HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

h_da

Agile Software Development

Part 6: Extreme Programming (XP)

Prof. Dr. A. del Pino

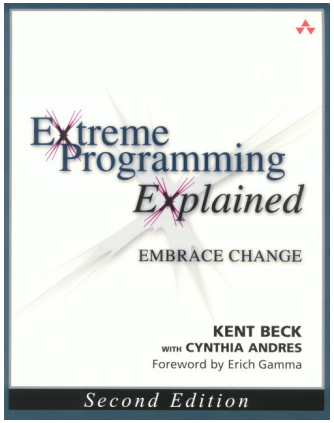
Extreme Programming (XP)

Successful software development teams

Kent Beck starts with the question what makes a software development team successful, and summarizes:

> Good teams are more alike than they are different.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

- Hence, by observing and analyzing successful software development teams, it should be possible to identify general characteristics and patterns, and to apply these findings to less successful teams for their improvement.
- Literature recommendation:



K. Beck. *Extreme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

What is XP ?

Kent Beck:

> Extreme Programming (XP) is about social change. [...]
>
> Productivity and confidence are related to our human relationships in the workspace as well as to our coding or other work activities. You need both technique and good relationships to be successful. XP addresses them both.
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

and

> Software development is a human process not a factory.
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005, pg. 89

Extreme Programming (XP)

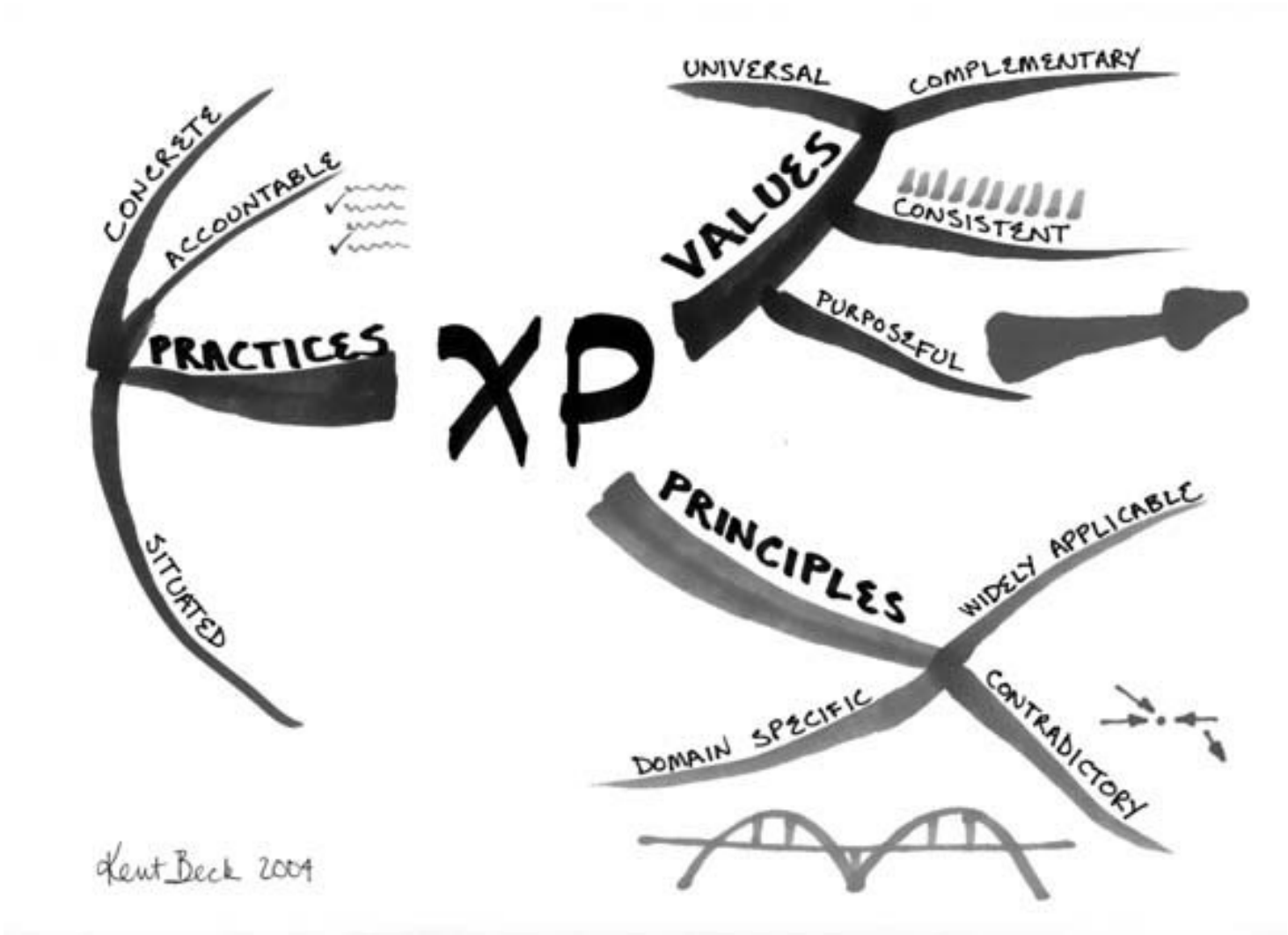


Image Source: http://www.threeriversinstitute.org/overview.jpg

Extreme Programming (XP)

Values

The biggest problem I encounter in what people „just know“ about software development ist that they are focused on individual action. What actually matters is not how any given person behaves as much as how the individuals behave as part of the team, and as part of an organization.

