



# eXtreme Programming (XP)

Andrea Polini

Software Project Management  
MSc in Computer Science  
University of Camerino

# Changes and SW development

Among the agile principles:

- **Welcome changing requirements**, even late in development. Agile processes harness change for the **customer's competitive advantage**

Easy to say, but ...

## Developers hate changes

They know that making changes can cause the introduction of bugs. The more changes you make the more **brittle the codebase gets**. Imagine what can happen with **frequent changes**.

# Changes and SW development

Among the agile principles:

- **Welcome changing requirements**, even late in development. Agile processes harness change for the **customer's competitive advantage**

Easy to say, but ...

## Developers hate changes

They know that making changes can cause the **introduction of bugs**. The more changes you make the more **brittle the codebase gets**. Imagine what can happen with **frequent changes**.

# Supporting changes in XP

## XP practices

To support changes XP proposes the adoption of a set of practices for SW development, that are organized in 4 disjoint groups:

- Programming
- Integration
- Planning
- Team

# Programming Practices

## Test first programming

- Tests are written **before the code**
- Code is **built to pass the tests**
- **Automated testing frameworks** are adopted (JUnit)
- Tests are run **every time** the code is built
- Tests are **part of the codebase**

## Pair programming

- Two programmers sits together at the workstation
- One of the programmer writes the code, and the other observes. Nevertheless **they constantly discuss what to write.**
- Effects:
  - reduced risks of **shortcuts**
  - **more innovative code**
  - continuous review of code

# Programming Practices

## Test first programming

- Tests are written **before the code**
- Code is **built to pass the tests**
- **Automated testing frameworks** are adopted (JUnit)
- Tests are run **every time** the code is built
- Tests are **part of the codebase**

## Pair programming

- Two programmers sits together at the workstation
- One of the programmer writes the code, and the other observes. Nevertheless **they constantly discuss what to write.**
- Effects:
  - reduced risks of **shortcuts**
  - **more innovative code**
  - continuous review of code

# Programming Practices

## Incremental design

- cost of large-scale design changes rise dramatically over time
- most economical cost design strategy is to **make big design decisions early** and defer all small-scale decisions until later

XP teams are confident in their **ability to adapt the design** to future requirements. The advice to XP teams is not to minimize design investment over the short run, but to **keep the design investment in proportion to the needs of the system** so far. Incremental design suggests that **the most effective time to design is in the light of experience**

Eliminate duplication. If you have the same login in two places you must work with design to understand how you can have only one copy.

# Integration Practices

## 10-minute build

- under 10 minutes build for the whole codebase
  - The build includes all the unit tests and generates a **report with pass/fail results**

## continuous integration

- use a server to permit to people to share the codebase and to work in parallel
- use check-out to create local copies and make commits for working copies
- check-out often and run tests before committing
- a **build token** can be used to pass the right to integrate and reduce the risks known as “integration hell”



# Planning Practices

Only software in *status done done* is delivered

## Weekly cycle

- **one-week iterations**
  - start with a planning meeting where **together with the customers** the team selects the stories for the iteration, and **split them in tasks**
  - each developer then select a story and develop tests for the story and tasks
  - than the developer write the code

## Stories

- **Stories** are the main tool to **identify needs** and to consequently drive the work

# Planning Practices

## Quarterly cycle

XP teams use quarterly cycle practice to do **long-term planning**.

- the team **discuss themes** to put together stories and to identify missing ones
- the team **reflects on the progress made** and on how the project is going overall

## Slack

- Add **minor lower-priority stories** to each weekly cycle

# Team Practices

## Sit together

Face to face interactions are fundamental for the health of the project. **Programming is a highly social activity.** Organize the workspace to suite private reflection and joint work.

## Informative workspace

- team working environment is set up **to automatically communicate important project information to anyone** in the project
- **information radiators** and **osmotic communication**

# Team Practices

## Energized work

- establish an environment where every team member is given **enough time and freedom to do the job**
- **avoid continuous distraction, and avoid unjustified pressure**
- work only as many hours as you can be productive
- work only as many hours as you can sustain

## Whole team

- All the contributors to an XP project sit together, **members of one team**
- Everyone on an XP team contributes in **any way that they can**. The best teams have **no specialists**, only general contributors with special skills.

