



# AGILE SOFTWARE ENGINEERING: XP, ESTIMATION & HOMETGROUND

JOHN STOUBY PERSSON



AALBORG UNIVERSITY  
DENMARK

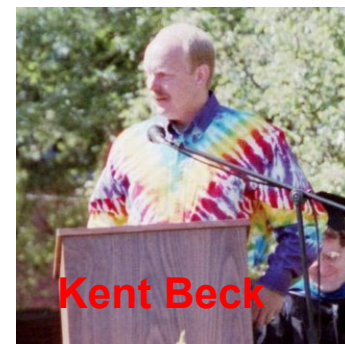
# Lecture objectives

An overview of eXtreme Programming (XP)

Basic skills in estimation of software development tasks

Prepared to chose and adapt a development method for a specific project

# The **extreme** in eXtreme Programming (XP)



If code reviews are good, we'll review code all the time (**pair programming**).

If testing is good, everybody will test all the time (**unit testing**), even the customers (functional testing).

If design is good, we'll make it part of everybody's daily business (**refactoring**).

If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the **simplest** thing that could possibly work).

If architecture is important, everybody will work defining and refining the architecture all the time (**metaphor**).

If integration testing is important, then we'll integrate and test several times a day (**continuous integration**).

If short iterations are good, we'll make the iterations really, really short - seconds and minutes and hours, not weeks and months and years (**the Planning Game**).

*Kent Beck 1999, Extreme programming explained: embrace change, Addison-Wesley*

# XP techniques

Concept	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found.

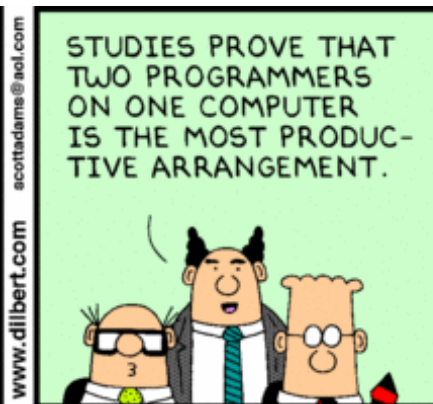
## XP techniques, contd.

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	40 h work week. Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. The customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

## XP: On-site customer

- A real customer sits with the team
- Answers the developers' questions
- Writes user stories and functional tests
- Provides value to the project by being ready available as a knowledgeable source on everything to do with the customers and users and their business

# XP: Pair Programming



## XP - User story template

User Role (*Who?*)

As a <type of user>, I can <immediate goal> so that <reason>.

Desired Function (*What?*)

End Result (*Why?*)

*Who, What, Why...  
what's **not** here?*

[DRAMATIC SHIFT!!!] By utilizing user stories (this template), we're saying that we're going to work collaboratively with the customer to discover what they want (during analysis) and deliver it during the sprint. The past way was for marketing (or customer) to have tightly defined requirements and we would chunk those up and meet them!!



## *XP - User story example*

- **As a user** I want to be able to set the alarm on my cell phone so I can get up in the morning.
- **As a snoozer** I want to be able to activate 'snooze' when the alarm goes off, so I can sleep 10 minutes more.
- **As a user** I want to set the alarm so I can get up at the same time every morning.

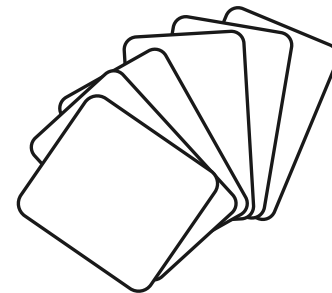
## XP: Story-based planning

- The planning game is based on user stories
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story
- Stories are assigned 'effort points' (also called story points) reflecting their size and difficulty of implementation
- The number of effort/story points implemented per day is measured giving an estimate of the team's 'velocity'
- This allows the total effort required to implement the next release to be estimated


