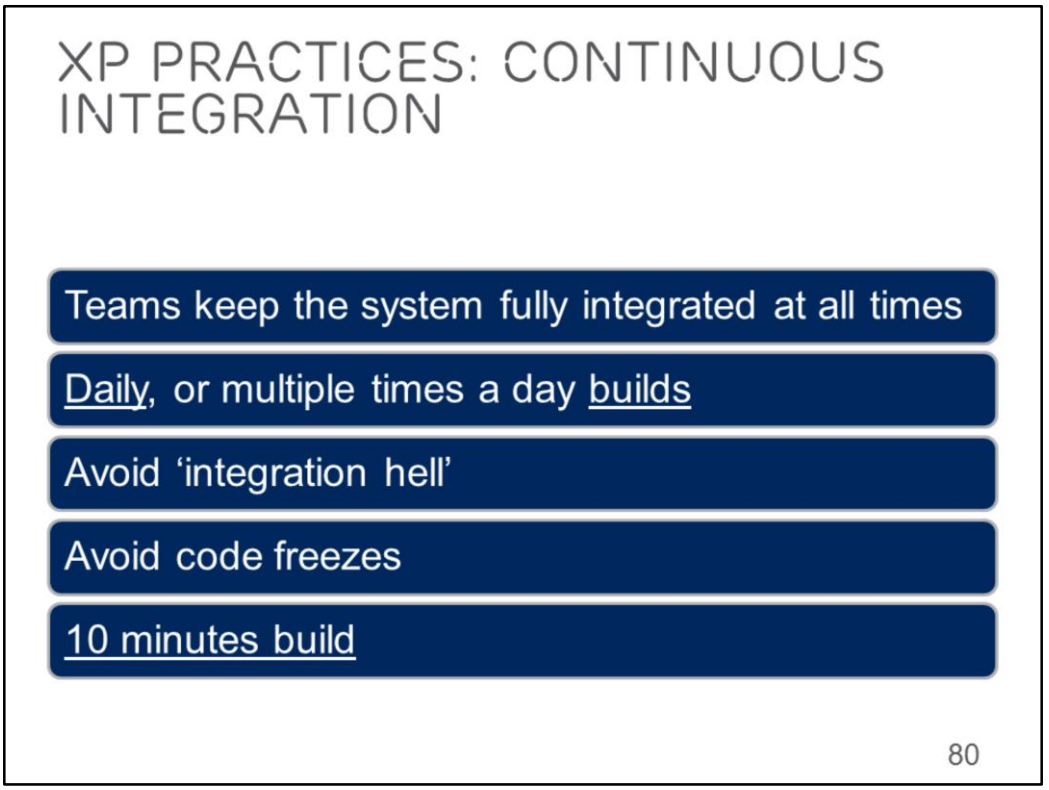
XP PRACTICES: DESIGN IMPROVEMENT

Continuous design improvement process called 'refactoring':

- Removal of duplication
- Increase cohesion
- Reduce coupling

Refactoring is supported by comprehensive testing - customer tests and programmer tests

79

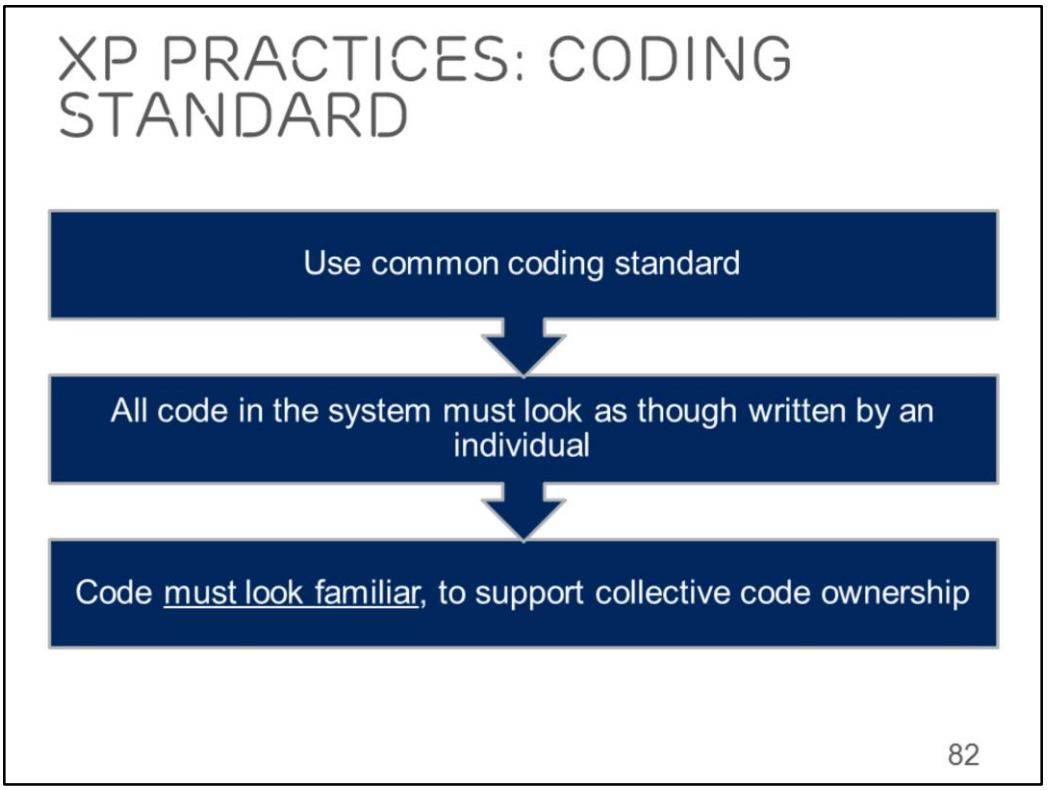


'integration hell', e.g., integrating a big chunk of code changes at the last minute which results in conflicts, and can take more time to resolve as compared to the time required to make original changes.