Source: K. Beck: „Exteme Programming Explained. Embrace Change“, Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Communication

> What matters most in team software development is communication. When probles arise in development, most often someone already knows the solution; [...] Communication is important for creating a sense of team and effective cooperation.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Compare this with a principle from the agile manifesto:

> The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
>
> Source: Agile manifesto

Factors to facilitate communication ?

Extreme Programming (XP)

Simplicity

> To make a system simple enough to gracefully solve today's problem is hard work. Yesterday's simple solution may be fine today, or it may look simplistic or complex. [...]
>
> I ask people to think about the question, „What is the simplest thing that could possibly work?“ [...]
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change“, Addison-Wesley, 2nd edition, 2005

Compare this with a principle from the agile manifesto:

> Simplicity – the art of maximizing the amount of work not done – is essential.
>
> Source: Agile manifesto

Extreme Programming (XP)

Feedback

Change is inevitable, but change creates the need for feedback. [...] Being satisfied with improvement rather than expecting instant perfection, we use feedback to get closer and closer to our goals. [...]

XP teams strive to generate as much feedback as they can handle as quickly as possible. They try to shorten the feedback cycle to minutes or hours instead of weeks or months. The sooner you know, the sooner you can adapt.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Feedback

> Feedback also contributes to simplicity. Which of three solutions will turn out to be simplest? Try all three and see. While implementing the same thing three times may seem wasteful, it may be the most efficient way to arrive at a solution whose simplicity you can live with.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Compare this with the lean tool *set based development*:

> When you have a difficult problem, try this: Develop a set of alternative solutions to a problem, see how well they actually work, and then merge the best features of the solutions or choose one of the alternatives.
>
> Source: M. Poppendieck, T. Poppendieck: *Lean Software Development – An Agile Toolkit*, Addison-Wesley, 2003

Extreme Programming (XP)

Courage

> Courage is effective action in the face of fear. [...]
>
> Courage as a primary value without counterbalancing values is dangerous. [...] in concert with the other values it is powerful. The courage to speak truths, pleasant or unpleasant, fosters communication and trust. The courage to discard failing solutions and seek new ones encourages simplicity. The courage to seek real, concrete answers creates feedback.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Your experience?

Extreme Programming (XP)

Respect

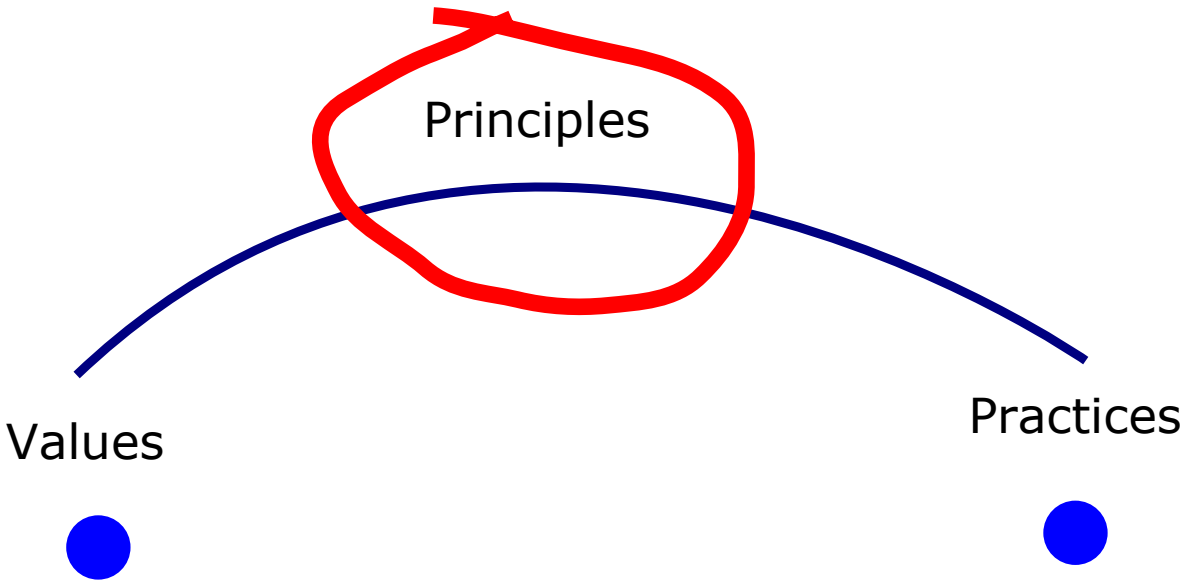
The previous four values point to one that lies below the surface of the other four: respect. If members of a team don't care about each other and what they are doing, XP won't work. If members of a team don't care about a project, nothing can save it. [...]

For software development to simultaneously improve in humanity and productivity, the contributions of each person on the team need to be respected. I am important and so are you.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Your experience?

Extreme Programming (XP)



Understanding the principles gives you the opportunity to create practices that work in harmony with your existing practices and your overall goals.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Humanity

> People develop software. This simple, inescapable fact invalidates most of the available methodological advice. Often, software development doesn't meet human needs, acknowledge human frailty, and leverage human strength. Acting like software isn't written by people exacts a high cost on participants [...]
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

- Your experience? Examples ?
- Details of the work environment, e.g. cubicles. See also *Tom DeMarco*, Peopleware.

Extreme Programming (XP)

Economics

> Someone has to pays for all this. [...]
>
> Software development is more valuable when it earns money sooner and spends money later. Incremental design explicitly defers design investment until the last responsible moment in an effort to spend money later. Pay-per-use provides a way of realizing revenue from features as soon as they are deployed. [...]
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Compare this with the *agile manifesto* and the *lean approach*: Which principles are addressed ?