# Scrum - Planning Poker

## Getting started

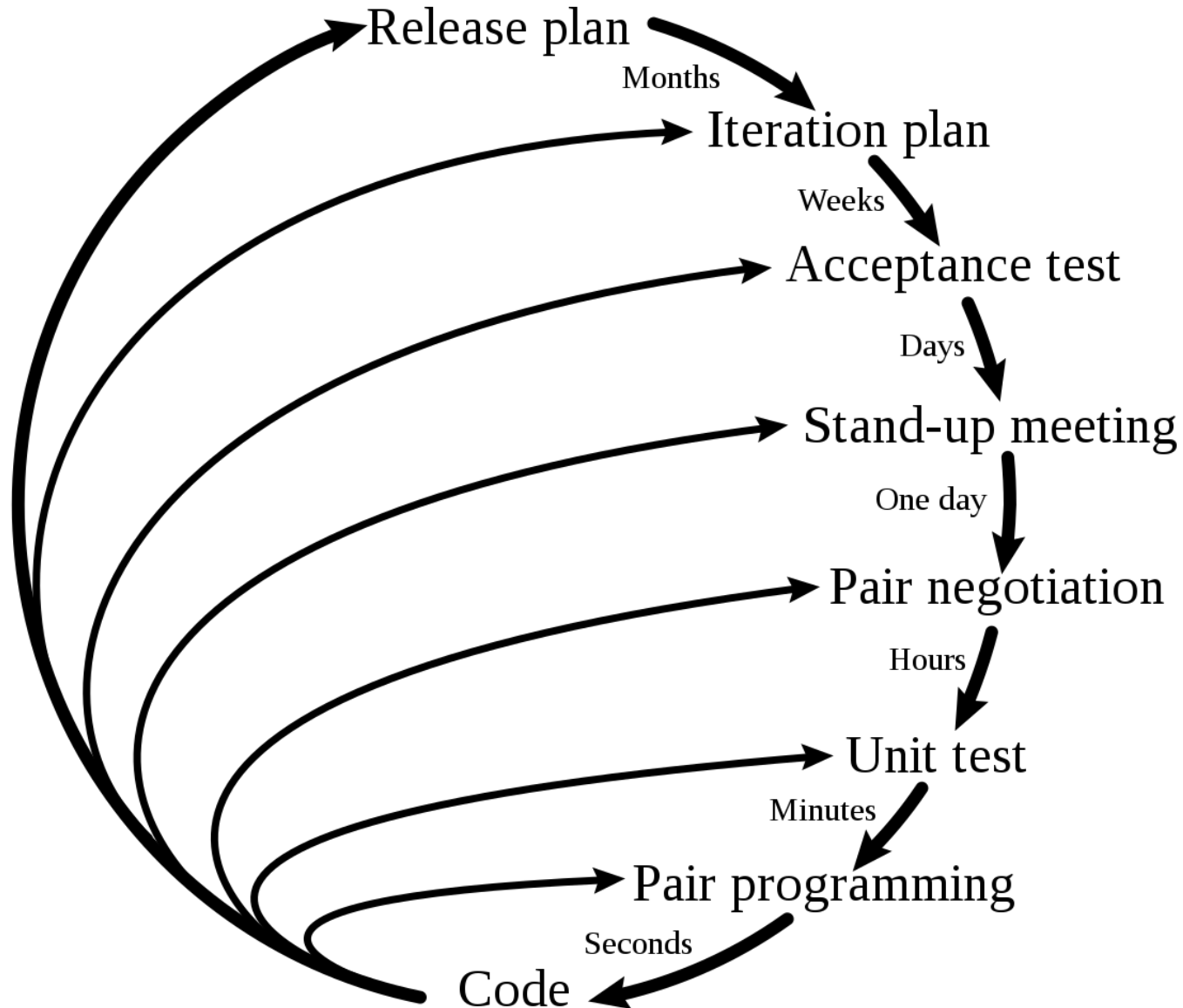
1. Each person in your team should have a set of cards with the numbers “1, 2, 3, 5, 8, 13”
2. Each team member holds a set up so only they can see them
3. Estimates are made by the people doing the work, the PO will be present to answer questions and clarify product needs.
4. Estimates are done for each story individually
5. All team members show their selected card at the same time
6. If difference is more than 2 numbers in the series, the person with lowest and highest number present their reasons
7. Entire teams replay and continue until at most 1 number difference, e.g. lowest 5 highest 8



# Test Driven Development (TDD)

1. Identify your next small step - in XP you'll take small steps and run fast
  2. Think of how you will test that you've accomplished this step.
  3. Write the code for one of your tests.
  4. Write just enough code that your test compiles.
  5. Your test should be failing at this point.
  6. Now you write just enough code to make your test pass. You may be tempted to take care of other issues, but you need to stay focused on your current goal. Pass the test.
  7. Look for the next test you have to write. [i.e. back to (2)]
  8. Now return to step (1) and continue with the next small step.
- 

# Planning/feedback loops



# BALANCING PLAN-DRIVEN AND AGILE

# Boehm & Turner's Observations

1. Neither agile nor plan-driven methods provide a silver bullet.
2. **Agile and plan-driven methods have home grounds where one clearly dominates the other.**
3. Future trends are toward application developments that need both agility and discipline.
4. Some balanced methods are emerging.
5. It is better to build your method up than to tailor it down.
6. Methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communications, and expectations management.