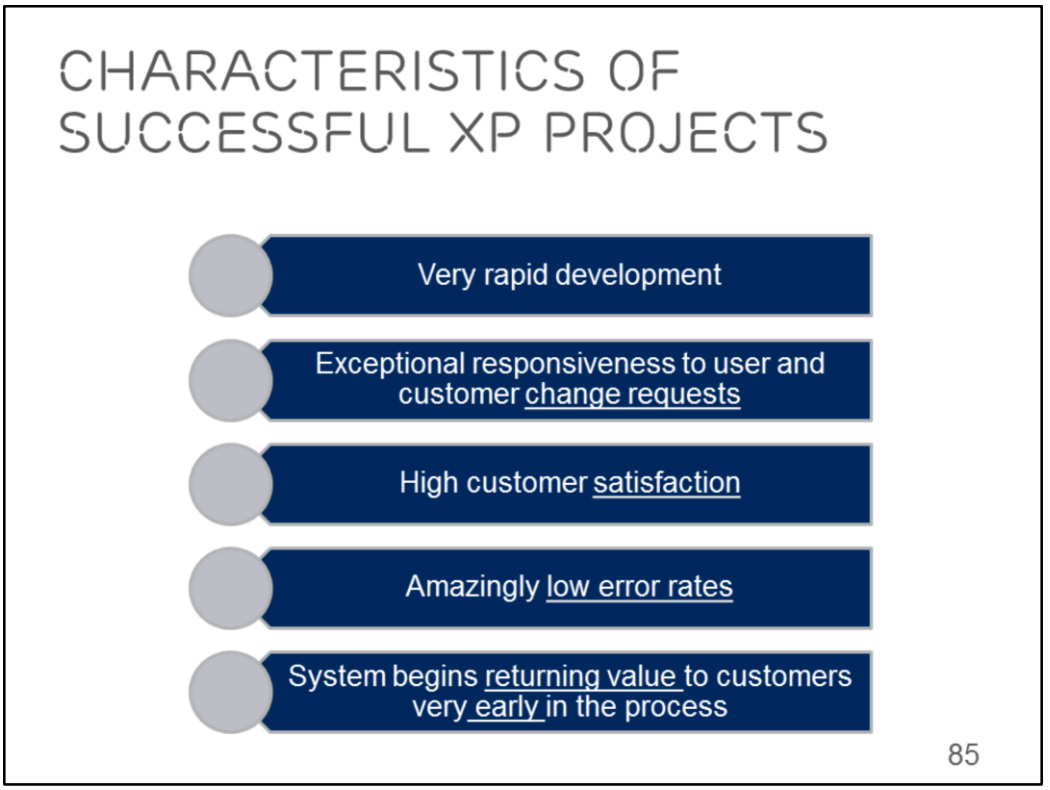
XP PRACTICES: COLLECTIVE CODE OWNERSHIP

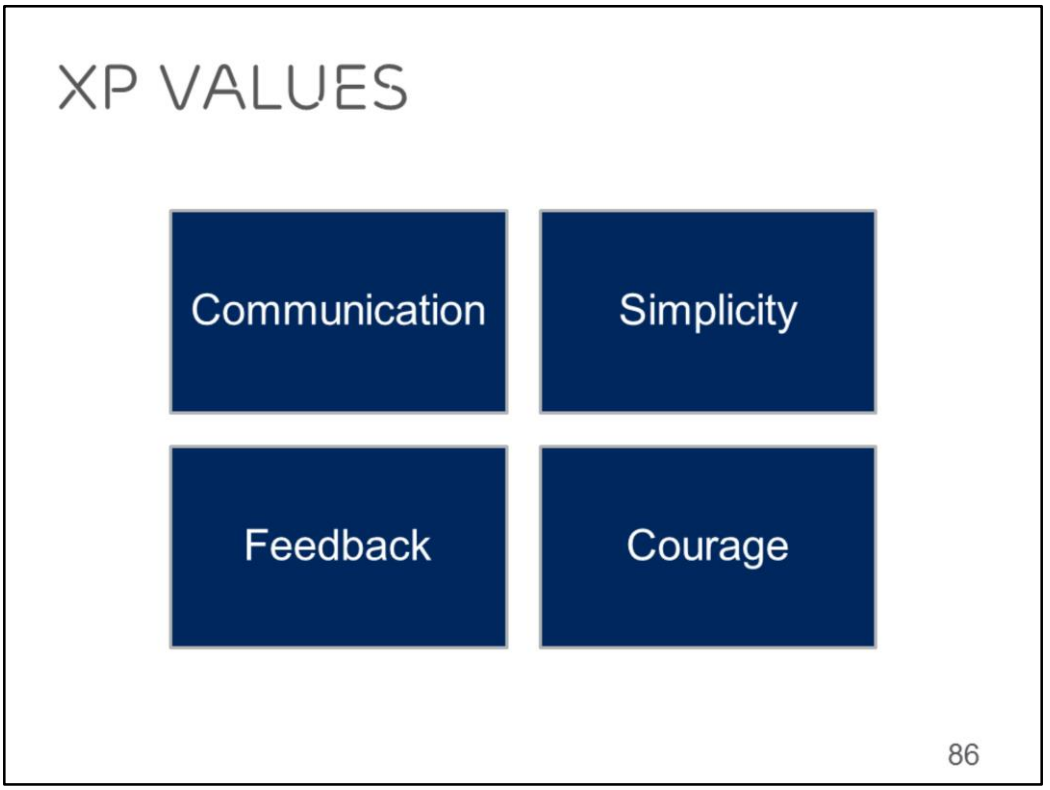
- Any pair of programmers can improve any code at any time
- All code gets the benefit of many people's attention
- Avoid duplication
- Programmer tests catch mistakes
- Pair with expert when working on unfamiliar code

81


XP PRACTICES: CODING STANDARD
Use common coding standard
All code in the system must look as though written by an individual
Code must look familiar, to support collective code ownership
82

In project management, a **death march** is a project where the members feel it is destined to fail, or requires a stretch of unsustainable overwork. The general feel of the project reflects that of an actual death march because the members of the project are forced to continue the project by their superiors against their better judgment.


CHARACTERISTICS OF SUCCESSFUL XP PROJECTS
Very rapid development
Exceptional responsiveness to user and customer change requests
High customer satisfaction
Amazingly low error rates
System begins returning value to customers very early in the process
85


XP VALUES
Communication
Simplicity
Feedback
Courage
86

XP VALUES: COMMUNICATION

- Poor communication in software teams is one of the root causes of failure of a project
- Stress on good communication between all stakeholders--customers, team members, project managers
- Customer representative always on site
- Paired programming

87

XP VALUES: SIMPLICITY

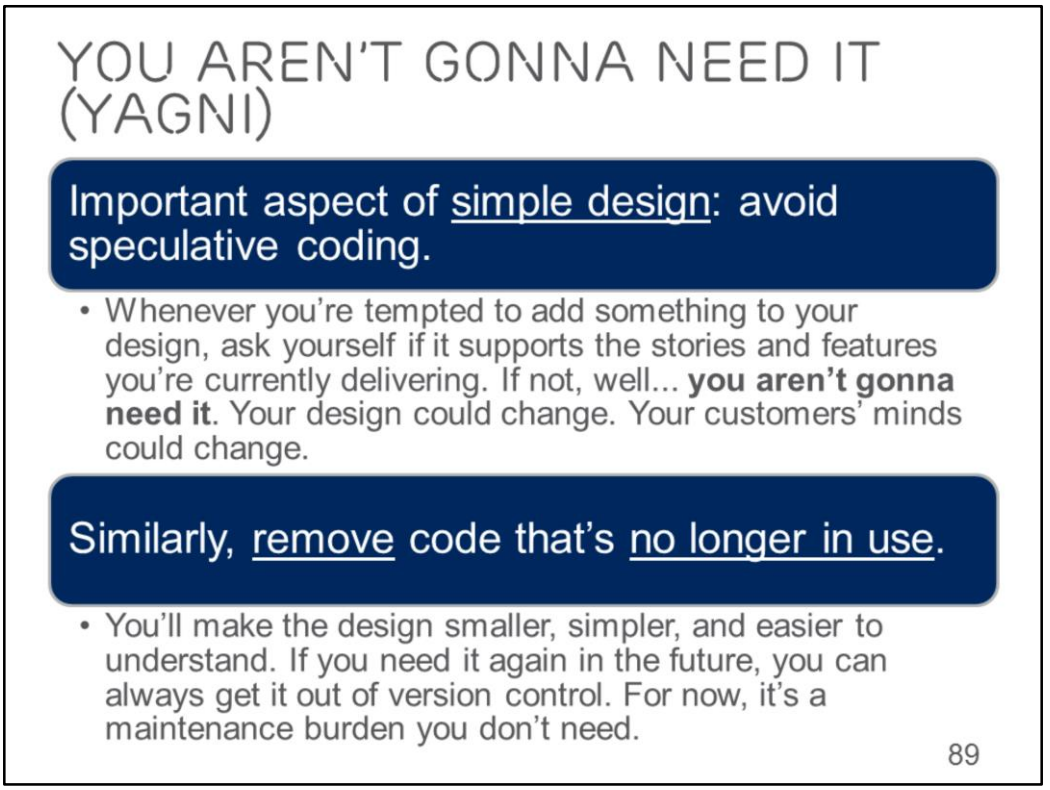
'Do the Simplest Thing That Could Possibly Work'

- Implement a new capability in the simplest possible way
- Refactor the system to be the simplest possible code with the current feature set

'You Aren't Going to Need It' (YAGNI)

