Practices of an Agile Developer

Venkat Subramaniam
venkats@agiledeveloper.com

Andy Hunt
andy@pragmaticbookshelf.com

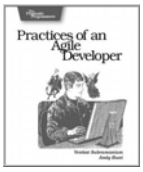
Abstract

You have worked on software projects with varying degree of success. What were the reasons for the success of your last project? What were the reasons for those that failed? A number of issues contribute to project success - some non-technical in nature. In this presentation the speakers will share with you practices in a number of areas including coding, developer attitude, debugging, and feedback. The discussions are based on the book with the same title as the talk.

In this session you will learn about practices beyond what well know methodologies prescribe. While we reemphasize some popular practices, we will also discuss other often overlooked, but important practices - practices that contribute to success of projects.

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

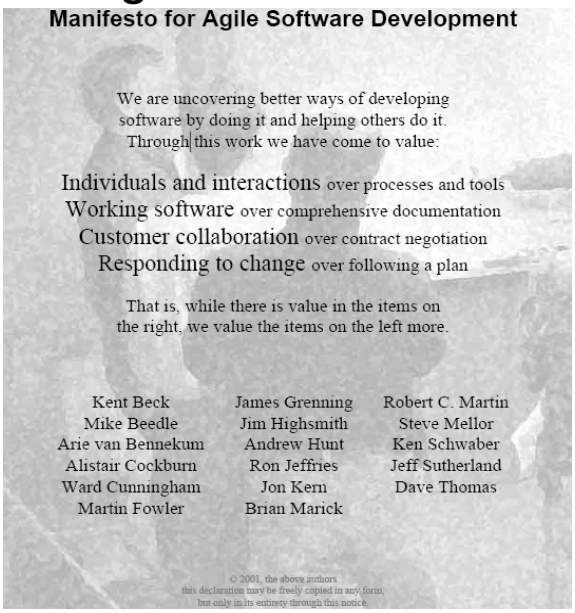


Agile Software Development

- What's makes software development challenging?
- What's Agility?
- The Spirit of Agility

Continuous development, not episodic

Agile Manifesto



Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Practices and Balance

- We'll start with often convincing, but troubling thoughts
- We'll discuss good practices, recommendations, dos and don'ts

- Summarize our advice for the practice
- What It Feels Like and
- Keeping your balance

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Practices Discussed

- We'll discuss *select* practices in areas
  - Beginning Agility
  - Feeding Agility
  - Delivering What Users Want
  - Agile Feedback
  - Agile Debugging
  - Agile Collaboration

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Beginning Agility

- What makes a big difference in Agile Development?
- It's not tools, techniques, process,…
- It's you and your team
  - Your attitude, your ego, how you get along with the team, how the team gets along with you – makes a big difference
- So, we start by focusing on you and your attitude

Work For Outcome

The first, and most important step in addressing a problem is to determine who caused it. Find that moron! Once you've established fault, then you can make sure that the problem doesn't happen again. Ever.

- Worst kind of job to hold – working in a reactive team
- Fixing the problem must be top priority
- Blame don't fix bugs
- Focus on fixing problem than affixing blame
- Be part of a solution, not the problem

Work for Outcome…

Be part of the solution, not the problem. Blame doesn't fix bugs or create working code, so instead of pointing fingers, point to possible solutions—every mistake is an opportunity to learn, and it's the positive outcome that counts.

What It Feels Like

It feels safe to admit that you don't have the answer. A big mistake feels like a learning opportunity, not a witch hunt. It feels like the team is working together, not blaming each other.

Keeping Your Balance

- "It's not my fault" is rarely true. "It's all your fault" is usually equally incorrect
- If you aren't making any mistakes, you're probably not trying hard enough
- No point arguing if it's a feature or flaw, if you can fix the darn thing quick…

Criticize Ideas, Not People

You have a lot invested in your design. You've put your heart and soul into it. You know it's better than any one else's. Don't even bother listening to their ideas; they'll just confuse the issue.

- We take pride in what we do
- Design discussions sometime get out of hand – focused on who instead of what
- Ways to present your questions…
- Development is innovative, requires several minds and creative ideas
- Negativity kills innovation
- Ideas shape solutions
- Set deadline, argue opposite, use mediator, support decision
- Realize life is full of compromise

Criticize Ideas, Not People...

Criticize ideas, not people. Take pride in arriving at a solution rather than proving whose idea was better.

What It Feels Like

It feels comfortable when the team discusses the genuine merits and possible drawbacks of several candidate solutions. You can reject solutions that have too many drawbacks without hurt feelings, and imperfect (but still better) solutions can be adopted without guilt.

