Great software development is...

We've talked about several things that you'll need for successful software. You've got the customer's big ideas to deal with, their money you're spending, and a schedule you've got to worry about. You've got to get all of those things right if you're going to build consistently great software.

Great software development delivers...

What the customer needs, otherwise called the software requirements. We'll talk more about requirements in the next chapter...

What is needed,

When we agreed with the customer that the software would be finished.

On Time,

and

On Budget

Not billing the customer for more money than was agreed upon.

BRAIN POWER

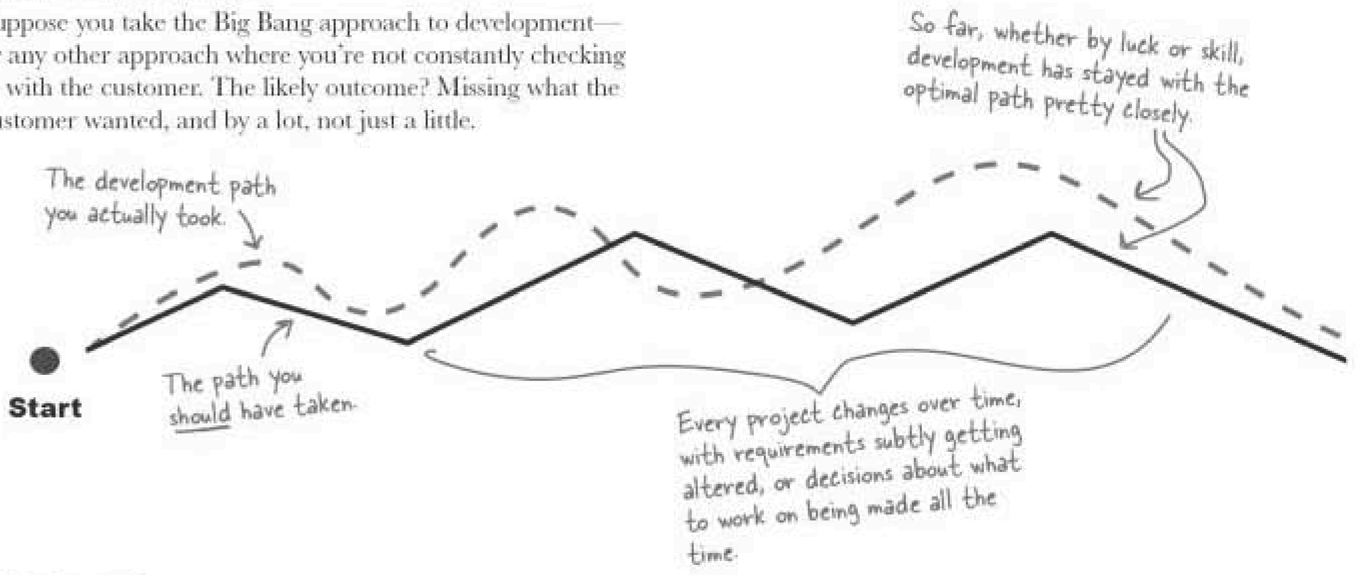
Can you think of three examples of software you've been involved with where at least one of these rules was broken?

Getting to the goal with ITERATION

The secret to great software development is **iteration**. You've already seen that you can't simply ignore the customer during development. But iteration provides you a way to actually ask the question, at each step of development, "How am I doing?" Here are two projects: one without iteration, and one with.