Extreme Programming (XP)

Mutual Benefit

> Every activity should benefit all concerned. Mutual benefit is the most important XP principle and the most difficult to adhere to. There are always solutions to any problem that cost one person while benefitting another. When the situation is desperate, these solutions seem attractive. [...]
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Why is this principle so important ?

Extreme Programming (XP)

Self-Similarity

> When nature finds a shape that works, she uses it everywhere she can. The same principle applies to software development: try copying the structure of one solution into a new context, even at different scales.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Examples ?

Extreme Programming (XP)

Improvement

> In software development, „perfect“ is a verb, not an adjective. There is no perfect process. There is no perfect design. There are no perfect stories. You can, however, perfect you process, your design, and your stories. [...] Put improvement to work by not waiting for perfection.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change“, Addison-Wesley, 2nd edition, 2005

Compare also with the adaptive software development approach:

> Deviations guide us toward the correct solution.
>
> Source: J. Highsmith. *Adaptive Software Development*. Dorset House Publishing, 2000, pg. 43

Similarities and differences to continuous improvement (PDCA) a la Deming and Shewhart ?

Extreme Programming (XP)

Diversity

> Software development teams where everyone in alike, while comfortable, are not effective. Teams need to bring together a variety of skills, attitudes, and perspectives to see problems and pitfalls, to think of multiple ways to solve problems, and to implement the solutions. Teams need diversity.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Compare this with diversity as a property of CAS:

> The persistence of any individual agent [...] depends on the context provided by the other agents. Roughly, each kind of agent fills a niche that is defined by the interactions centering on that agent.
>
> Source: John H. Holland. *Hidden Order. How Adaptation Builds Complexity*. Basic Books, 1995, pg. 27

Extreme Programming (XP)

Reflection

> Good teams don't just do their work, they think about how they are working and why they are working. They analyze why they succeeded or failed. They don't try to hide their mistakes but expose them and learn from them. No one stumbles into excellence. [...]
>
> Reflection isn't a purely intellectual exercise. You can gain insight by analyzing data, but you can also learn from your gut. [...]

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Your experience ?

Extreme Programming (XP)

Flow

> Flow in software engineering is delivering a steady flow of valuable software by engaging in all the activities of development simultaneously. The practices of XP are biased towards a continuous flow of activities rather than discrete phases. [...] The daily build, for example, is flow-oriented. However, daily builds are a small step on the road to flow. [...]
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

Compare with the flows in CAS:

> In general terms, the nodes are processors – agents – and the connectors designate the possible interactions.
>
> Source: John H. Holland. *Hidden Order. How Adaptation Builds Complexity*. Basic Books, 1995, pg. 23

Extreme Programming (XP)

Flow

> Flow in software engineering is delivering a steady flow of valuable software by engaging in all the activities of development simultaneously. [...]
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

Compare this with the adaptive development approach:

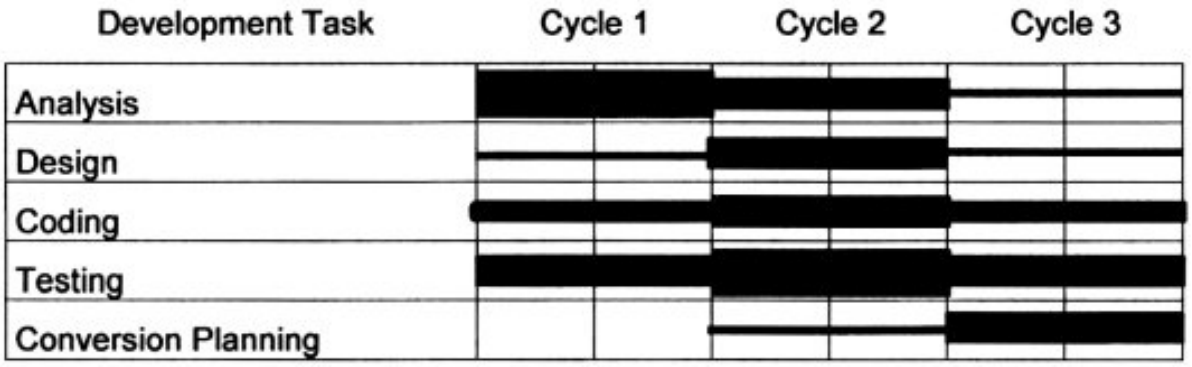


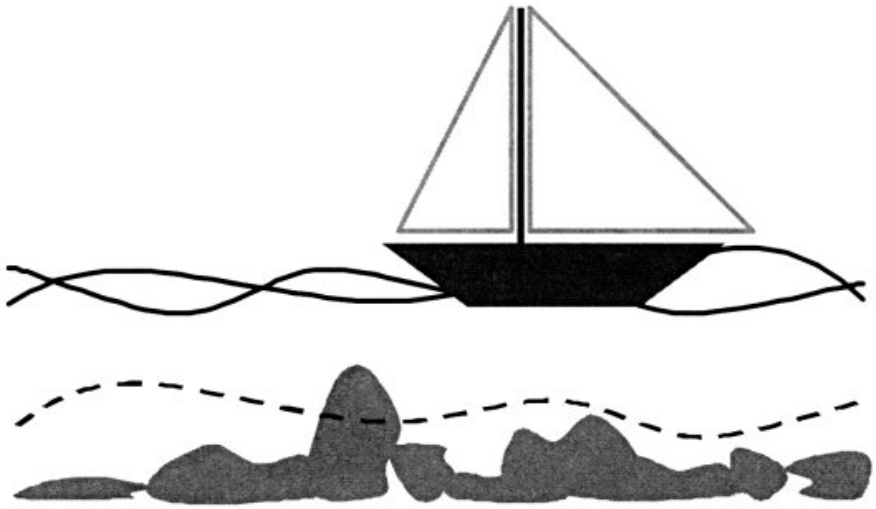
Image source: J. Highsmith. *Adaptive Software Development*. Dorset House Publishing, 2000, pg. 100