Keeping Your Balance

- Always contribute ideas, but don't expect everyone to use it
- Be realistic, fair, ask yourself if your concerns are reasonable
- No "best practices," there are only "better practices"
- Be unemotional, not indifferent or unconcerned

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Feeding Agility

- Agility requires constant motion
- In corporate world, there's only one person to look out for your interests – you
- Evolution appears episodic if you hide in a cave
- You have to keep up with change – incrementally and iteratively

Keep Up With Change

Technology changes so fast it's overwhelming. That's just the nature of it. Stick to your old job with the language you know; you can't possibly keep up.

- You are in an exciting, ever changing field
- Learning is part of our profession
- You can't be an expert at everything, but
- Don't stay ignorant of what's evolving
- Learn iteratively and incrementally
- Get the latest buzz
- Attend local user groups
- Attend workshops or conferences
- Read voraciously

Keep Up With Change...

Constantly be aware of new and emerging technologies. You don't have to become an expert at everything, but be familiar with where the industry is headed.

What It Feels Like

You feel aware of what's going on; you know about technologies as they are announced and adopted. If you had to switch jobs into a new technology area, you could.

Keeping Your Balance

- Gauge your effort – not everything new matures
- Don't try to be an expert in everything
- If you're an expert at a few things, it's easier to gain expertise in selected new areas
- Understand not just technology, but what it solves
- Don't jump up to convert you app to new technology just for the sake of learning

Invest In Your Team

Don't share what you know—keep it to yourself. It's to your advantage to be the Smart One on the team. As long as you're smart, you can forget about those other losers.

- Members of your team have different expertise and strengths
- It's to your benefit to be in mature and qualified team
- You loose what you don't use
- Have brown-bag sessions
- Raise the awareness of your entire team

Invest In Your Team…

Raise the bar for your team—increase their knowledge and skills. Brown bag sessions help bring people together, getting them excited about technologies or techniques that will benefit your project.

What It Feels Like

It feels like everyone is getting smarter. The whole team is aware of new technology, and starts pointing out how to apply it, or points out pitfalls to watch for.

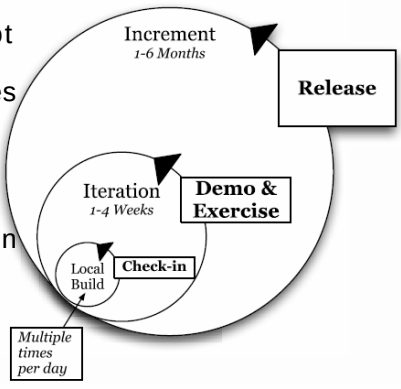
Keeping Your Balance

- Pick good books for your sessions
- Not all the topics will be winners, or even seem appropriate at the moment. Pay attention anyway
- Keep discussion in the team
- Stretch beyond purely technical books
- Don't turn these into design meetings

Feel The Rhythm

We haven't had a code review in a long time, so we're going to review everything all this week. Also, it's probably about time we made a release as well, so we picked three weeks from Tuesday for a next release.

- Failure walks hand-in-hand with irregularity
- Haphazard activities often jolt you, not help
- Agile projects have rhythms and cycles
- Makes life easier
- A lot of practices have to happen all the time, through our your development cycle
- Several small activities have to happen – check in code often, review in increments, do continuous builds, …
- Biggest rhythm of all is iteration itself
- Set the length so it's easier to reach decisions, keep the project moving forward



Feel The Rhythm…

Don’t let tasks bunch up, instead, tackle them regularly. It’s easier to tackle them regularly when you maintain steady, repeatable intervals between events.

What It Feels Like

It feels like consistent, steady rhythm. Edit, run tests, review, over a consistent iteration length, and release. It’s easier to dance when you know when the next beat falls.

Keeping Your Balance

- Have no leftovers – have all code tested and checked in by end of day
- Follow fixed regular length iteration
- Find a comfortable iteration length and stick to it
- Set small reachable goals, celebrate your success

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Delivering What Users Want

- Agility depends heavily on your ability to identify and adapt change
- Affects your ability to develop
  - on time
  - within budget
  - and creating a system that actually meets the users' needs

Let Customers Make Decisions

Developers are creative and intelligent, and know the most about the application. Therefore, developers should be making all the critical decisions. Any time the business people butt in they just make a mess of things; they don't understand logic the way we do.

- Developers make a lot of decisions
- Not all decisions must be made by developers – especially critical business decisions
- You can let customers decide now or they will decide later at much greater cost
- Present customers with pros and cons, show potential cost and benefits from business point of view

Decide what you shouldn't decide

Let Customers Make Decisions...