- Never implement a feature you don't need now

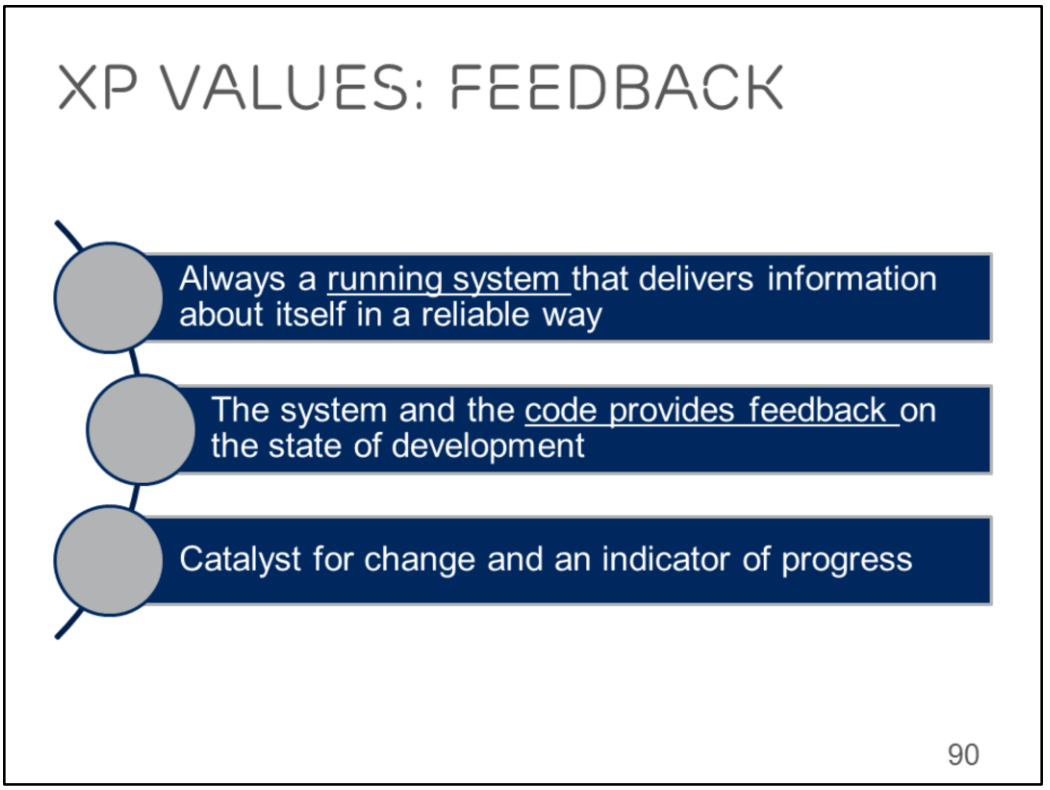
88

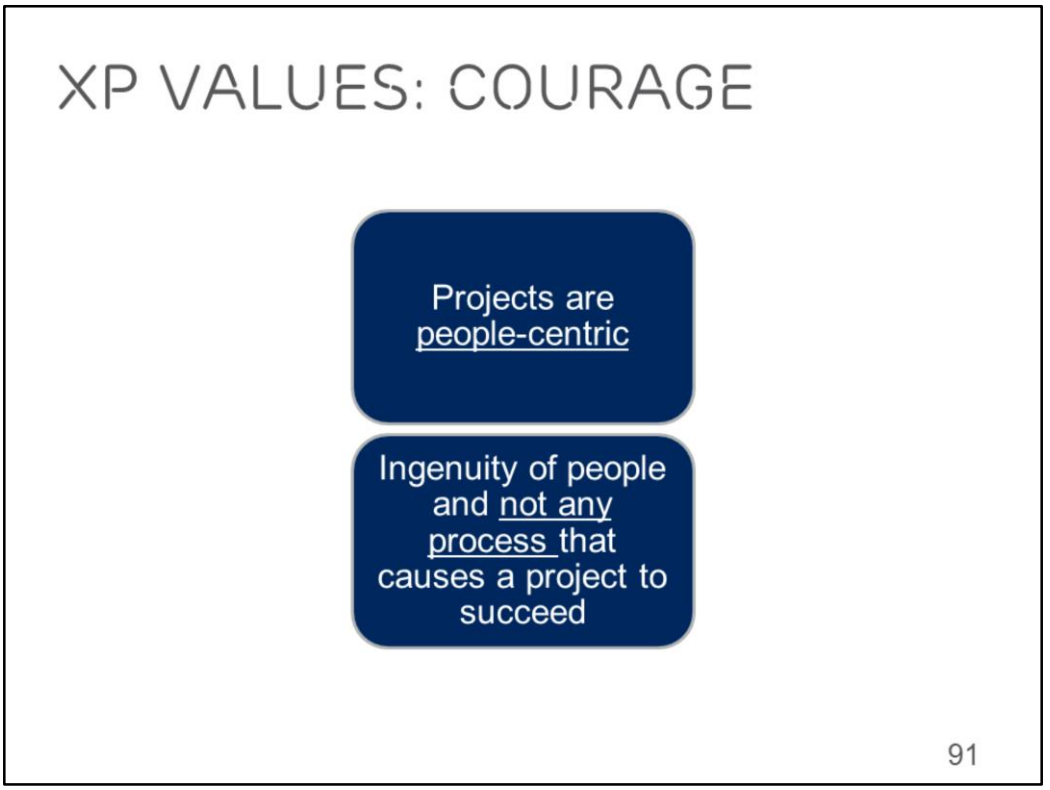


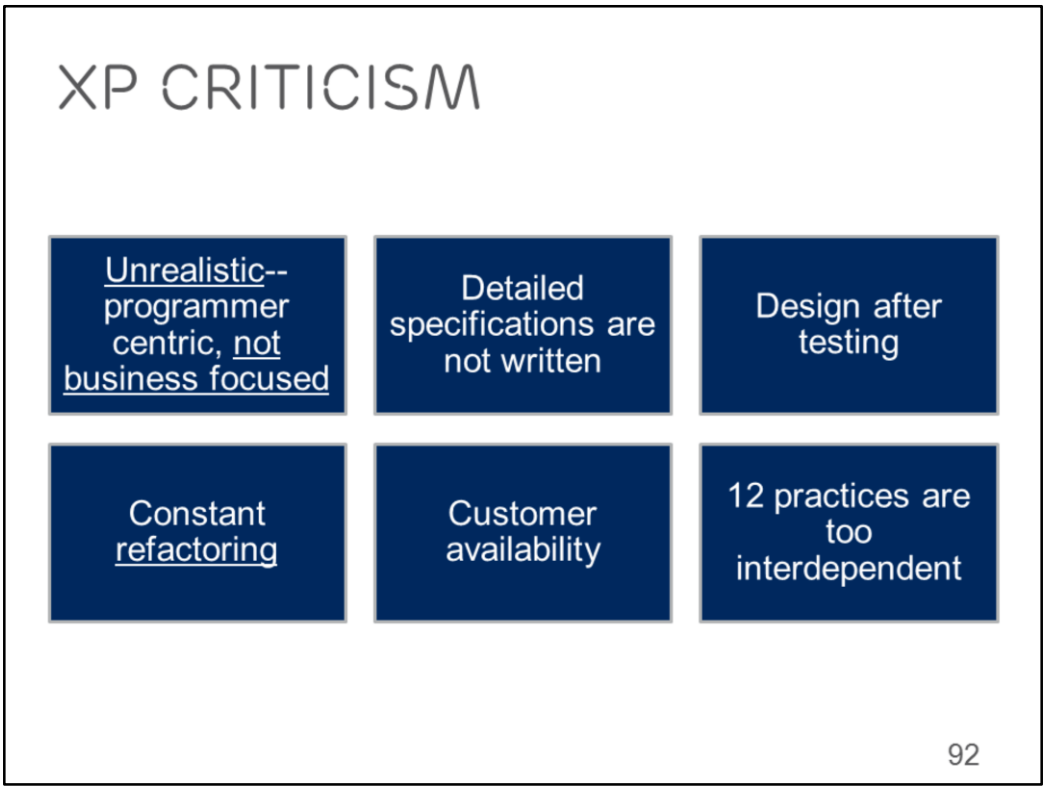
Important aspect of simple design: avoid speculative coding. Whenever you're tempted to add something to your design, ask yourself if it supports the stories and features you're currently delivering. If not, well... you aren't gonna need it. Your design could change. Your customers' minds could change.

Similarly, remove code that's no longer in use. You'll make the design smaller, simpler, and easier to understand. If you need it again in the future, you can always get it out of version control. For now, it's a maintenance burden you don't need.

We do this because excess code makes change difficult. Speculative design, added to make specific changes easy, often turns out to be wrong in some way, which actually makes changes more difficult. It's usually easier to add to a design than to fix a design that's wrong. The incorrect design has code that depends on it, sometimes locking bad decisions in place.


XP VALUES: FEEDBACK
Always a running system that delivers information about itself in a reliable way
The system and the code provides feedback on the state of development
Catalyst for change and an indicator of progress
90


XP VALUES: COURAGE
Projects are people-centric
Ingenuity of people and not any process that causes a project to succeed
91


XP CRITICISM
Unrealistic-- programmer centric, not business focused
Detailed specifications are not written
Design after testing
Constant refactoring
Customer availability
12 practices are too interdependent
92

XP THOUGHTS

The best design is the code.

Testing is good. Write tests before code. Code is complete when it passes tests.

Simple code is better. Write only code that is needed. Reduce complexity and duplication.

Keep code simple. Refactor.

Keep iterations short. Constant feedback.

93

COMMON XP MISCONCEPTIONS

No written design documentation

- *Truth: no formal standards for how much or what kind of docs are needed.*

No design

- *Truth: minimal explicit, up-front design; design is an explicit part of every activity through every day.*

XP is easy

- *Truth: although XP does try to work with the natural tendencies of developers, it requires great discipline and consistency.*

94

MORE MISCONCEPTIONS

XP is just legitimized hacking

- *Truth: XP has extremely high quality standards throughout the process*
- *Unfortunate truth: XP is sometimes **used as an excuse** for sloppy development*

XP is the one, true way to build software

- *Truth: it seems to be a sweet spot for certain kinds of projects*

95

XP SUMMARY (BY ISTQB)

Values:

- communication, simplicity, feedback, courage, respect

Principles:

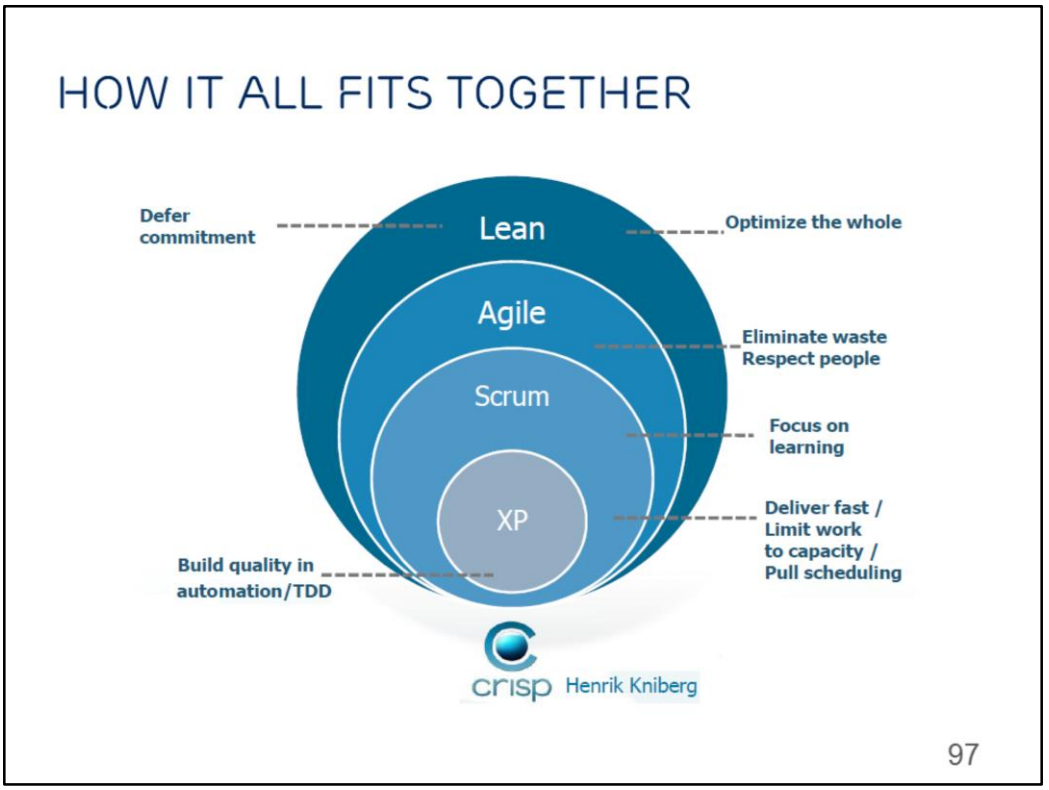
- humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, accepted responsibility