Extreme Programming (XP)

Flow

I visited a team that used to deploy every week. It had more and more problems, until it was taking six days to deploy a week's worth of software. The team choose to deploy every two weeks. This amplified their integration and deployment problems.

Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005



Compare this with the 2nd of the 14 *Toyota Way* principles:

Create continuous process flow to bring problems to the surface.

Source: J. K. Liker. *The Toyota Way*. McGraw-Hill, 2004, pg. 37 ff.

Image Source: Mary and Tom Poppendieck. *Implementing Lean Software Development*. Addison-Wesley, 2007, pg. 7

Extreme Programming (XP)

What causes the flow ?

> Work is pulled through the system based on actual demand, not pushed through the system based on projected demand.
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005, pg. 86

Compare this with the lean *pull* principle:

> [T]he ability to design, schedule, and make exactly what the customer wants just when the customer wants it means you can throw away the sales forecast and simply make what customers actually tell you they need. That is, you can let the customer *pull* the product from you as needed rather than pushing products, often unwanted, onto the customer.
>
> Source: J. P. Womack, D. T. Jones. *Lean Thinking*. Free Press, 2003 (originally published in 1996), pg. 24

Extreme Programming (XP)

Opportunity

To reach excellence, problems need to turn into opportunities for learning and improvement, not just survival. [...]

Turning problems into opportunities takes place across the development process. [...] Can't make accurate long-term plans? Fine – have a quarterly cycle during which you refine your long-term plans. A person makes too many mistakes? Fine – program in pairs.

Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

Your experience ?

Extreme Programming (XP)

Redundancy

The critical, difficult problems in software development should be solved several different ways. Even if one solution fails utterly, the other solutions will prevent disaster. [...]

Defects are addressed in XP by many of the practices: pair programming, continuous integration, sitting together, real customer involvement, and daily deployment, for example.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Failure

> If you're having trouble succeeding, fail. Don't know which of three ways to implement a story? Try it all three ways. Even if they all fail, you'll certainly learn something valuable. Isn't failure waste? No, not if it imparts knowledge. Knowledge is valuable and sometimes hard to come by.
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Two aspects of knowledge (Takeuchi and Nonaka, 1995)

- Explicit knowledge – Symbols such as words and numbers, which can easily be shared, communicated, and processed.
- Tacit knowledge - Intuitive knowledge, must first be converted into such symbols to be communicated with others.

Extreme Programming (XP)

Learning from failure – Tacit vs. explicit knowledge

A young child screams with pain upon touching a hot stove. A little comfort and mild medication soon make things well, except for a small blister. That evening the parent, returning home, greets the child as usual: "Hi – and what did you learn today?" "Nothing," comes the cheerful response. But never again will the child touch the burner, except cautiously, even when the stove is cold.

Levitt, 1991, p. 17, cit. in: I. Nonaka, H. Takeuchi. *The Knowledge-Creating Company*. Oxford University Press, 1995, pg. 9

Other examples from your experience in the context of software development ?

Extreme Programming (XP)

Quality

Quality is not a control variable. Projects don't go faster by accepting lower quality. They don't go slower by demanding higher quality. [...]

Quality isn't a purely economic factor. People need to do work they are proud of. I remember talking to a manager of a mediocre team. He went home on the weekends and made fancy ironwork as a blacksmith. He met his need for quality; he just met it outside of work. [...]

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Quality

> We all tend to tie our self-esteem strongly to the quality of the product we produce – not the quantity of product, but the quality. (For some reason, there is little satisfaction in turning out huge amounts of mediocre stuff, although that may be just what's required for a given situation.)

Source: T. DeMarco, T. Lister. *Peopleware*. Dorset House Publishing, 2nd edition, 1999

Extreme Programming (XP)

Baby Steps

> It's always tempting to make big changes in big steps. After all, there's a long way to go and a short time to get there. [...]
>
> Baby steps acknowledge that the overhead of small steps is much less than when a team wastefully recoils from aborted big changes.

Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005

Compare this with adaptive software development:

> High speed is easy; high change is the real challenge.

Source: J. Highsmith. *Adaptive Software Development*. Dorset House Publishing, 2000, pg. 19

Extreme Programming (XP)

Baby Steps – The people side of it

> Momentous change taken all at once is dangerous. It is people who are being asked to change. Change is unsettling. People only change so fast.
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005, pg. 33

In other words:

> Here is something to repeat to yourself whenever you set out to ask people to change:
>
> **MANTRA: The fundamental response to change is not logical, but emotional.**
>
> Source: T. DeMarco, T. Lister. *Peopleware*. Dorset House Publishing, 2nd edition, 1999, pg. 197

Your experience ?