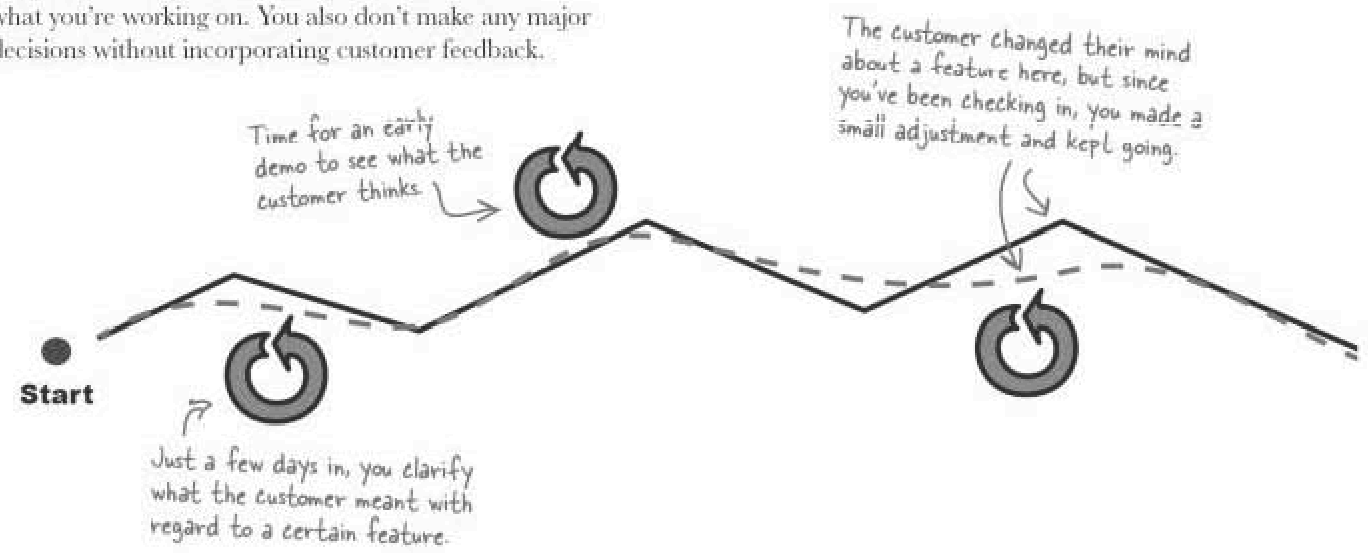
Without iteration...

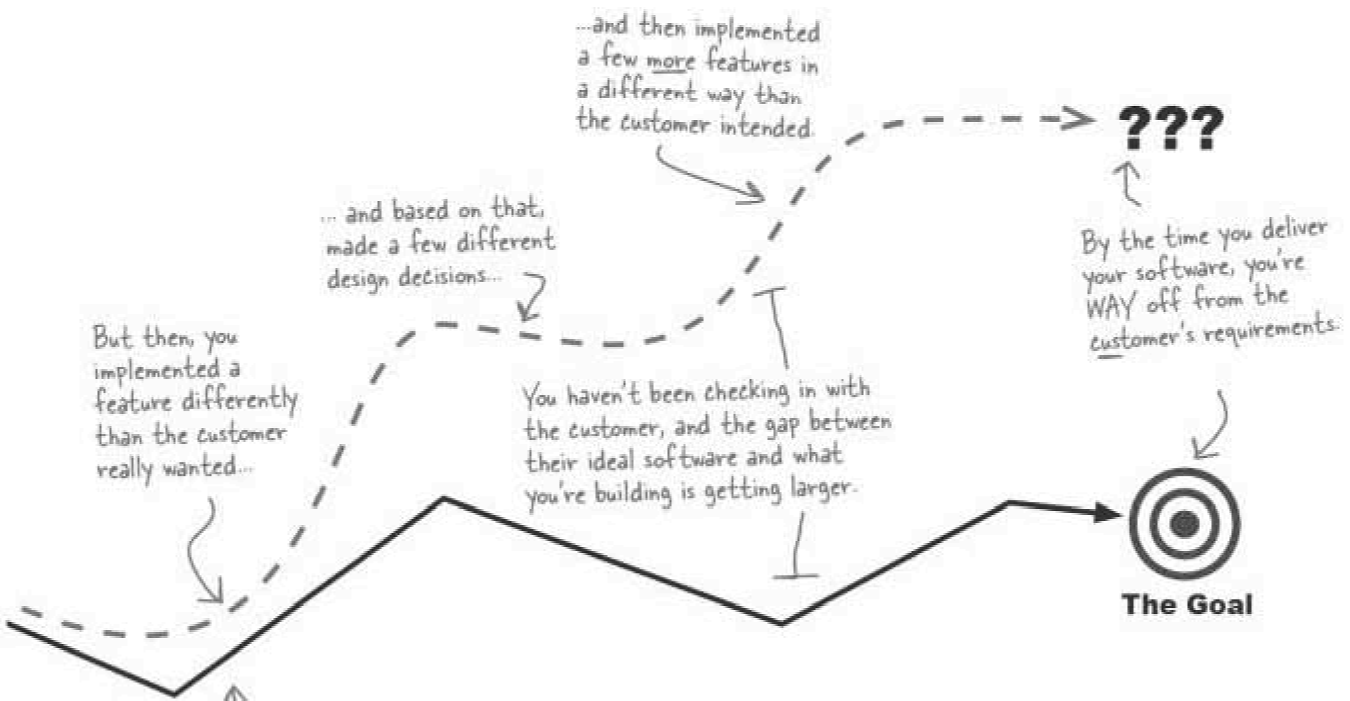
Suppose you take the Big Bang approach to development—or any other approach where you're not constantly checking in with the customer. The likely outcome? Missing what the customer wanted, and by a lot, not just a little.