Present the details and let your customers decide. Developers, managers, or business analysts shouldn't make business critical decisions—let the business owners make those.

What It Feels Like

Business applications are developed as a partnership between the business owner and the developers. It should feel like a partnership—a good, honest, working relationship.

Keeping Your Balance

- Record decisions and reasoning, but don't turn that into documentation heavy
- Don't bug busy business people with trivial low-level details that don't impact business
- Don't assume a low-level details doesn't impact business
- Don't know is an acceptable answer for a business owner. Advise best you can, take their input, and prepare for eventual change.

Let Design Guide, not Dictate

Design documents should be as detailed as possible, so that any lowly coder can just type in the code… Don't forget all the fields of the class. Never deviate from the design, no matter what you discover while coding.

- Design is key to Agile development
  - Don't use Agility as an excuse to hacking
  - Agility discourages ceremony, not design
- Design will evolve as your understanding evolves
- Strategic Design vs. Tactical Design
- How do you evaluate the quality of design?

Let Design Guide, not Dictate…

A good design is a map that points you in the right direction. It's not the territory itself; it shouldn't dictate the specific route. Do not let the design (or the designer) hold you hostage.

What It Feels Like

A good design is accurate, but not precise. That is, what it says should be correct, but it shouldn't go far as to include details that might change or that are uncertain. It's an intent, not a recipe.

Keeping Your Balance

- No "Big Design Up-front" doesn't mean no design
- Design will change, but there is still value in initial design – what you lean from it is invaluable
- No need to be bogged down with heavy weight tools
  - If white boards, sketches and PostIt notes are excellent design tools

Fixed Prices are Broker Promises

We have to deliver a fixed bid for this project. We don't have all the details yet, but need to put a bid in. I need an estimate for the whole team by Monday, and we'll have to deliver the whole project by the end of the year.

- Telling customers "you'll know when we're done" is not reasonable
- But, fixed price contracts are problematic
- We have to keep up with requirements change, technology change
- How to avoid broker promise, can we estimate better, or enter into different deal?
- You can estimate better if project is similar to what you've done before. Estimate if you can.
- Alternately
  - Offer to build initial, small, useful portion of system within 6-8 weeks
  - Iterative development can help you and your clients see how things are progressing
  - Allow client to continue or pull the plug anytime

Fixed Prices are Broker Promises...

Let the team work on this project, with this client, to get realistic estimates. Give the client control over their features and budget.

What It Feels Like

Your estimates will change throughout the project—they aren't fixed. But you'll feel increasingly confident that you can forecast the amount accomplished with each iteration better and better. Your estimates improve over time.

Keeping Your Balance

- If you're in a plan-based, non-agile environment, consider a plan-based, non-agile development methodology, or consider a different environment
- If you refuse to give estimates, you may lose the contract to someone else who gives an estimate, however, unrealistic it is
- Being agile doesn't mean, "don't ask me for estimates"
- You might consider fixed price per iteration set in the contract

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Agile Feedback

- Feedback is a critical component of Agile Development to make small continuous adjustments
- What are the ways to get feedback?
- Feedback can come from
  - Code
  - Builds
  - Team
  - ...

Different Makes a Difference

As long as the code works on your machine, that's okay. Who cares if it works on some other platform; you don't have one.

- "It works on my machine" isn't good enough
- If something is different, it will make a difference
- If your product has to run on different versions of JVM, CLR, platform of OS, ... it's your responsibility to make sure it works well on those
- Don't depend on your users to figure that our for you
- Automate unit tests on platforms
- Use VMWare or Virutal PC if hardware is a concern

Automate to save time

Different Makes a Difference...

Run unit tests on each supported platform and environment combination. Use continuous integration tools to run the tests. Find problems proactively before they find you.

What It Feels Like

It feels like unit testing, only more so—it's unit testing across multiple worlds.

Keeping Your Balance

- Hardware is cheaper than your time
- Be selective if you have too many platforms
- Even if you have fewer clients on a platform, it is necessary to test
- Set up your continuous integration tool so you're not bombarded with several notifications for same error

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Agile Debugging

- Murphy's law says "If something can go wrong, it will"
- You can't timebox debugging sessions
- But, you can be proactive about bugs and problems, however
  - Learning to keep solution logs
  - Treating warnings as errors
  - Attacking problems in isolation…

Attack Problems in Isolation

Stepping line-by-line through a massive code base is pretty scary. But the only way to debug a significant problem is to look at the entire system. All at once. After all, you don't know where the problem may be, and that's the only way to find it.