Extreme Programming (XP)

Accepted Responsibility

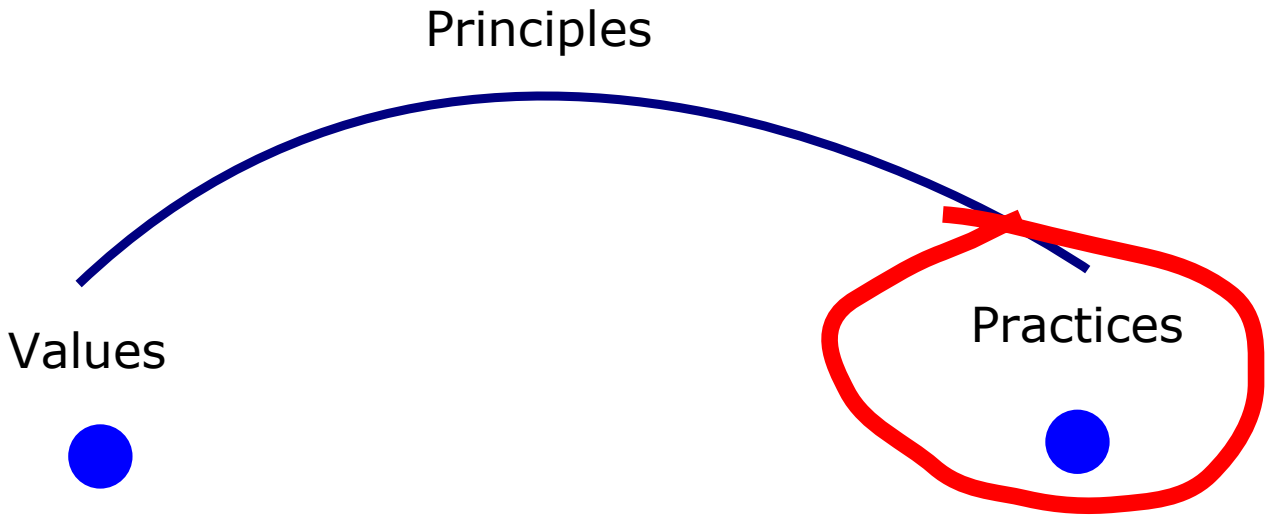
> Responsibility cannot be assigned; it can only be accepted. If someone tries to give you responsibility, only you can decide if you are responsible or if you aren't.
>
> Source: K. Beck. *Exteme Programming Explained. Embrace Change*. Addison-Wesley, 2nd edition, 2005, pg. 34

Extending this to a whole development team:

> On software projects, shared responsibility means that everyone owns the result. [...] It is built on personal responsibility.
>
> Source: J. Highsmith. *Adaptive Software Development*. Dorset House Publishing, 2000, pg. 132

What does this mean in the context of being responsible to implement a story ?

Extreme Programming (XP)



[T]he kind of things you'll see XP teams doing day-to-day.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Practices

Kent Beck:

Practices are situation dependent. If the situation changes, you choose different practices to meet those conditions. Your values do not have to change in order to adapt to a new situation. [...]

The practices are a vector from where you are to where you can be with XP. In XP, you make progress this ideal state of effective development. [...]

The primary practices are useful independent of what else you are doing. They each can give you immediate improvement. You can start safely with any of them. The corollary practices are likely to be difficult without first mastering the primary practices.

The amplification effect of using the practices together means there is an advantage to adding practices as quickly as you can.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Sit Together

Kent Beck:

> *Develop in an open space big enough for the whole team. Meet the need for privacy and „owned" space by having small private spaces nearby [...]*
>
> *Tearing down the cubicle walls before the team is ready is counter-productive. If the team members' sense of safety is tied to having their own little space, removing that sense of safety before replacing it with the safety of shared accomplishment is likely to produce resentment and resistance. [...]*
>
> *„Sit Together" predicts that the more face time you have, the more humane and productive the project. If you have a multi-site project and everything is going well, keep doing what you're doing. If you have problems, think about ways to sit together more, even if it means traveling.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Whole Team

Kent Beck:

> *Include on the team people with all the skills and perspectives necessary for the project to succeed. This is really nothing more than the old idea of cross-functional teams. [...]*
>
> *What constitutes a „whole team" is dynamic. If a set of skills or attributes becomes important, bring a person with these skills on the team. If someone is no longer necessary, he can go elsewhere. [...]*
>
> *An issue that often arises is ideal team size. [...] For larger projects, finding ways to fracture the problem so it can be solved by a team of teams allows XP to scale up. [...]*
>
> *Some organizations try to have teams with fractional people: „You'll spend 40% of your time working for these customers and 60% work for those customers." In this case, so much time is wasted on task-switching that you can see immediate improvement by grouping the programmers into teams.*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Informative Workspace

Kent Beck:

> *Make your workspace about your work. An interested observer should be able to walk into the team space and get a general idea of how the project is going in fifteen seconds. [...]*
>
> *Many teams implement this practice in part by putting story cards on a wall. Sorting the cards spatially conveys information quickly. [...]*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