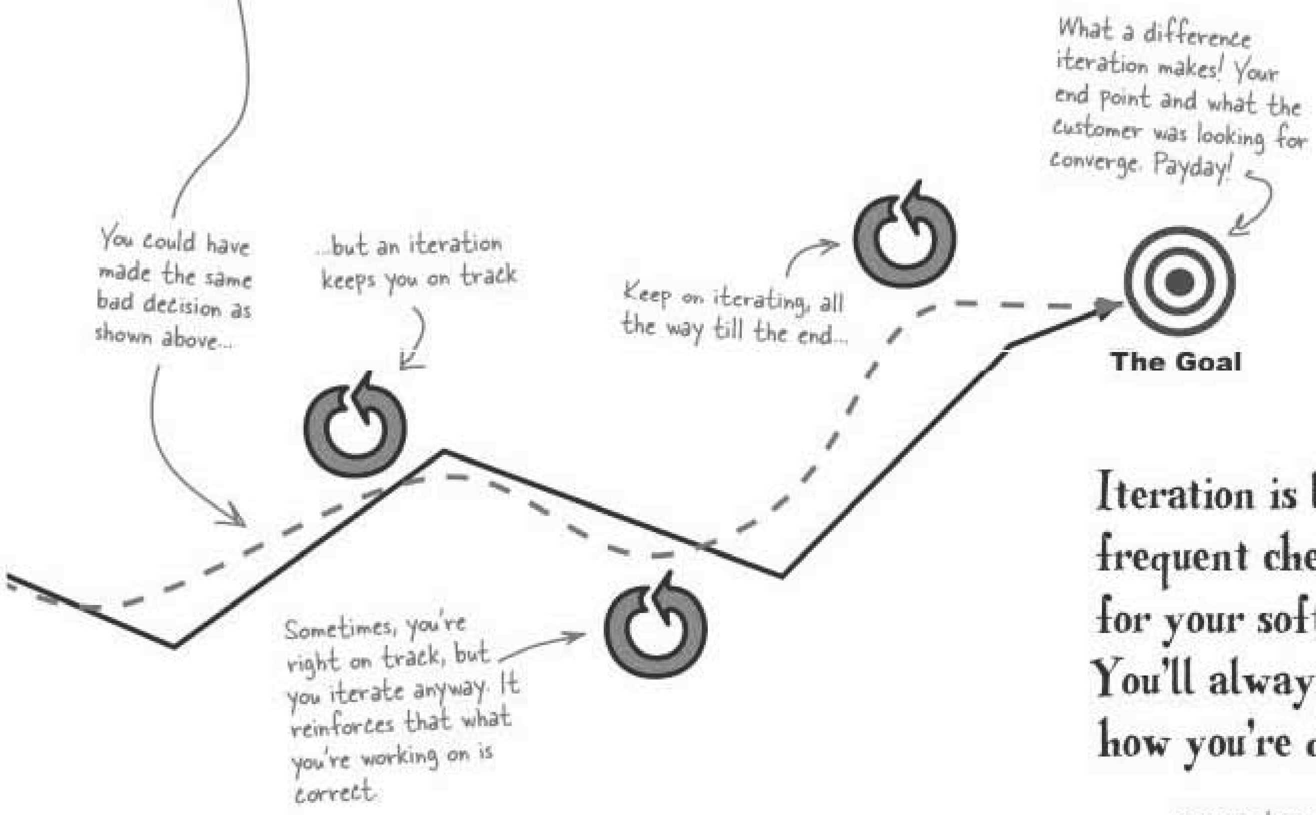

So far, whether by luck or skill, development has stayed with the optimal path pretty closely.
The development path you actually took.
Start
The path you should have taken.
Every project changes over time, with requirements subtly getting altered, or decisions about what to work on being made all the time.

With iteration...

This time, you decide that every time you make significant progress you'll check with the customer and refine what you're working on. You also don't make any major decisions without incorporating customer feedback.


The customer changed their mind about a feature here, but since you've been checking in, you made a small adjustment and kept going.
Time for an early demo to see what the customer thinks.
Start
Just a few days in, you clarify what the customer meant with regard to a certain feature.


...and then implemented a few more features in a different way than the customer intended.
???
... and based on that, made a few different design decisions...
By the time you deliver your software, you're WAY off from the customer's requirements.
But then, you implemented a feature differently than the customer really wanted...
You haven't been checking in with the customer, and the gap between their ideal software and what you're building is getting larger.
The Goal


What a difference iteration makes! Your end point and what the customer was looking for converge. Payday!
You could have made the same bad decision as shown above...
...but an iteration keeps you on track
Keep on iterating, all the way till the end...
The Goal
Sometimes, you're right on track, but you iterate anyway. It reinforces that what you're working on is correct.

Iteration is like a frequent checkup for your software. You'll always know how you're doing.

there are no Dumb Questions