Primary practices:

- sit together, whole team, informative workspace (radiators), energized work, pair programming, stories, weekly cycle, quarterly cycle, slack (do not use 100% allocation), 10 minute build, continuous integration, test first programming, incremental design

Many other agile practices use some aspects of XP

96



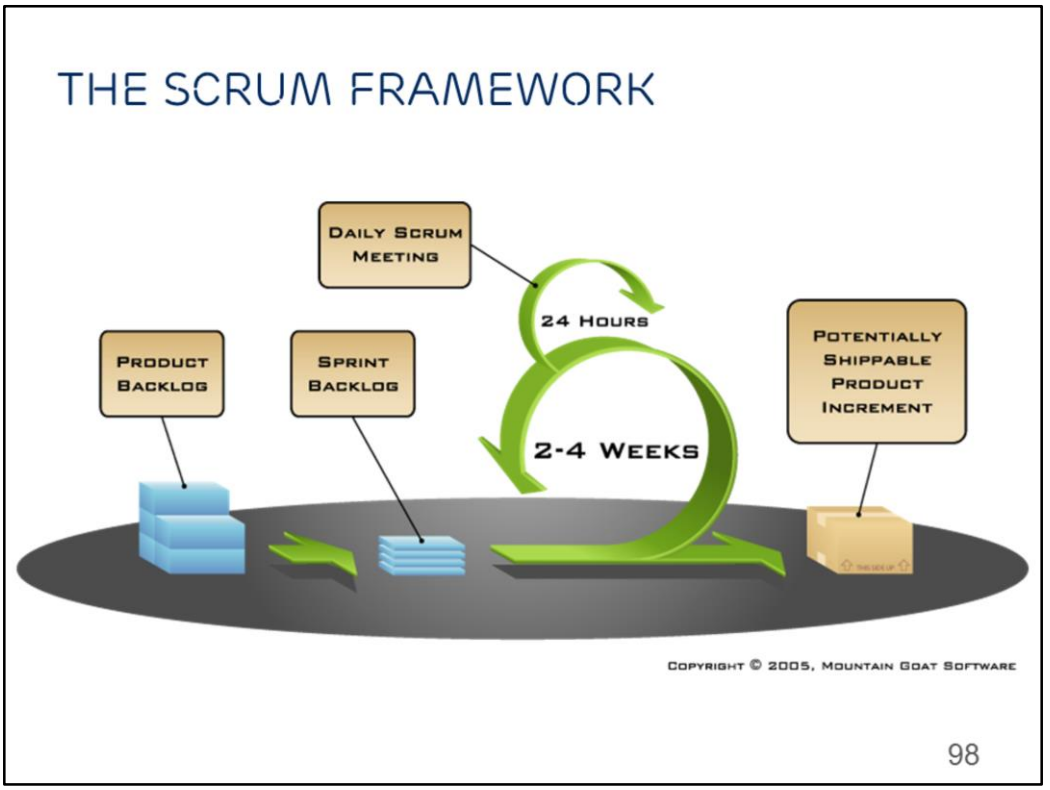
Speakers notes:

It begins with Lean as a concept, optimizing the whole vaule flow

With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)



Speakers notes:

Process description of Scrum as one example of a method that can be used within Lean and Agile product development

Speakers notes:

Product owner

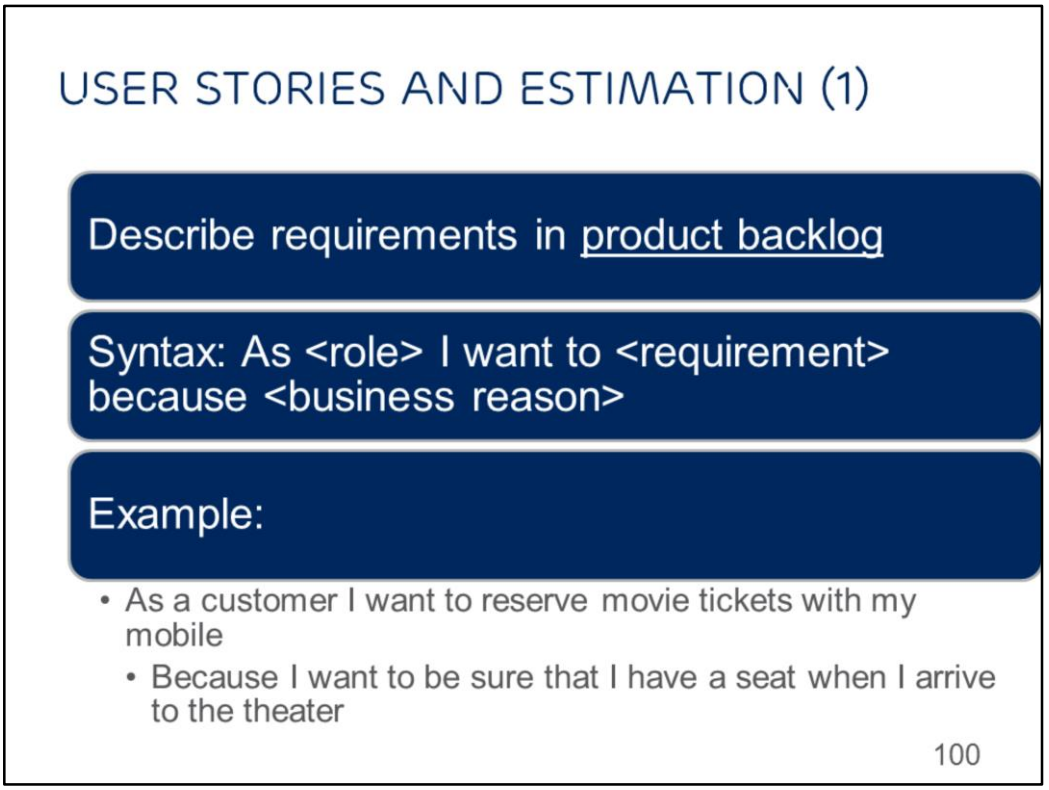
- Represents the interests of all the stakeholders
- ROI objectives
- Prioritizes the product backlog

Team

- Cross-functional
- Self-managing
- Self-organizing

Coach

- Coaches the team in the Agile and Lean process
- Challenges the team for continuous improvement
- Teaching the way we do Agile & Lean
- Ensures the following of Agile & Lean rules and practices



Speakers notes:

User stories are a way of describing customer requirements without having to create formalized requirement documents and without performing administrative tasks related to maintaining them.

A user story could describe a small feature but normally a feature is divided into several user stories.

USER STORIES AND ESTIMATION (2)

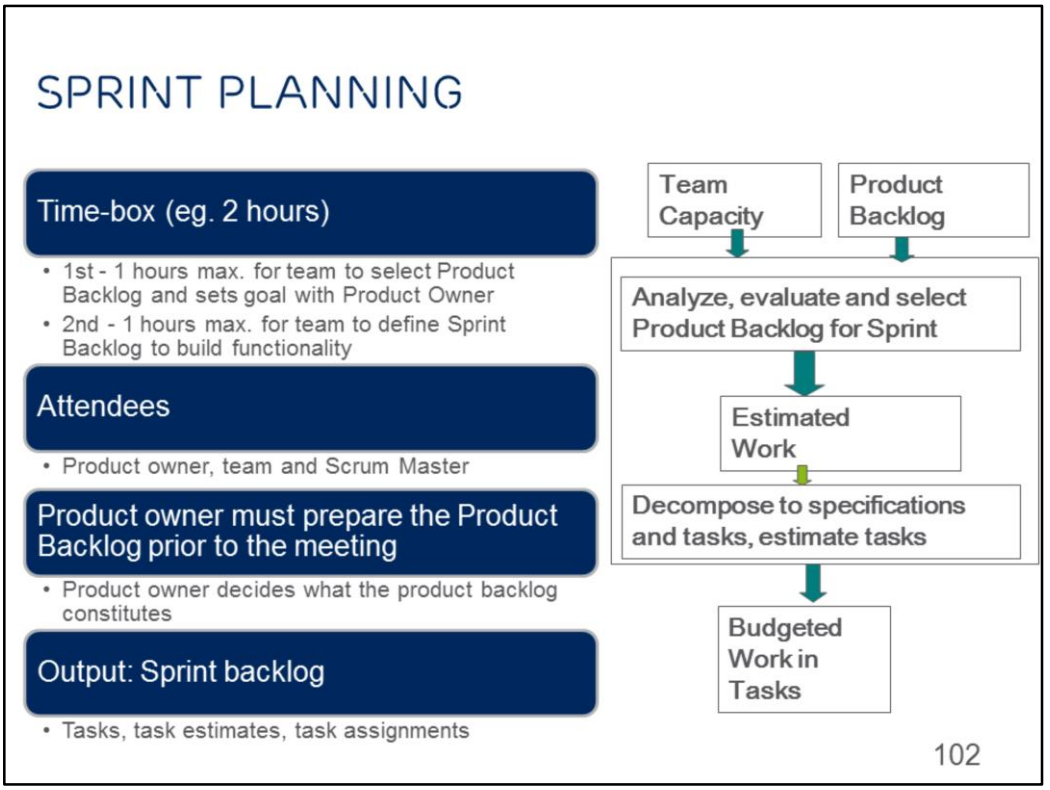
Planning poker method

- Product owner (or a stakeholder with the best knowledge) explains the story
- Team members estimate the story independently and select a card
- They show the cards simultaneously
- Explain why estimates differ
- End or go back to step 2