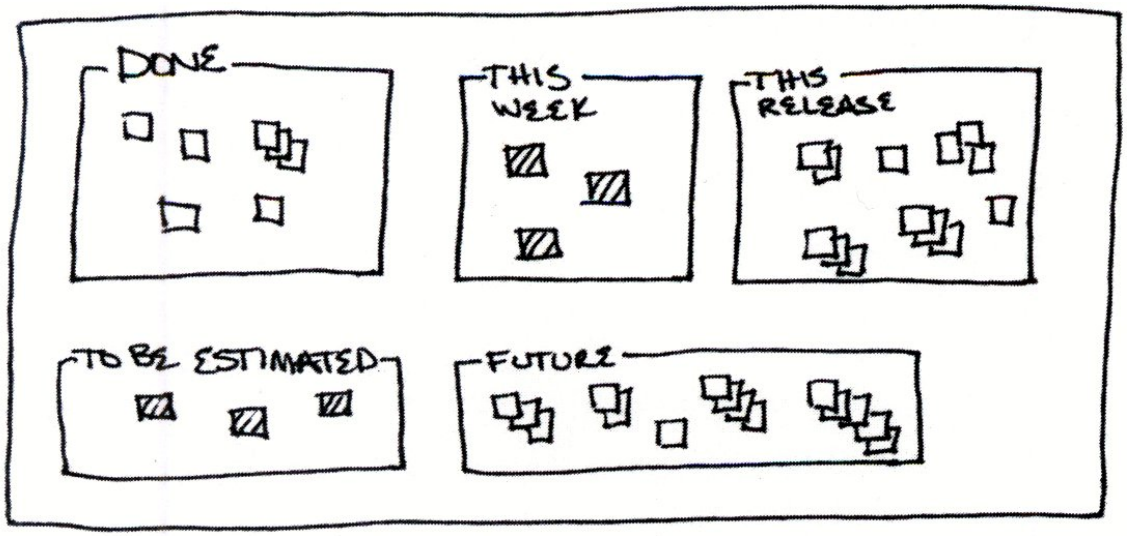


Image: K. Beck: „Extreme Programming Explained. Embrace Change"

Extreme Programming (XP)

Energized Work

Work only as many hours as you can be productive and only as many hours as you can sustain. Burning yourself out unproductively today and spoiling the next two days' work isn't good for you or the team. [...]

Software development is a game of insight, and insight comes to the prepared, rested, relaxed mind. [...]

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Compare also with *Tom DeMarco „Peopleware", Chapter 10.*

Extreme Programming (XP)

Pair Programming

Write all production programs with two people sitting at one machine. Set up the machine so the partners can sit comfortably side-by-side. Move the keyboard and mouse back and forth so you are comfortable while you are typing. Pair programming is a dialog between two people simultaneously programming (and analyzing and designing and testing) and trying to program better. [...]

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Pair Programming ...

Kent Beck:

> *Pairing doesn't mean that you can't think alone. [...] You can even prototype alone and still respect pairing. [...] When you're done exploring, bring the resulting idea, not the code, back to the team. With a partner you'll reimplement it quickly. [...]*
>
> *Pair programming is tiring but satisfying. Most programmers can't pair for more than five to six hours in a day. [...]*
>
> *Rotate pairs frequently. Some teams report good results obeying a timer that tells them to shift partners every sixty minutes (every thirty minutes when solving difficult problems). [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Pair Programming ...

Kent Beck:

> *An issue that has come up and requires comment is the close contact in pair programming. Different individuals and cultures are comfortable with different amounts of body space. [...] Personal space must be respected for both parties to work well.*
>
> *Personal hygience and health are important issues when pairing. Cover your mouth when you cough. Don't come to work when you are sick. Avoid strong colognes that might affect your partner. [...]*
>
> *If you are uncomfortable pairing with someone on the team, talk about it with someone safe; a respected team member, a manager, or someone in human resources. If you aren't comfortable, the team isn't doing as well as it could. And chances are others are uncomfortable too.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Stories

Plan using units of customer-visible functionality. „Handle five times the traffic with the same response time". „Provide a two-click way for users to dial frequently used numbers." As soon as a story is written, try to estimate the development effort necessary to implement it.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Stories ...

Kent Beck:

> *Early estimation is a key difference between stories and other requirements practices. Estimation gives the business and technical perspectives a chance to interact, which creates value early, when an idea has the most potential. [...]*
>
> *Give stories short names in addition to a short prose or graphical description. Write the stories on index cards and put them on a frequently-passed wall. [...]*
>
> *Every attempt I've seen to computerize stories has failed to provide a fraction of the value of having real cards on a real wall. [...]*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Weekly Cycle

Kent Beck:

> *Plan work a week at a time. Have a meeting at the beginning of every week. During this meeting:*
>
> 1. *Review progress to date, including how actual progress for the previous week matched expected progress.*
> 2. *Have the customers pick a week's worth of stories to implement this week.*
> 3. *Break the stories into tasks. Team members sign up for tasks and estimate them.*
>
> *Start the week by writing automated tests that will run when the stories are completed. Then spend the rest of the week completing the stories and getting the tests to pass.*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Weekly Cycle ...

Kent Beck:

> *Planning is a form of necessary waste. It doesn't create much value all by itself. Work on gradually reducing the percentage of time you spend planning. Some teams start with a whole day of planning for a week, but gradually refine their planning skills until they spend an hour planning for the week. [...]*
>
> *Some people like to start their week on a Thursday or Wednesday. [...] shifting the start of the cycle makes some sense as long as it doesn't put pressure on people to work over the weekend. Working weekends is not sustainable; if the real problem is that the estimates are overly optimistic, then work on improving your estimates. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Quarterly Cycle

Kent Beck:

> *Plan work a quarter at a time. [...] During quarterly planning:*
>
> - *Indentify bottlenecks, especially those controlled outside the team.*
> - *Initiate repairs.*
> - *Plan the theme or themes for the quarter.*
> - *Pick a quarter's worth if stories to address those themes.*
> - *Focus on the big picture, where the project fits within the organization. [...]*
>
> *The separation of „themes" from „stories" is intended to address the tendency of the team to get focused and excited about the details of what they are doing without reflecting on how this week's stories fit into the bigger picture. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Slack