- Stepping through code may help catch more stress
- Dealing with entire code base does not make it easy to ask for help when problems erupt
- Layering is a collateral advantage of unit testing
- Isolate problem from its surroundings
- You can focus more on what's relevant
  - You can experiment without worries
  - You can get to the problem quicker

Prototype to isolate

Attack Problems in Isolation...

Attack problems in isolation. Separate the problem from its surroundings, especially in a large application, and you'll save time and reduce stress.

What It Feels Like

When faced with a problem that you have to isolate, it feels like searching for a needle in a teacup, not a needle in a haystack.

Keeping Your Balance

- If you separate code from its environment and
  - the problem goes away, you've helped isolate the problem
  - the problem is still there, you've helped isolate the problem
- You may use a binary chop to isolate the problem – divide the problem space into half until you have found the smallest part with the problem
- Check your solutions log to see if you've seen this problem before

Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Agile Collaboration

- Team and Team work critical for Agile Development
- Your actions have consequences on the team's productivity and progress
- Everyone's action must be relevant to the context of the project
- Individuals actions, in turn, affect the project context
- What can we do to be effective in the team…

Schedule Regular Face Time

You need to hold meetings—lots of them. In fact, we're going to keep scheduling more meetings until we discover why no work is getting done.

- Most of us may hate meetings
- But communication is key
- How do we know what everyone is doing?
- We don't want an isolated developer fighting an irrelevant problem or a problem with solution others on the team know
- Standup meetings help a great deal
- Very short meeting where developers share
  - What did I achieve yesterday?
  - What am I planning to do today?
  - What's in my way?
- Several advantages
  - Kicks off the day
  - Brings issues into the open
  - Helps determine areas that need additional helping hands
  - Keeps people abreast
  - Speeds development by sharing code and ideas
  - Encourages forward momentum, seeing others' progress motivates each of us

Schedule Regular Face Time...

Use standup meetings to keep the team on the same page. Keep the meeting short, focused, and intense.

What It Feels Like

You look forward to the stand up meeting. You get a good sense of what everyone else is working on, and can bring problems out into the open easily.

Keeping Your Balance

- Start meeting promptly
- Keep it short, focused, utilize time wisely
- For smaller teams, reduce frequency of meeting
- Keep a watch on level of details presented
- If you find this a waste of time, may be your not operating as a team

Be a Mentor

It took you a long time and a lot of hard work to get where you are. Keep it to yourself so you look better. Use your superior skill to intimidate your teammates.

- You may know more about certain things than anyone else on your team
- What can you do with this new-found authority?
  - Criticize others, make fun of their decisions and code, or
  - Share what you know, making everyone around you better
- By taking time to explain you
  - get better understanding of it yourself
  - get a different perspective
- Being a Mentor, you don't spoon feed people
- You help them learn
- Be a mentor, not a tormentor

Knowledge grows when given

Be a Mentor…

Make a conscious effort to be a mentor. There's fun in sharing what you know—you gain as you give. You motivate others to achieve better results. You improve the competence of your team.

What It Feels Like

You find that teaching is another way to improve your own learning, and others come to trust that you can help them.

Keeping Your Balance

- If you teach same thing to different people, keep notes and write an article or a book on the topic
- You are investing in yourself when you're a mentor
- Pair programming can be effective way to mentor
- Don't let lazy developers interrupt you - help them figure out answers
- Don't torment others, give them the answer if they're really lost

Quiz Time

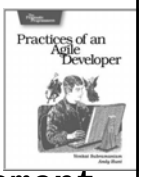
Practices of an Agile Developer

- Agile Software Development
- Devil and the details
- Select Practices
- Beginning Agility
- Feeding Agility
- Delivering What Users Want
- Agile Feedback
- Agile Debugging
- Agile Collaboration
- Epilogue

Succeeding on Projects

- We've discussed only a dozen practices
  - 45 Practices covered in the book
- One New Practice
- Not all at once
- Which practices should you use?
- Where do you go from here?

References...



- Venkat Subramaniam and Andy Hunt, "Practices of an Agile Developer," The Pragmatic Programmers.
- Ken Schwaber, "Agile Project Management with Scrum," Microsoft Press.
- Jared Richardson and Will Gwaltney, "Ship It!," The Pragmatic Programmers.
- Johanna Rothman and Esther Derby, "Behind Closed Doors: Secrets of Great Management," The Pragmatic Programmers.
- Andy Hunt and Dave Thomas, "The Pragmatic Programmer," Addison-Wesley.
- Peter Senge, "The Fifth Discipline: The Art and Practice of the Learning Organization," Currency/Doubleday.