# Home Grounds (Definition)

	Agile	Plan-driven
APPLICATION		
Primary goals	Rapid value; responding to change	Predictability; stability; high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable; low change; project/organization-focused
MANAGEMENT		
Customer relations	Dedicated on-site customers, where feasible; focused on prioritized increments	As-needed customer interactions; focused on contract provisions; increasingly evolutionary
Planning/control	Internalized plans; qualitative control	Documented plans; quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge



# Home Grounds definition, contd.

## TECHNICAL

Requirements	Proritized informal stories and test cases; undergoing unforeseeable change	Formalized project; capability; interface; quality; foreseeable evolution requirements
Development	Simple design; short increments; refactoring assumed inexpensive	Architect for parallel development; longer increments; refactoring assumed expensive
Test	Executable test cases define requirements	Documented test plans and procedures

## PERSONNEL

Customers	Dedicated; colocated CRACK (Collaborative, Representative, Authorized, Committed, Knowledgable) performers	CRACKperformers, not always colocated
Developers	> 30% FT Cockburn level 2 and 3 experts; no 1b or -1	50% level 3s early; 10% throughout; 30% 1b's; 0 -1
Culture	Comfort and empowerment via many degrees of freedom; thriving on chaos	Comfort and empowerment via framework of policies and procedures; thriving on order <sup>17</sup>

# Cockburn's developer levels

Level	Characteristics
3	Able to revise a method (break its rules) to fit an unprecedented new situation
2	Able to tailor a method to fit a precededented new situation
1A	With training, able to perform discretionary method steps (e.g., sizing stories to fit increments, composing patterns, compound refactoring, complex COTS integration). With experience can become Level 2.
1B	With training, able to perform procedural method steps (e.g. coding a simple method, simple refactoring, following coding standards and CM procedures, running tests). With experience can master some Level 1A skills.
-1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

Drawing on the three levels of understanding in Aikido (Shu-Ha-Ri), Alistair Cockburn has identified three levels of software method understanding that can help sort out what various levels of people can be expected to do within a given method framework [2]. We have taken the liberty of splitting his Level 1 to address some distinctions between agile and disciplined methods, and adding an additional level to address the problem of method-disrupters.

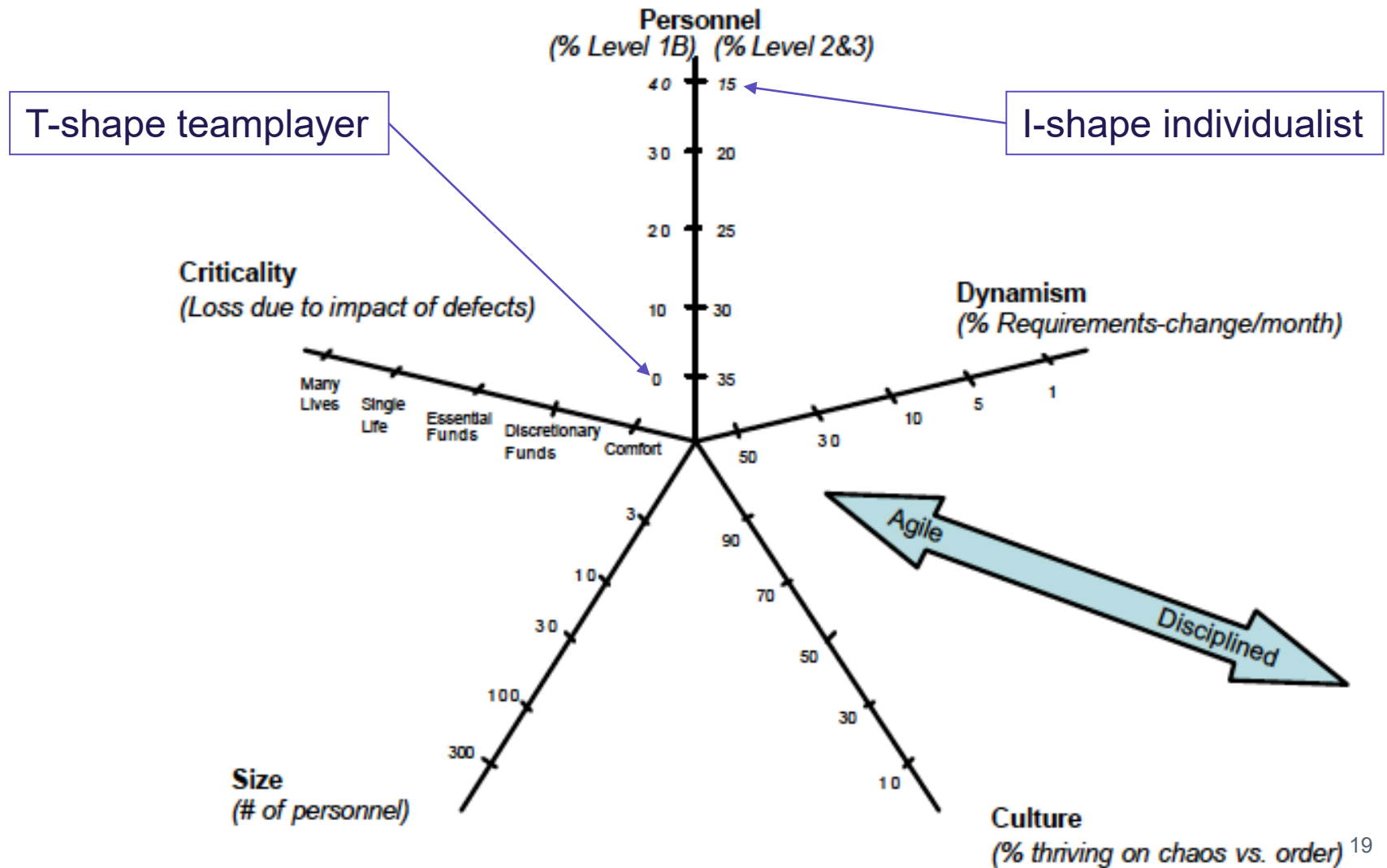
Level -1 people should be rapidly identified and found work to do other than performing on either agile or disciplined teams.