# XP values

## XP additional values

- **Communication:** each team member is aware of the work everyone else is doing
- **Simplicity:** developers focus on writing the most simple and direct solutions possible
- **Feedback:** constant tests and feedback loops keep the quality of the product in check
- **Courage:** each team member is focused on making the best choice for the project, even if it means having to discard failing solutions or approach things differently
- **Respect:** each team member is important and valuable for the project

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity:** projects are built by people - balance project and people needs
- **Economics:** somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit:** practices that benefit individual, team, customer together
- **Self similarity:** when you find a practice that works stick with it at different levels
- **Improvement:** do your best today and try to improve for tomorrow
- **Diversity:** different opinions and perspectives lead to better solutions
- **Reflection:** good teams stay aware of what's working and what isn't in their proc.
- **Flow:** constant delivery means a continuous flow of development work
- **Opportunity:** problems are opportunities to learn something new
- **Redundancy:** it can seem wasteful but often can avoid big quality problems
- **Failure:** It's OK to try things that don't work. You can learn from failing
- **Quality:** You can't deliver faster by accepting a lower quality product
- **Accepted responsibility:** resp. of something means authority to get it done
- **Baby steps:** take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- Self similarity: when you find a practice that works stick with it at different levels
- Improvement: do your best today and try to improve for tomorrow
- Diversity: different opinions and perspectives lead to better solutions
- Reflection: good teams stay aware of what's working and what isn't in their proc.
- Flow: constant delivery means a continuous flow of development work
- Opportunity: problems are opportunities to learn something new
- Redundancy: it can seem wasteful but often can avoid big quality problems
- Failure: It's OK to try things that don't work. You can learn from failing
- Quality: You can't deliver faster by accepting a lower quality product
- Accepted responsibility: resp. of something means authority to get it done
- Baby steps: take small steps in the right direction rather than making big changes when adopting new practices



# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices



# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Embracing changes

Gap between the practices and the values. Principles help to guide you in applying the practices. Still to permit to embrace changes.

- **Humanity**: projects are built by people - balance project and people needs
- **Economics**: somebody is paying - everybody has to keep the budget in mind
- **Mutual Benefit**: practices that benefit individual, team, customer together
- **Self similarity**: when you find a practice that works stick with it at different levels
- **Improvement**: do your best today and try to improve for tomorrow
- **Diversity**: different opinions and perspectives lead to better solutions
- **Reflection**: good teams stay aware of what's working and what isn't in their proc.
- **Flow**: constant delivery means a continuous flow of development work
- **Opportunity**: problems are opportunities to learn something new
- **Redundancy**: it can seem wasteful but often can avoid big quality problems
- **Failure**: It's OK to try things that don't work. You can learn from failing
- **Quality**: You can't deliver faster by accepting a lower quality product
- **Accepted responsibility**: resp. of something means authority to get it done
- **Baby steps**: take small steps in the right direction rather than making big changes when adopting new practices

# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate

# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate

# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate

# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate

# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate



# Notes from the principles

- XP teams reject separation of roles, this reduces the capability of doing the maximum, the opportunities for improving and learning, and finally it reduces the possibility to get help
- Pairs generally rotate to foster diversity and learning
- Pair programming generally enable reflection and feedbacks
- teams write hours they think are needed to implement a story
- the use of stories fits with the principles
- you accept critics and are not afraid of making critics to your teammate

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- Incremental deployment: deploy small pieces of the system individually than one “big shot”
- Team continuity: keep effective teams together
- Shrinking teams: use people from teams to spread competences and XP culture
- route-cause analysis: figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- Shared code: everyone collectively owns the code
- Code and tests: write code and test and generate the rest (people do not read dusty binders)
- Single codebase: don't manage multiple versions
- Daily deployment: push a new version of the software into production every day
- Negotiated scope contract: fix the time and have an ongoing negotiation of the scope, instead than viceversa
- Pay-per-use: charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- Team continuity: keep effective teams together
- Shrinking teams: use people from teams to spread competences and XP culture
- route-cause analysis: figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- Shared code: everyone collectively owns the code
- Code and tests: write code and test and generate the rest (people do not read dusty binders)
- Single codebase: don't manage multiple versions
- Daily deployment: push a new version of the software into production every day
- Negotiated scope contract: fix the time and have an ongoing negotiation of the scope, instead than viceversa
- Pay-per-use: charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles



# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Corollary practices

- **Real customer involvement:** involve customers in planning and actually listen to them
- **Incremental deployment:** deploy small pieces of the system individually than one “big shot”
- **Team continuity:** keep effective teams together
- **Shrinking teams:** use people from teams to spread competences and XP culture
- **route-cause analysis:** figure out a problem if something went wrong, and why the problem occurred, and what caused the occurrence
- **Shared code:** everyone collectively owns the code
- **Code and tests:** write code and test and generate the rest (people do not read dusty binders)
- **Single codebase:** don't manage multiple versions
- **Daily deployment:** push a new version of the software into production every day
- **Negotiated scope contract:** fix the time and have an ongoing negotiation of the scope, instead than viceversa
- **Pay-per-use:** charge not for the development but for the usage. Improve feedbacks and relevance of functionality

Practices are natural when the mindset absorbed the values and the principles

# Simplicity and Incremental design

Build software that can be extended and changed easily. Clever solutions can be dangerous. **Prefer simplicity!**

- Rookies are not more “dangerous” than experienced developers
- Complexity often comes from **guessing future needs**.