101

Speakers notes:

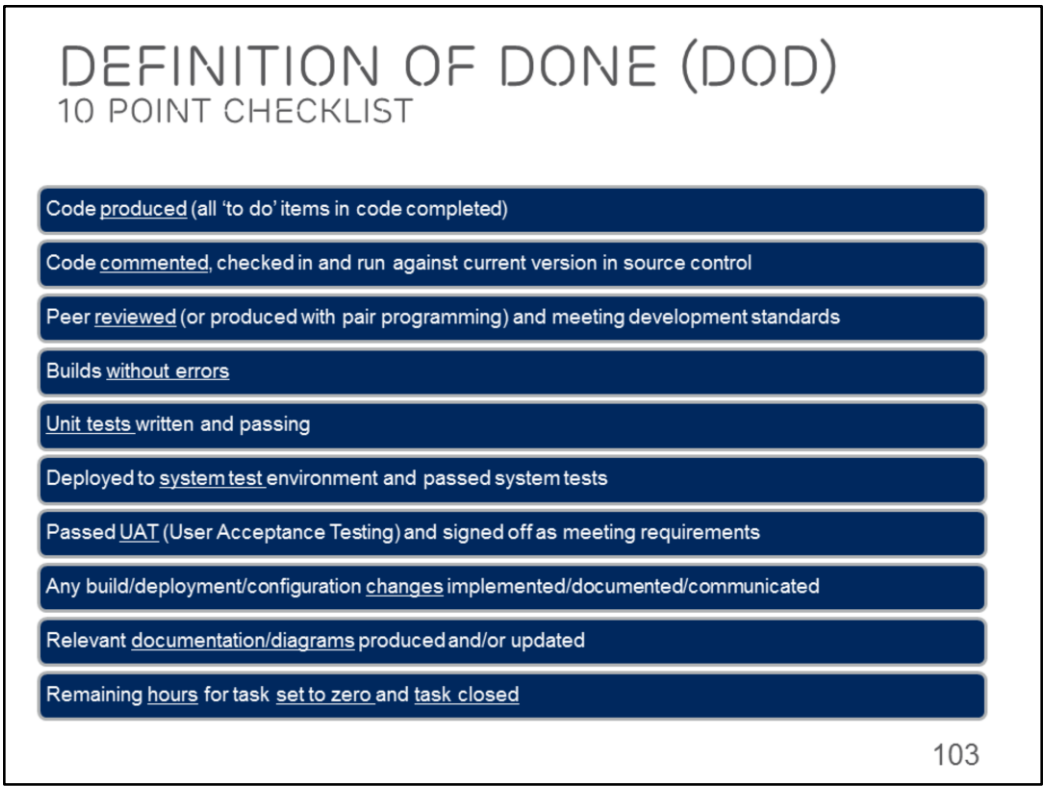
This is an exercise which will focus on the ability to cooperate in a Team



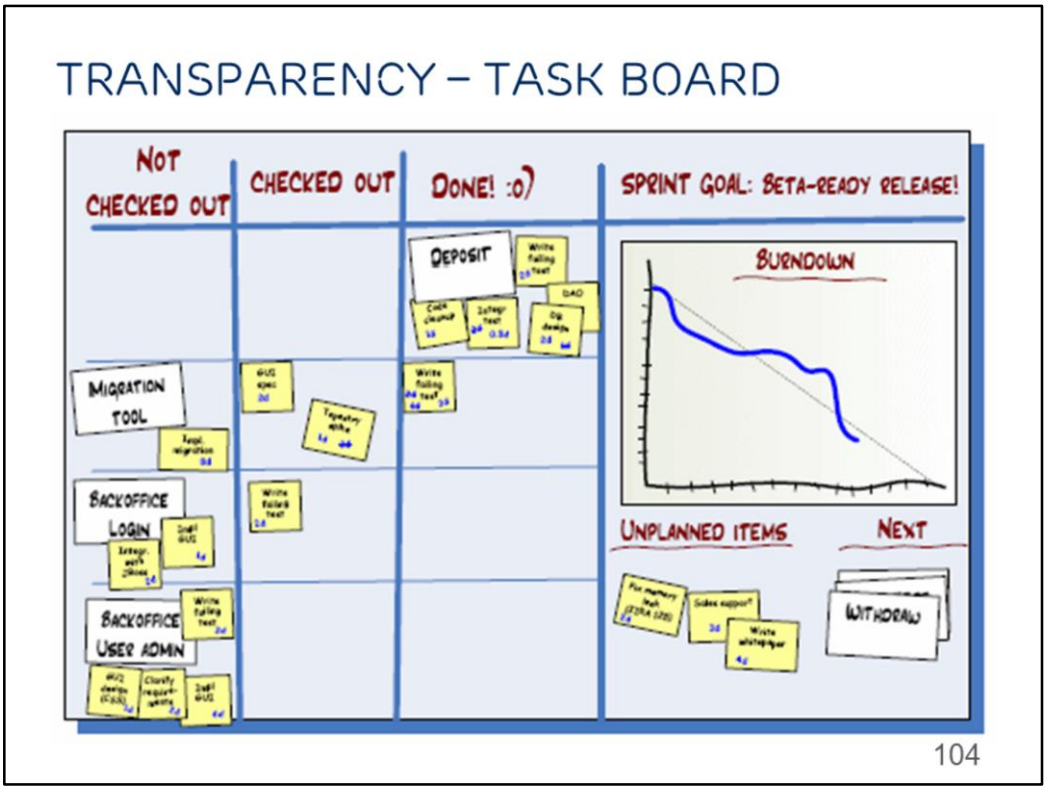
Speakers notes:

The very first time a Team work like this is set up it might take an hour or two.

This example could be a SW Team with a “normal size” of 6-8 members, (depending on the product, its maturity and complexity) that after implementation of Agile and Lean wow now can be done within a few minutes, or significantly shorter planning time.



See more at: http://www.allaboutagile.com/definition-of-done-10-point-checklist/#sthash.8rcJSONz.dpuf



Picture of task board: Kniberg, Henrik 2006. Scrum and XP from the Trenches. <http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf>

Speakers notes:

Normally the team has their Daily Scrum standing at this task board. A Daily Scrum is a:

-Daily 15 minute work meeting;

-Same place and time every day;

- Where everyone answers three questions;

    What have you done since last meeting?

    What will you do before next meeting?

    What is in your way?

-In order to find Impediments and make Decisions

The definition of Done is very important to agree upon, settle this within the Team

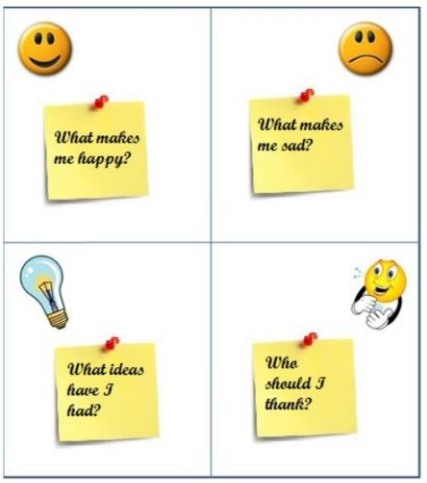
RETROSPECTIVES

Set the stage
- Focus for this retrospective

Gather data
- Ground it in facts, not opinions

Generate insights
- Observe patterns

Decide what to do
- Move from discussion to action



105

Speakers notes:

Point out that retrospectives are for the team and should thereby be run by he team, not a manager (the team should even decide if the manager is allowed to participate).

The goal is to find impediments for better ways of working. Earlier, before Agile ways of working, this was normally done once or twice a day. Now, we want to do this at the end of every sprint.