Q: What if I'm sure that I know what the customer wants at the beginning of a project? Do I still need to iterate?

A: Absolutely. Iteration and getting feedback from your customer is important *especially* when you think you know it all up front. Sometimes it can seem like a complete no-brainer on a simple piece of software, but checking back with the customer is ALWAYS worth it. Even if the customer just tells you you're doing great, and even if you actually do start out with all the right requirements, iteration is still a way to make sure you're on the right track. And, don't forget, the customer can always change their mind.

Q: My entire project is only two months long. Is it worth iterating for such a short project?

A: Yep, iteration is still really useful even on a very short project. Two months is a whopping 60 days of chances to deviate from the customer's ideal software, or misunderstand a customer's requirement. Iteration lets you catch any potential problems like this before they creep into your project. And, better yet, before you look foolish in front of your customer.

No matter how big the team, or how long the project, iteration is ALWAYS one of the keys to building great software.

Q: Wouldn't it just be better to spend more time getting to know what the customer really wants, really getting the requirements down tight, than always letting the customer change their mind midstream?

A: You'd think so, but actually this is a recipe for disaster. In the bad old days, developers used to spend ages at the beginning of a project trying to make sure they got all the customer's requirements down completely before a single line of code or design decision was made.

Unfortunately, this approach still failed. Even if you think that you completely understand what the customer needs at the beginning, the ***customer*** often doesn't understand. So they're figuring out what they want as much as you are.