## code smells/antipatterns

there are recurring situations in coding that indicate that you are “violating” some properties

# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- half-baked code – strong hard-coded dependencies
- very large classes
- duplicated code
- spaghetti code
- lasagna code

# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- **half-baked code** – strong hard-coded dependencies
- very large classes
- duplicated code
- spaghetti code
- lasagna code



# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- **half-baked code** – strong hard-coded dependencies
- **very large classes**
- duplicated code
- spaghetti code
- lasagna code

# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- **half-baked code** – strong hard-coded dependencies
- **very large classes**
- **duplicated code**
- spaghetti code
- lasagna code

# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- **half-baked code** – strong hard-coded dependencies
- **very large classes**
- **duplicated code**
- **spaghetti code**
- **lasagna code**

# Typical Code smells

They can be statically or dynamically identified

- **shotgun surgery** – simple changes lead to cascading changes
- **half-baked code** – strong hard-coded dependencies
- **very large classes**
- **duplicated code**
- **spaghetti code**
- **lasagna code**

# Problems with “cleverness”

Typical problems in the code:

- **hooks** – use of placeholders
- **edge case** – obsession for rare and exceptional scenarios
- **framework trap**

# The framework trap

Generally developers like to **overgeneralize**. A simple problem often can lead to a framework that intend to solve the same problem in many different contexts

- e.g. need for a web page overgeneralized in a framework to define pages

## Libraries vs Frameworks

# The framework trap

Generally developers like to **overgeneralize**. A simple problem often can lead to a framework that intend to solve the same problem in many different contexts

- e.g. need for a web page overgeneralized in a framework to define pages

## Libraries vs Frameworks

# Technical debt

When you release software with poor design and code you are taking a debt (**technical debt**)

- An effective XP team fix technical debt by refactoring mercilessly
- slack are good candidates for repaying the debt
- reflection and refactoring are needed to capture smells
- **fail fast** strategy reduce risks of accumulating debts

**Continuous integration** – at least once per day the whole codebase should be integrated



# TDD

Now we'll walk through a custom List implementation using the Test-Driven Development (TDD) process.

## ATTENTION

TDD is a design tool, enabling us to drive our implementation with the help of tests

```
public class CustomList<E> implements List<E> {  
    private Object[] internal = {};  
    // empty implementation methods  
}
```

# First Cycle

```
@Test
public void givenEmptyList_TrueIsReturned() {
    List<Object> list = new CustomList<>();

    assertTrue(list.isEmpty());
}
```

Given the test which is a simple correct implementation?

```
@Override
public boolean isEmpty() {
    return true;
}
```

# First Cycle

```
@Test
public void givenEmptyList_TrueIsReturned() {
    List<Object> list = new CustomList<>();

    assertTrue(list.isEmpty());
}
```

Given the test which is a simple correct implementation?

```
@Override
public boolean isEmpty() {
    return true;
}
```

## 2nd Cycle

```
@Test
public void givenNonEmptyList_thenFalseIsReturned() {
    List<Object> list = new CustomList<>();
    list.add(null);

    assertFalse(list.isEmpty());
}

@Override
public boolean isEmpty() {
    if (internal.length != 0) {
        return false;
    } else {
        return true;
    }
}
```

## 2nd Cycle

```
@Test
public void givenNonEmptyList_thenFalseIsReturned() {
    List<Object> list = new CustomList<>();
    list.add(null);

    assertFalse(list.isEmpty());
}

@Override
public boolean isEmpty() {
    if (internal.length != 0) {
        return false;
    } else {
        return true;
    }
}
```

# Slack and Refactoring

```
@Override
public boolean isEmpty() {
    return internal.length == 0;
}
```

# FAQs

- Writing tests is an activity generally left to QA teams and to lower profiles. Is it worthy to engage senior and skilled programmers in such an activity?
- I'm a programmer and how do I know what to do next?
- Isn't better to assign tasks on the base of expertise?