Level 1B people roughly correspond to the “1975-average” developer profile. They can function well in performing straightforward software development in a stable situation. But they are likely to slow down an agile team trying to cope with rapid change, particularly if they form a majority of the team. They can form a well-performing majority of a stable, well-structured disciplined team.

Level 1A people can function well on agile or disciplined teams if there are enough Level 2 people to guide them. When agilists refer to being able to succeed on agile teams with ratios of 5 Level 1 people per Level 2 person, they are generally referring to Level 1A people.

Level 2 people can function well in managing a small, precededented agile or disciplined project but need the guidance of Level 3 people on a large or unprecedented project. Some Level 2s have the capability to become Level 3s with experience. Some do not.

# Home Grounds decision tool



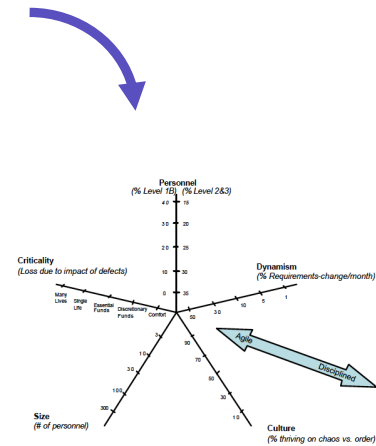
# Group exercises (September 24<sup>th</sup> and October 5<sup>th</sup>)

**Exercise 1: Compare and contrast XP and Scrum (20%)**

**Exercise 2: Estimation (40%)**

**Exercise 3: Analyze home grounds for your project (40%)**

Characteristics	Agile	Plan-driven
<b>Application</b>		
Primary Goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable; low change; project/organization focused
<b>Management</b>		
Customer Relations	Dedicated on-site customers; focused on prioritized increments	As-needed customer interactions; focused on contract provisions
Planning and Control	Internalized plans; qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
<b>Technical</b>		
Requirements	Prioritized informal stories and test cases; undergoing unforeseeable change	Formalized project, capability, interface, quality, foreseeable evolution requirements
Development	Simple design; short increments; refactoring assumed inexpensive	Extensive design; longer increments; refactoring assumed expensive
Test	Executable test cases define requirements, testing	Documented test plans and procedures
<b>Personnel</b>		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers, not always collocated
Developers	At least 30 percent full-time Cockburn Level 2 and 3 experts; no Level 1B or -1 personnel**	50 percent Cockburn Level 3s early; 10 percent throughout; 30 percent Level 1Bs workable; no Level -1s**
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)
* Collaborative, Representative, Authorized, Committed, Knowledgeable ** See Table 2. These numbers will particularly vary with the complexity of the application		



**Home ground  
Decision tool**

**Home ground**

# Lecture objectives

An overview of eXtreme Programming (XP)

Basic skills in estimation of software development tasks

Prepared to choose and adapt a development method for a specific project

**Suggestions to improve the following lectures? → [john@cs.aau.dk](mailto:john@cs.aau.dk)**