You need a way of helping your team and your customer grow their understanding of their software as you build it, and you can't do that with a Big Bang, up-front requirements approach that expects everything to be cast in stone from day one.

Q: Who should be involved in an iteration?

A: Everyone who has a say in whether your software meets its requirements, and everyone who is involved in meeting those requirements. At a minimum, that's usually your customer, you, and any other developers working on the project.

Q: But I'm only a team of one, do I still need to iterate?

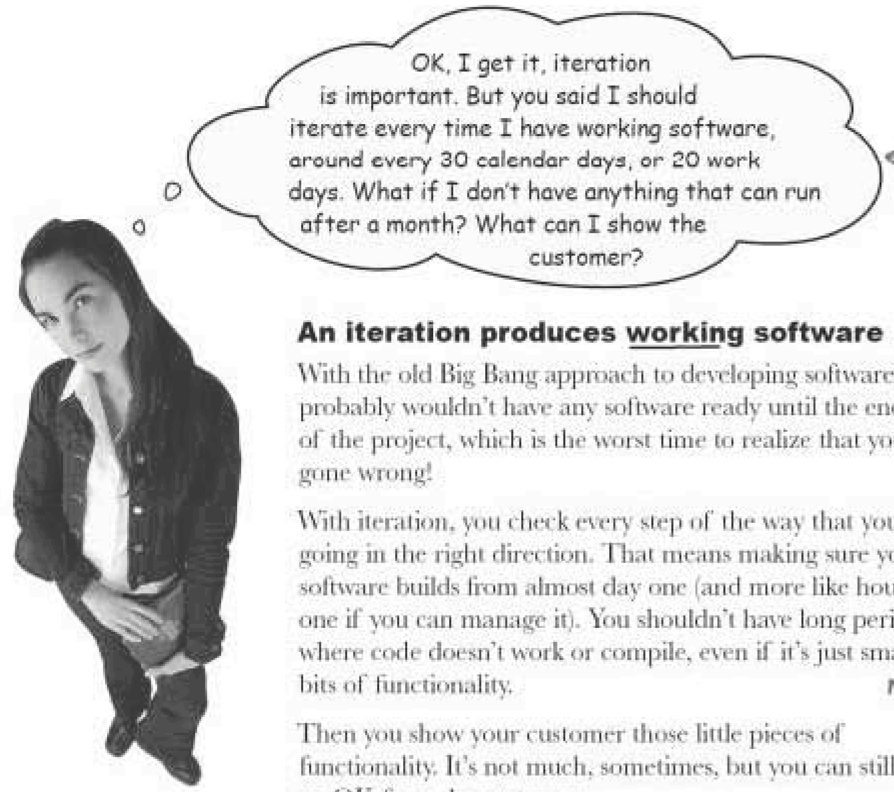
A: Good question, and the answer is yes (starting to detect a theme here?). You might only be a development team of one, but when it comes to your project there are always, at a minimum, two people who have a stake in your software being a success: your customer and you. You still have two perspectives to take into account when making sure your software is on the right path, so iteration is still really helpful even in the smallest of teams.

Q: How early in a project should I start iterating?

A: As early as you have a piece of software running that you can discuss with your customer. We normally recommend around 20 work days—1 calendar month, per iteration as a rule of thumb—but you could certainly iterate earlier. One- or two-week iterations are not unheard of. If you aren't sure about what a customer means on Day 2, call them. No sense waiting around, guessing about what you should be doing, right?

Q: What happens when my customer comes back with bad news, saying I'm way off on what I'm building. What do I do then?

A: Great question! When the worst happens and you find that you've deviated badly during an iteration, then you need to bring things back into line over the course of the next couple of iterations of development. How to do this is covered later on, but if you want to take a peek now, fast-forward to Chapter 4.