SCRUM, SUMMARY (BY ISTQB)

Practises

- Sprint (Iteration)
- Product increment
- Products backlog
- Definition of Done (DoD) – exit criteria
- Timeboxing – fix duration for iteration, fix daily meetings
- Transparency

No specific software development techniques

Roles

- Scrum Master (SM) ensures practices and rules are implemented, followed – process focused scrum theory
- Product Owner (PO) represents the customer and owns product backlog – he/she can change product backlog any time
- Development Team (3-9, self-organized) develops and tests product

Scrum does not prescribe testing approach

106

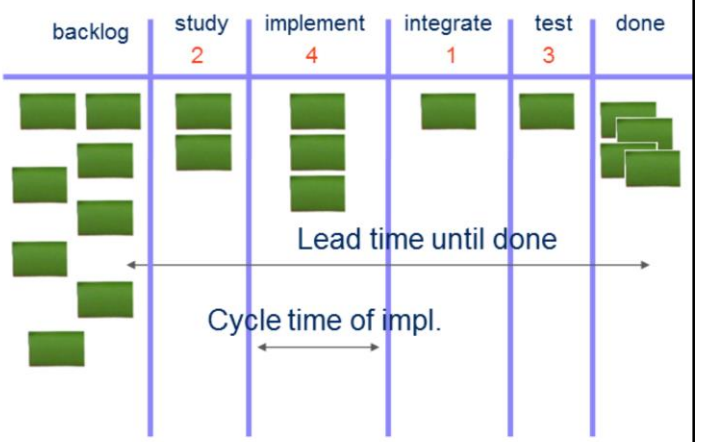
看板 – KANBAN CARDS LIMIT EXCESS WORK IN PROGRESS

- 看板 – kanban literally means “visual card,” “signboard,” or “billboard.”
- Toyota originally used Kanban cards to limit the amount of inventory tied up in “work in progress” on a manufacturing floor
- kanban cards act as a form of “currency” representing how WIP (Work In Progress) is allowed in a system.
- Kanban is an emerging process framework that is growing in popularity since it was first discussed at Agile 2007 in Washington D.C.

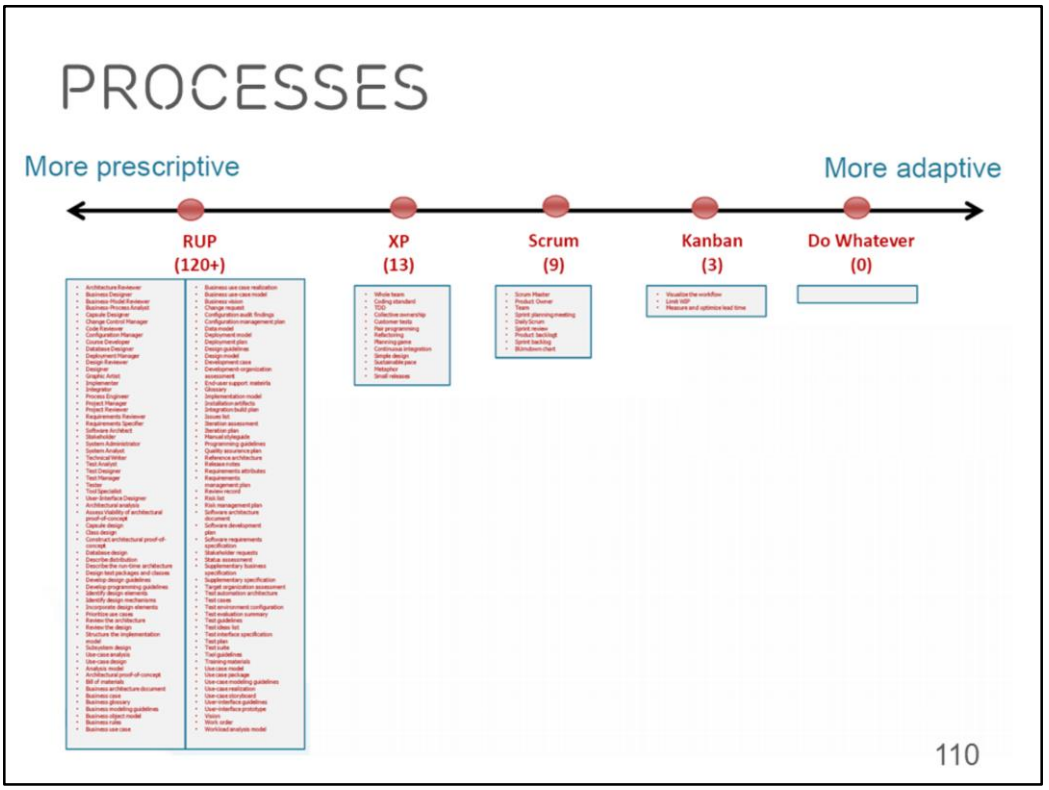
107

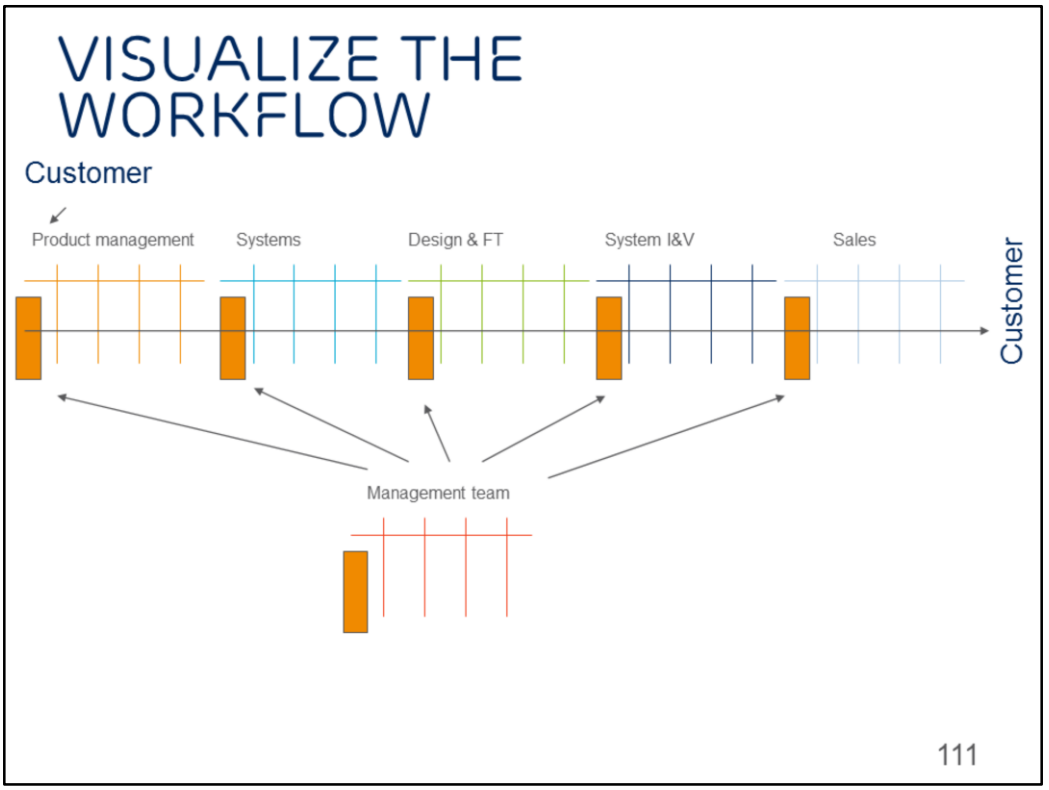
KANBAN BASIC RULES

- Visualize the workflow
- Limit Work In Progress (WIP)
- Measure and optimize lead time

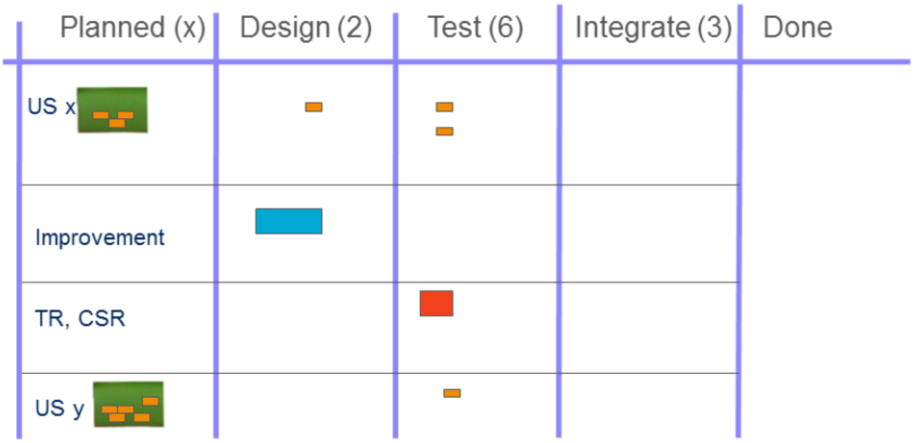


109


PROCESSES
More prescriptive
More adaptive
RUP
(120+)
XP
(13)
Scrum
(9)
Kanban
(3)
Do Whatever
(0)
110


VISUALIZE THE WORKFLOW
Customer
Product management
Systems
Design & FT
System I&V
Sales
Customer
Management team
111