Kent Beck:

> *In any plan, include some minor tasks that can be dropped if you get behind. You can always add more stories later and deliver more than you promised. It is important in an atmosphere of distrust and broken promises to meet your commitments. [...]*
>
> *You can structure slack in many ways. One week in eight could be „Geek Week". Twenty percent of the weekly budget could go to programmer-choosen tasks. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Ten-Minute Build

Kent Beck:

> *Automatically build the whole system and run all of the tests in ten minutes. A build that takes longer than ten minutes will be used much less often, missing the opportunity for feedback. [...]*
>
> *The ten-minute build is an ideal. What do you do on your way to that ideal? The statement of the practice gives thress clues: [...] If your process isn't automated, that's the first place to start. [...]*
>
> *Automated builds are much more valuable than builds requiring manual intervention. As the general stress level rises, manual builds tend to be done less often and less well, resulting in more errors and more stress. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Continuous Integration

Kent Beck:

> *Integrate and test changes after no more than a couple of hours. Team programming isn't a divide and conquer problem. It is a divide, conquer, and integrate problem. The integration step is unpredictable, but can easily take more time than the original programming. The longer you wait to integrate, the more it costs and the more unpredictable the cost becomes. [...]*
>
> *Integrate and build a complete product. If the goal is to burn a CD, burn a CD. If the goal is to deploy a web site, deploy a web site, even if it is to a test environment. Continuous integration should be complete enough that the eventual first deployment of the system is no big deal.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Synchroneous vs. asynchroneous integration: What are the advantages and disadvantages of these models ?

Extreme Programming (XP)

Test-First Programming

Kent Beck:

> *Write a failing automated test before changing any code. Test-first programming addresses may problems at once:*
>
> - *Scope creep – It's easy to get carried away programming and put in code „just in case." [...]*
> - *Coupling and cohesion – If it's hard to write a test, it's a signal that you have a design problem, not a testing problem. [...]*
> - *Trust – It's hard to trust the author of code that doesn't work. [...]*
> - *Rhythm – It's easy to get lost for hours when you are coding. When programming test-first, it's clearer what to do next: either write another test or make the broken test work. Soon this develops into a natural and efficient rhythm – test, code refactor, test, code, refactor. [...]*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Incremental Design

Kent Beck:

> *Invest in the design of the system every day. Strive to make the design of the system an excellent fit for the needs of the system that day. When your understanding of the best possible design leaps forward, work gradually but persistently to bring the design back into alignment with your understanding. [...]*
>
> *XP teams are confident in their ability to adapt the design to future requirements. Because of this, XP teams can meet their human need for immediate and frequent success as well as their economic need to defer investment to the last responsible moment. [...]*
>
> *As a direction for improvement, incremental design doesn't say that designing in advance of experience is horrible. It says that design done close to when it is used is more efficient. [...]*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Primary vs. corollary practices

Kent Beck suggests to implement the corollary practices only after the primary practices have been implemented:

> *If you begin deploying daily, for example, without getting the defect rate down close to zero (with pair programming, continuous integration, and test-first programming); you will have a disaster on your hands.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

In what follows we will explore these eleven corollary practices:

1. Real customer involvement
2. Incremental deployment
3. Team continuity
4. Shrinking teams
5. Root-Cause analysis
6. Shared Code
7. Code and tests
8. Single code base
9. Daily deployment
10. Negociated scope contract
11. Pay-per-use

Extreme Programming (XP)

Real Customer Involvement

Kent Beck:

> *Make people whose lives and business are affected by your system part of the team. [...] The point of customer involvement is to reduce wasted effort by putting the people with the needs in direct contact with the people who can fill those needs. [...]*
>
> *You will get different results with real customers. They are who you are trying to please. No customer at all, or a „proxy" for a real customer, leads to waste as you develop features that aren't used, specify tests that don't reflect real acceptance criteria, [...]*
>
> *The objection I hear to customer involvement is that someone will get exactly the system he wants, but the system won't be suitable for anyone else. It's easier to generalize a successful system than to specialize a system that doesn't solve anyone's problem. [...]*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Incremental Deployment

Kent Beck:

> *When replacing a legacy system, gradually take over its workload beginning very early in the project. [...]*
>
> *Big deployments have a high risk and high human and economic costs. What's the alternative ? Find a little piece of functionality or a limited data set that you can handle right away. Deploy it. You'll have to find a way to run both programs in parallel, splitting and merging files or training some users to use both programs. This scaffolding, technical or social, is the price you pay for insurance. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Team Continuity

Kent Beck:

Keep effective teams together. There is a tendency in large organizations to abstract people to things, plug-compatible programming units. Value in software is created not just by what people know and do but also by their relationships and what they accomplish together. [...]

Keeping gelled teams together doesn't mean that teams are entirely static. I have been astonished at how quickly new members begin contributing to established XP teams. They insist on signing up for tasks the first week and they are independently contributing after a month.

By mostly keeping teams together and yet encouraging a reasonable amount of rotation, the organization gets the benefits of both stable teams and of consistently spread knowledge and experience.