OK, I get it, iteration is important. But you said I should iterate every time I have working software, around every 30 calendar days, or 20 work days. What if I don't have anything that can run after a month? What can I show the customer?

20 working days is only a guideline. You might choose to have longer or shorter iterations for your project.

An iteration produces working software

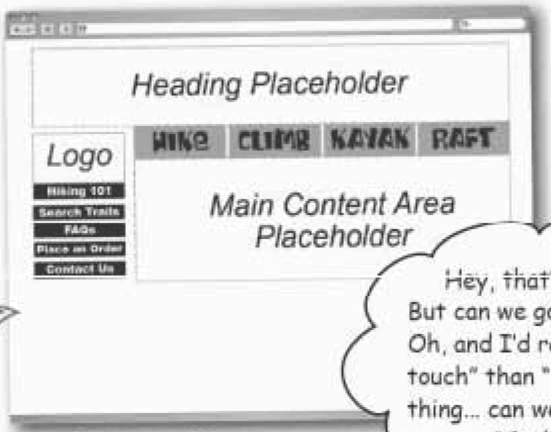
With the old Big Bang approach to developing software, you probably wouldn't have any software ready until the end of the project, which is the worst time to realize that you've gone wrong!

Continuous building and testing is covered in Chapters 6 and 7.

With iteration, you check every step of the way that you're going in the right direction. That means making sure your software builds from almost day one (and more like hour one if you can manage it). You shouldn't have long periods where code doesn't work or compile, even if it's just small bits of functionality.

A working build also makes a big difference to your team's productivity because you don't have to spend time fixing someone else's code before you can get on with your own tasks

Then you show your customer those little pieces of functionality. It's not much, sometimes, but you can still get an OK from the customer.



Tom got to see working software, and made some important comments you could address right away.

Hey, that's looking good. But can we go with rounded tabs? Oh, and I'd rather call it "Get in touch" than "Contact Us." Last thing... can we add an option for "Order Status?"

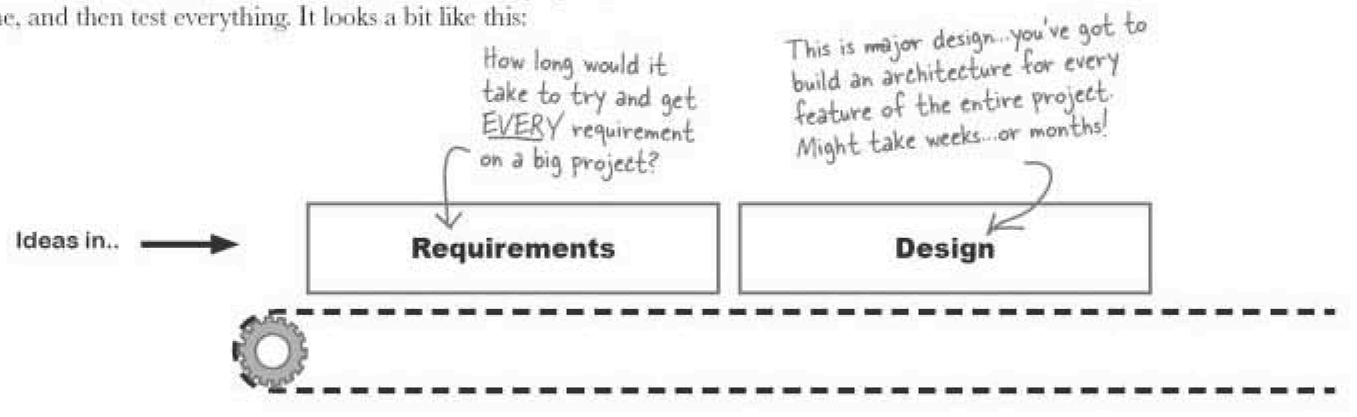
Here's a very simple portion of the Tom's Trails website. It only has the navigation, but it's still worth seeing what Tom thinks.