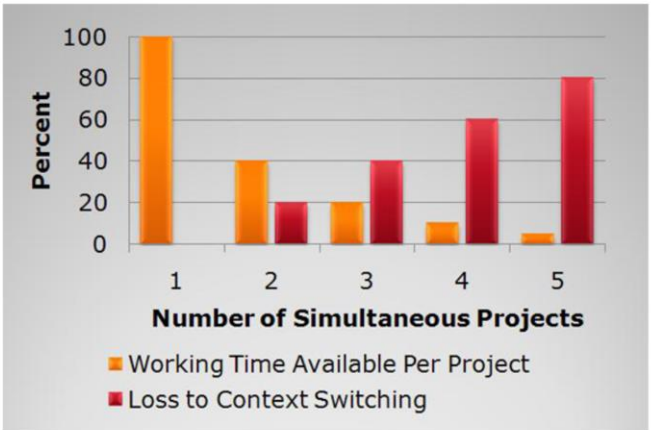
VISUALIZE THE WORKFLOW



112

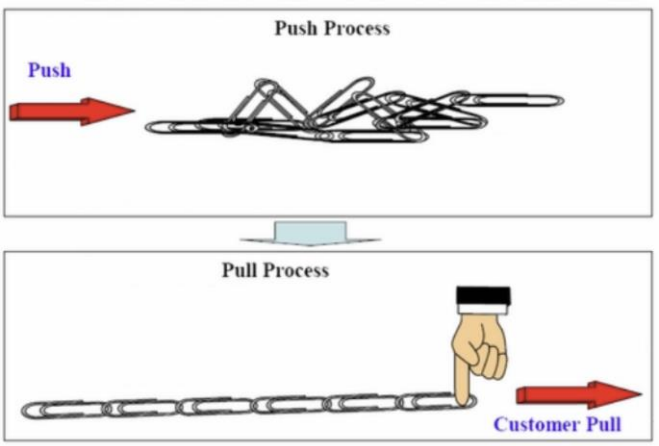
LIMITING WORK IN PROGRESS

20% time is lost to context switching per task, so fewer tasks means less time lost *(from Gerald Weinberg, Quality Software Management: Systems Thinking)*



113

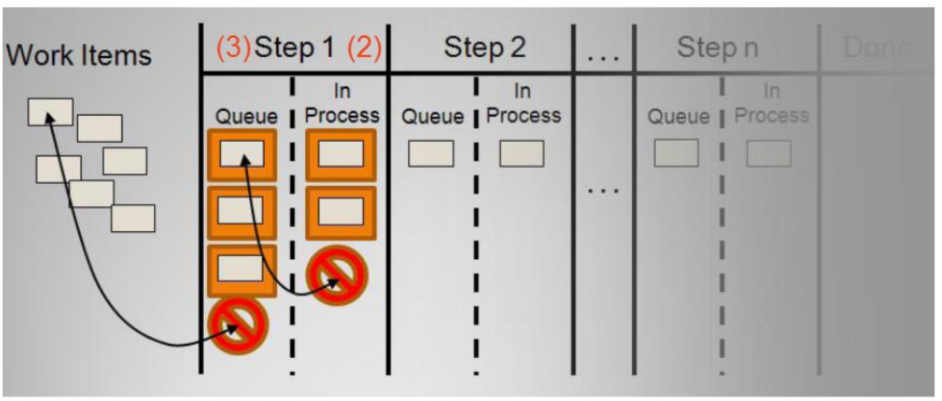
LIMITING WORK IN PROGRESS



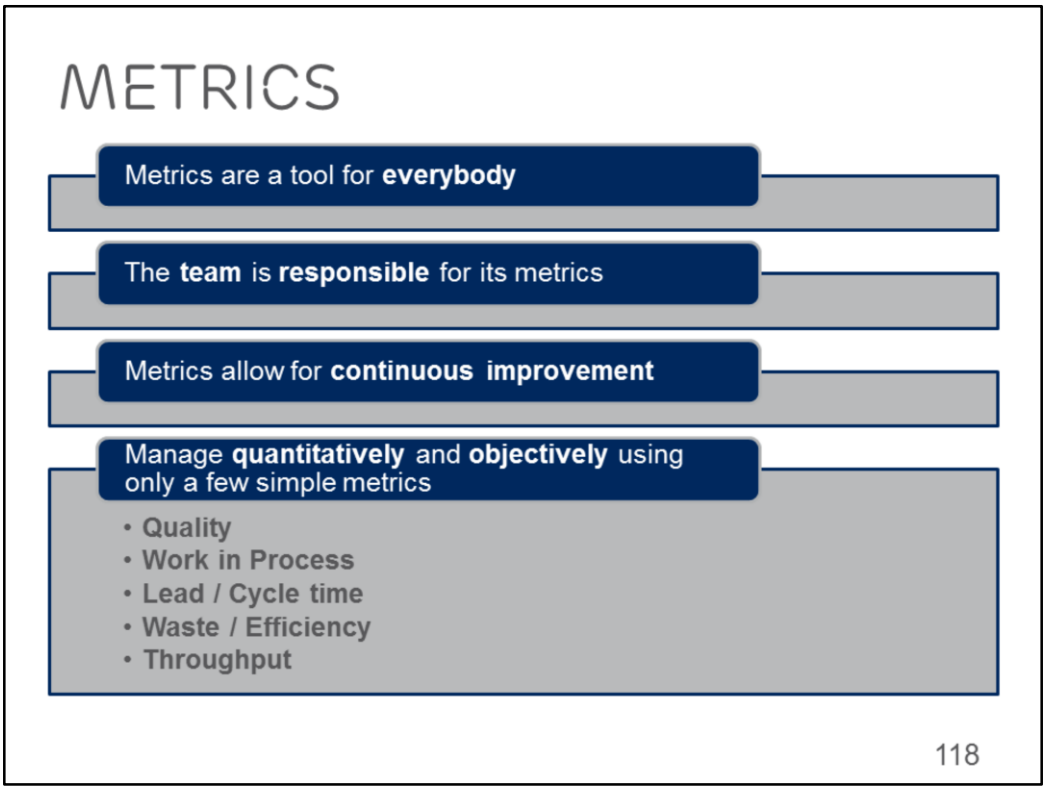
114

LIMITING WORK IN PROGRESS

New work items can only be pulled into a state if there is capacity under the WIP limit.



115


METRICS
Metrics are a tool for everybody
The team is responsible for its metrics
Metrics allow for continuous improvement
Manage quantitatively and objectively using only a few simple metrics
• Quality
• Work in Process
• Lead / Cycle time
• Waste / Efficiency
• Throughput
118

LITTLE'S LAW FOR QUEUING THEORY

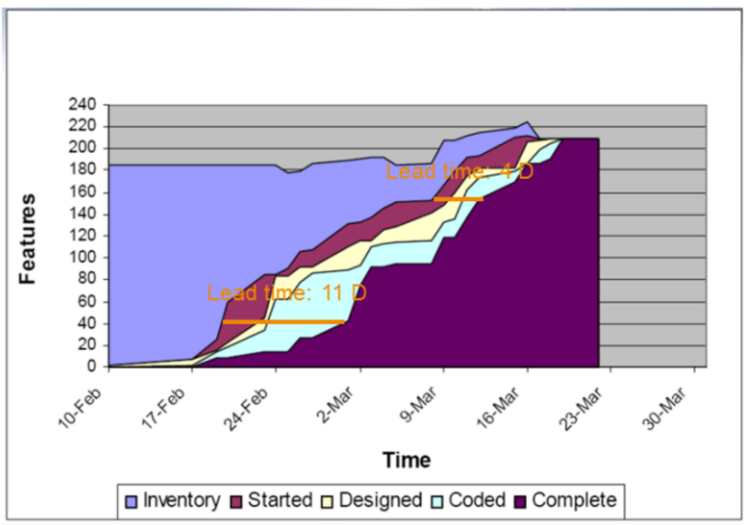
Total Cycle Time = Number of Things in Progress / Average Completion Rate

The only way to reduce cycle time is by either reducing the WIP, or improving the average completion rate.

- Achieving both is desirable.
- Limiting WIP is easier to implement by comparison.

119

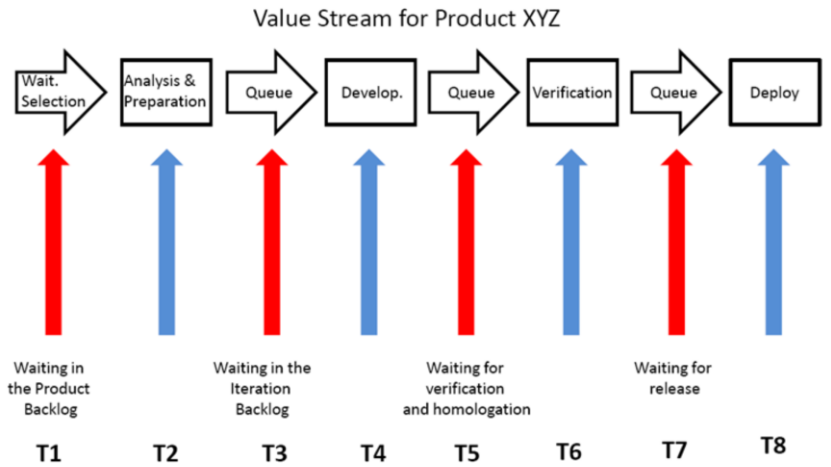
USE CUMULATIVE FLOW DIAGRAMS TO VISUALIZE WORK IN PROGRESS



www.agilemanagement.net/Articles/Papers/BorConManagingwithCumulat.html

120

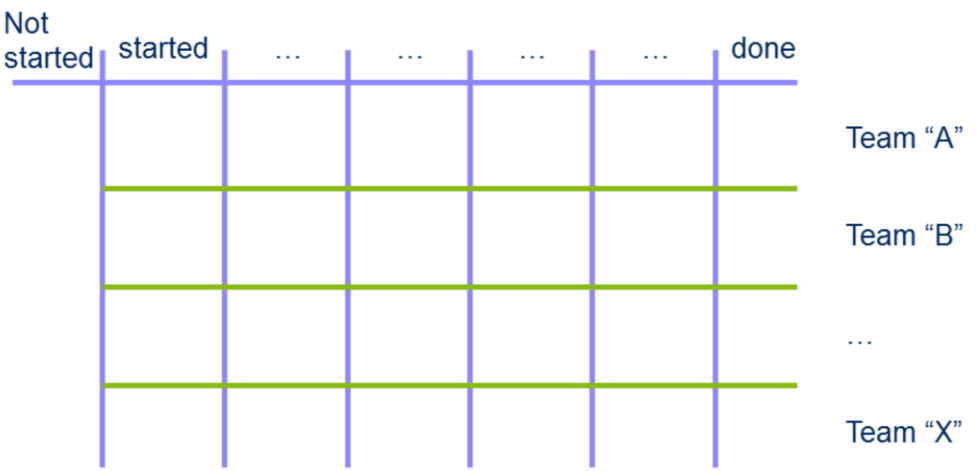
VALUE STREAM MAPPING



$$\text{Eficiency (\%)} = \frac{\text{TValue} * 100}{\text{TValue} + \text{TWaste}}$$