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Shrinking Teams

Kent Beck:

> *As a team grows in capacity, keep its workload constant but gradually reduce its size. This frees people to from more teams. When the team has too few members, merge it with another too-small team. [...] The other strategies I've seen for scaling up to larger workloads, like creating bigger and bigger teams, work so poorly that alternatives are worth considering. [...]*
>
> *Figure out how many stories the customer needs each week. Strive to improve development until some of the team members are idle; then you're ready to shrink the team and continue.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Taiichi Ohno:

> Teamwork combined with other factors can allow a smaller team to win. The same is true in a work environment.
>
> Source: T. Ohno: „Toyota Production System. Beyond Large-Scale Production", Productivity Press, 1988

Extreme Programming (XP)

Root-Cause Analysis

Kent Beck:

> *Every time a defect is found after development, eliminate the defect and its cause. The goal is not just that this defect won't ever recur, but that the team will never make the same kind of mistake again.*
>
> *In XP, this is the process for responding to a defect:*
>
> 1. *Write an automated system-level test that demonstates the defect, including the desired behavior. [...]*
> 2. *Write a unit test with the smallest possible scope that also reproduces the defect.*
> 3. *Fix the system so the unit test works. This could cause the system test to pass also. If not, return to 2.*
> 4. *Once the defect is resolved, figure out why the defect was created and wasn't caught. Initiate the necessary changes th prevent this kind of defect in the future.*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Root-Cause Analysis ...

Kent Beck:

> *After Five Whys, you find the people problem lying at the heart of the defect (and it's almost always a people problem). Addressing that problem and the other problems encountered along the way will give you some reassurance that you won't ever have to deal with this particular mistake again.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Root-Cause Analysis

Taiichi Ohno:

> WHEN CONFRONTED WITH a problem, have you ever stopped and asked why five times? It is difficult to do even though it sounds easy. For example, suppose a machine stopped functioning:
>
> 1. Why did the machine stop?
>   There was an overload and the fuse blew.
> 2. Why was there an overload?
>   The bearing was not sufficiently lubricated.
> 3. Why was it not lubricated sufficiently?
>   The lubrication pump was not pumping sufficiently.
> 4. Why was it not pumping sufficiently?
>   The shaft of the pump was worn and rattling.
> 5. Why was the shaft worn out?
>   There was no strainer attached and metal scrap got in.
>
> Source: T. Ohno: „Toyota Production System. Beyond Large-Scale Production", Productivity Press, 1988

Extreme Programming (XP)

Root-Cause Analysis

Taiichi Ohno:

> Repeating why five times, like this, can help uncover the root problem and correct it.
>
> If this procedure were not carried through, one might simply replace the fuse or the pump shaft.
>
> In that case, the problem would recur within a few months.
>
> [...]
>
> By asking why five times and answering it each time, we can get to the real cause of the problem, which is often hidden behind more obvious symptoms.

Source: T. Ohno: „Toyota Production System. Beyond Large-Scale Production", Productivity Press, 1988

Extreme Programming (XP)

Shared Code

Kent Beck:

> *Anyone on the team can improve any part of the system at any time. If something is wrong with the system and fixing it is not out of scope for what I'm doing right now, I should go ahead and fix it. [...]*
>
> *Pair programming helps teammates demonstrate their commitment to quality to each other and helps them normalize their expectations for what constitutes quality.*
>
> *Continuous integration is another important prerequisite for collective ownership. [...] Two pairs making many, widespread changes increase the chance of expensive-to-resolve incompatible changes. If the team is making lots of changes, it may want to reduce the interval between integrations to keep the cost of integration down.*

Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Code and Tests

Kent Beck:

> *Maintain only the code and the tests as permanent artifacts. Generate other documents from the code and the tests. Rely on social mechanisms to keep alive important history of the project.*
>
> *Customers pay for the what the system does today and what the team can make with the system tomorrow. Any artifacts contributing to these two sources of value are themselves valuable. Everything else is waste. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Single Code Base

Kent Beck:

> *[...] Multiple code streams are an enormous source of waste in software development. I fix a defect in the currently deployed software. Then I have to retrofit the fix to all the other deployed versions and the active development branch. Then you find that my fix broke something you were working on and you interrupt me to fix my fix. And on and on. [...]*
>
> *Don't make more versions of your source code. Rather than add more code bases, fix the underlying design problem that is preventing you from running from a single code base.*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Daily Deployment

Kent Beck:

> *Put new software into production every night. Any gap between what is on a programmer's desk and what is in production is a risk. [...]*
>
> *Daily deployment is a corollary practice because it has so many prerequisites. The defect rate must be at most a handful per year. The build environment must be smoothly automated. The deployment tool must be automated, including the ability to roll out incrementally and roll back in case of failure. Most importantly, the trust in the team and with customers must be highly developed. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Negociated Scope Contract

Kent Beck:

> *Write contracts for software development that fix time, costs, and quality but call for an ongoing negotiation of the precise scope of the system. Reduce risk by signing a sequence of short contracts instead of a long one.*
>
> *You can move in the direction of negociated scope. Big, long contracts can be split in half or thirds, with the optional part to be exercised only of both parties agree. [...]*
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Extreme Programming (XP)

Pay-Per-Use

Kent Beck:

> With pay-per-use systems, you charge for every time the system is used. [...] Connecting money flow directly to software development provides accurate, timely information with which to drive improvement. [...]
>
> Even if you can't implement pay-per-use, you might be able to go to a subscription model, in which software is purchased monthly or quarterly. With the subscription model, the team at least has the retention rates (...) as a source of information about how the team is doing. [...]
>
> One objection to pay-per-use is that customers want predictable costs. If the price advantage of pay-per-use is large enough, customers may not mind. [...]
>
> Source: K. Beck: „Exteme Programming Explained. Embrace Change", Addison-Wesley, 2nd edition, 2005

Example: Transaction fees (Phone calls, stock exchange transactions, ...).