Instead of building the entire site at once, we broke the problem up into smaller chunks of functionality. Each chunk can then be demonstrated to the customer separately.

Each iteration is a mini-project

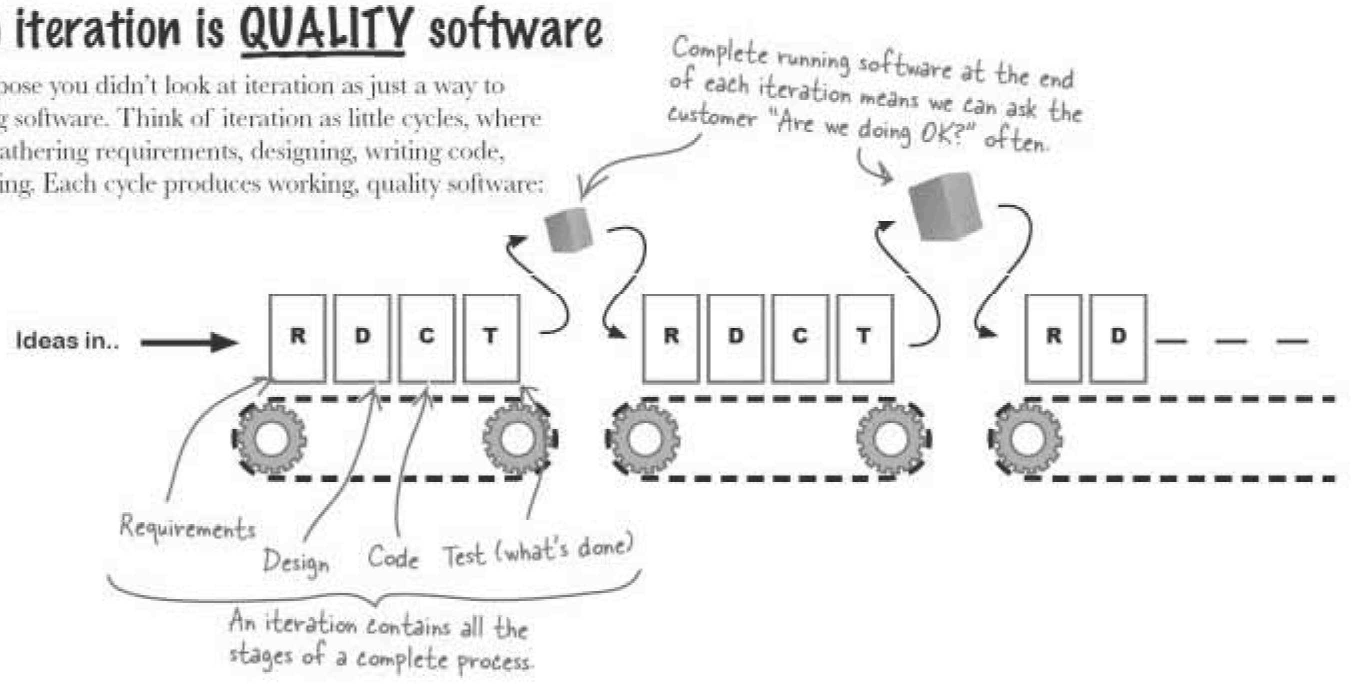
With iteration, you take the steps you'd follow to build the entire project, and put those steps into **each iteration**. In fact, each iteration is a mini-project, with its own requirements, design, coding, testing, etc., built right in. So you're not showing your customer junk... you're showing them well-developed bits of the final software.