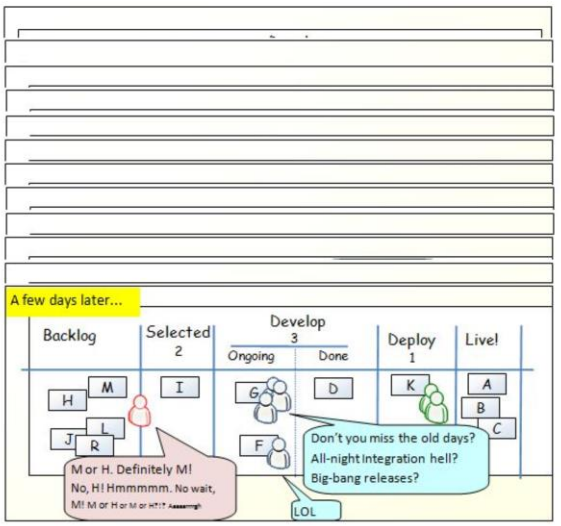
121

SCALING – SWIM LANES

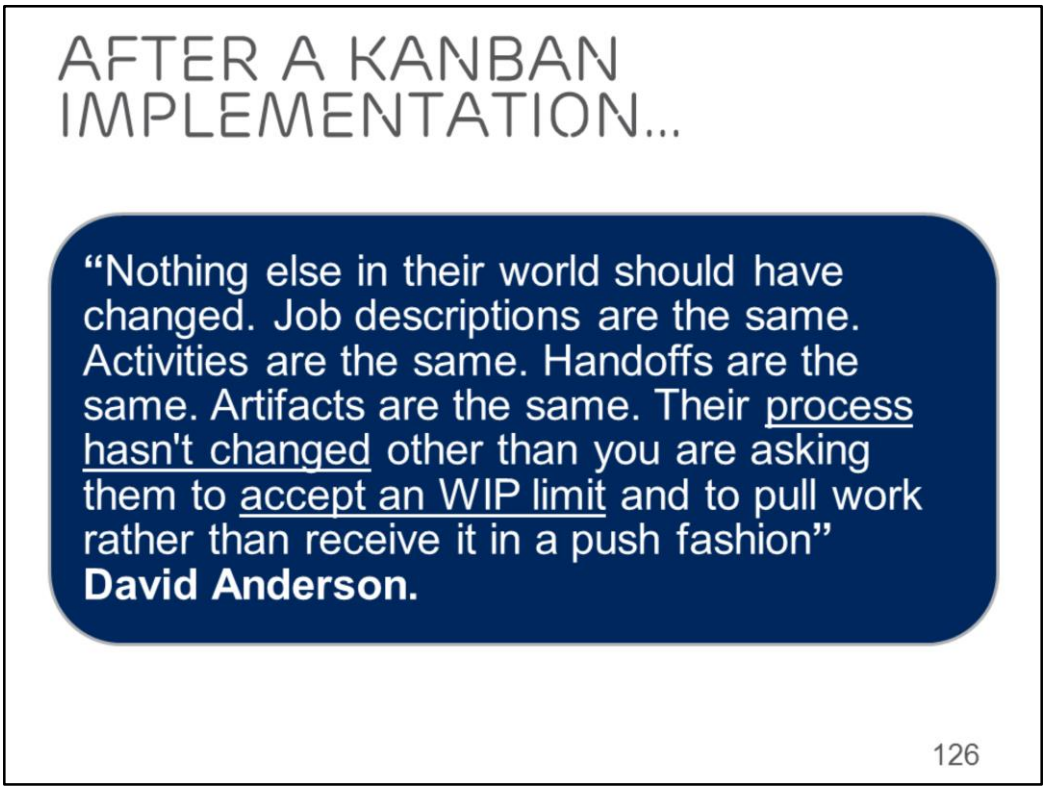


123

ONE DAY IN KANBAN LAND



125


AFTER A KANBAN IMPLEMENTATION...
"Nothing else in their world should have changed. Job descriptions are the same. Activities are the same. Handoffs are the same. Artifacts are the same. Their process hasn't changed other than you are asking them to accept an WIP limit and to pull work rather than receive it in a push fashion"
David Anderson.
126

SOURCES

- http://www.limitedwipsociety.org/
- http://www.crisp.se/henrik.kniberg/kanban-vs-scrum.pdf

127

KANBAN SUMMARY (BY ISTQB)

Optimize flow of work in value-added chain

Instruments:

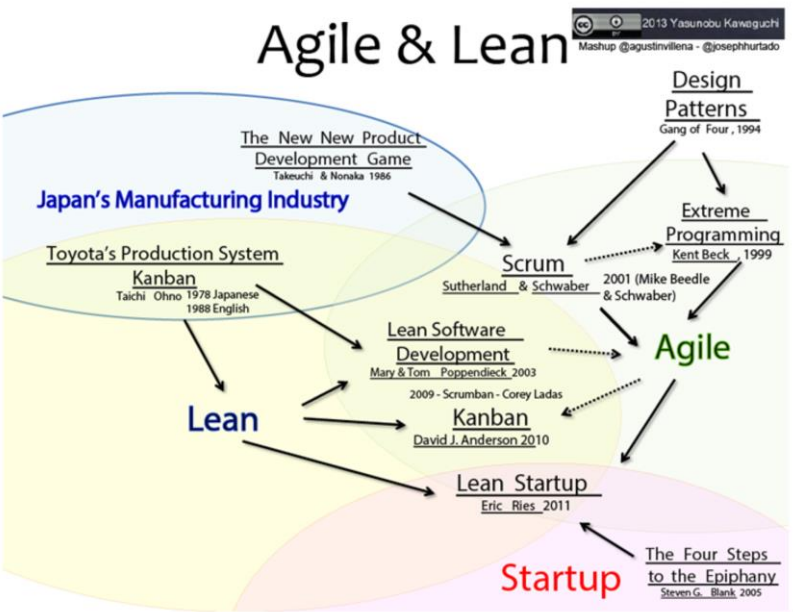
- Kanban board
- Work-in-progress limit
- Lead time

Both Kanban and Scrum provide status transparency and backlogs, but:

- Iteration is optional in Kanban
- Items can be delivered one at a time or in a release
- Timeboxing is optional

128

LEAN & AGILE



129

DOCUMENTATION SYSTEMS

- Helps in generating on-line documentation or offline reference manual from documented source files.
- Combine source code with documentation and other reference materials
- Make it easier to keep the documentation and code in sync
- We will see:
  - Doxygen
  - Javadoc
  - T3Doc

130

DOXYGEN

Source code documentation generator tool, Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extend D.

Most useful tags:

- \file
- \author
- \brief
- \param
- \returns
- \todo (not used in assignments)

131

JAVADOC

Attach special comments, called *documentation comment* (or *doc comment*) to classes, fields, and methods. /** ... */

Use a tool, called *javadoc*, to automatically generate HTML pages from source code.

Javadoc Tags: Special keyword recognized by javadoc tool. Common Tags:

• @author	Author of the feature
• @version	Current version number
• @since	Since when
• @param	Meaning of parameter
• @return	Meaning of return value
• @throws	Meaning of exception
• @see	Link to other feature

132

T3DOC

TTCN-3 source code tagging

Standard: ETSI ES 201 873-10

Example

```
/*****************************************
** @desc XYZ                            **
** Initialize to pre-trial defaults.    **
**                                      **
123
*****************************************/
```

133

TTCN-3 DOCUMENTATION TAGS

	Simple Data Types	Structured Data Types	Component Types	Port Types	Modulepars	Constants	Templates	Signatures	Functions (TTCN-3 and external)	Altsteps	Test Cases	Modules	Groups	Control Parts	Component local definitions	Used in implicit form (see clause 7)	Embedded in other tags
@author	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@config											X						
@desc	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@exception								X								X	
@member		X[1]	X	X	X[1]	X[1]	X[1]									X	
@param							X	X	X	X	X					X	
@priority											X						
@purpose											X	X					
@remark	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@reference	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@requirement									X	X	X	X					
@return								X	X							X	
@see	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@since	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@status	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@url	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@verdict									X	X	X	X					
@version	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		

NOTE: [1] Preceding language elements of record, set, union or enumerated types only.

134