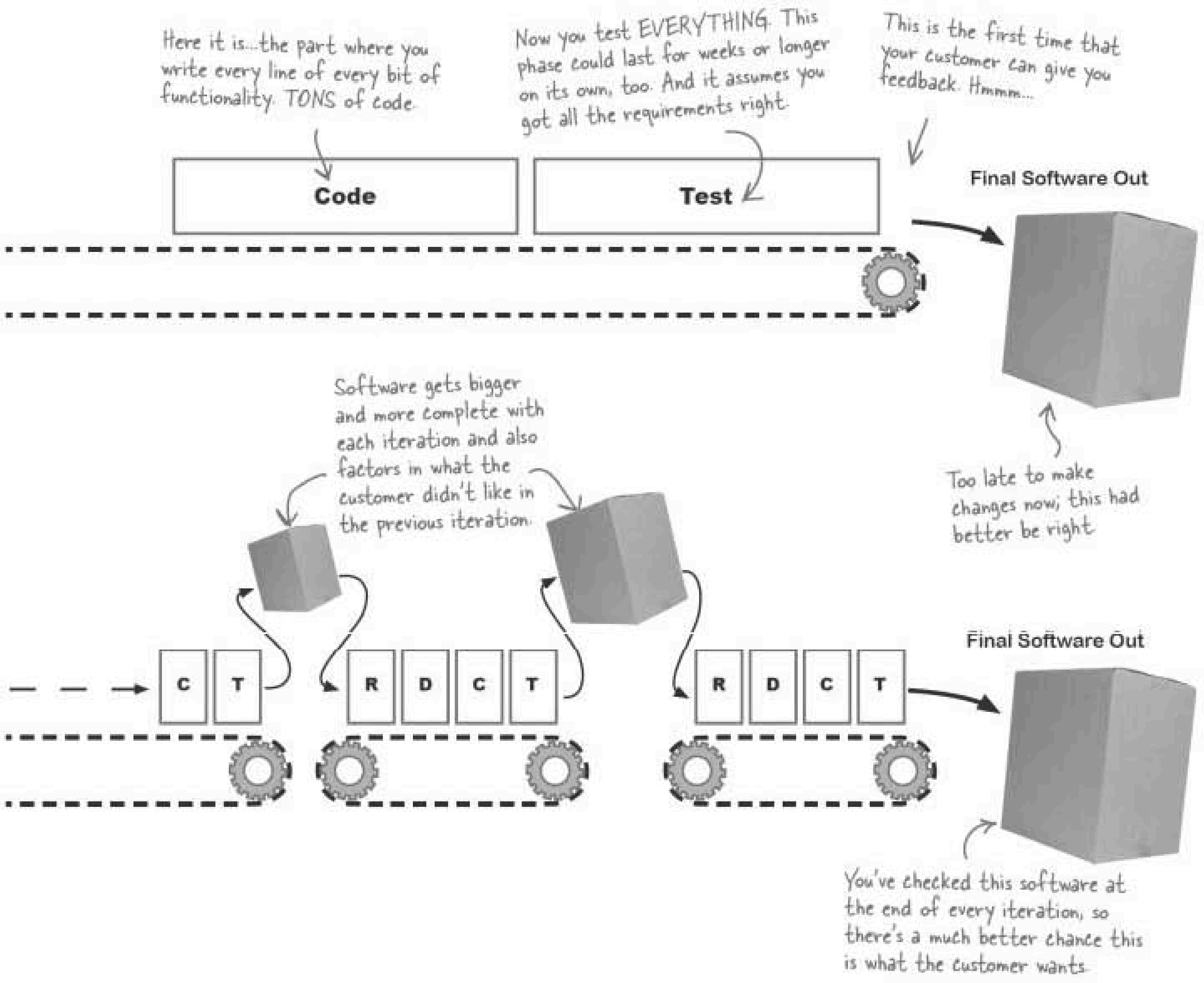
Think about how most software is developed: You gather requirements (what your customer wants), build a design for the entire project, code for a long time, and then test everything. It looks a bit like this:


How long would it take to try and get EVERY requirement on a big project?
This is major design...you've got to build an architecture for every feature of the entire project. Might take weeks...or months!
Ideas in..
Requirements
Design

Each iteration is QUALITY software

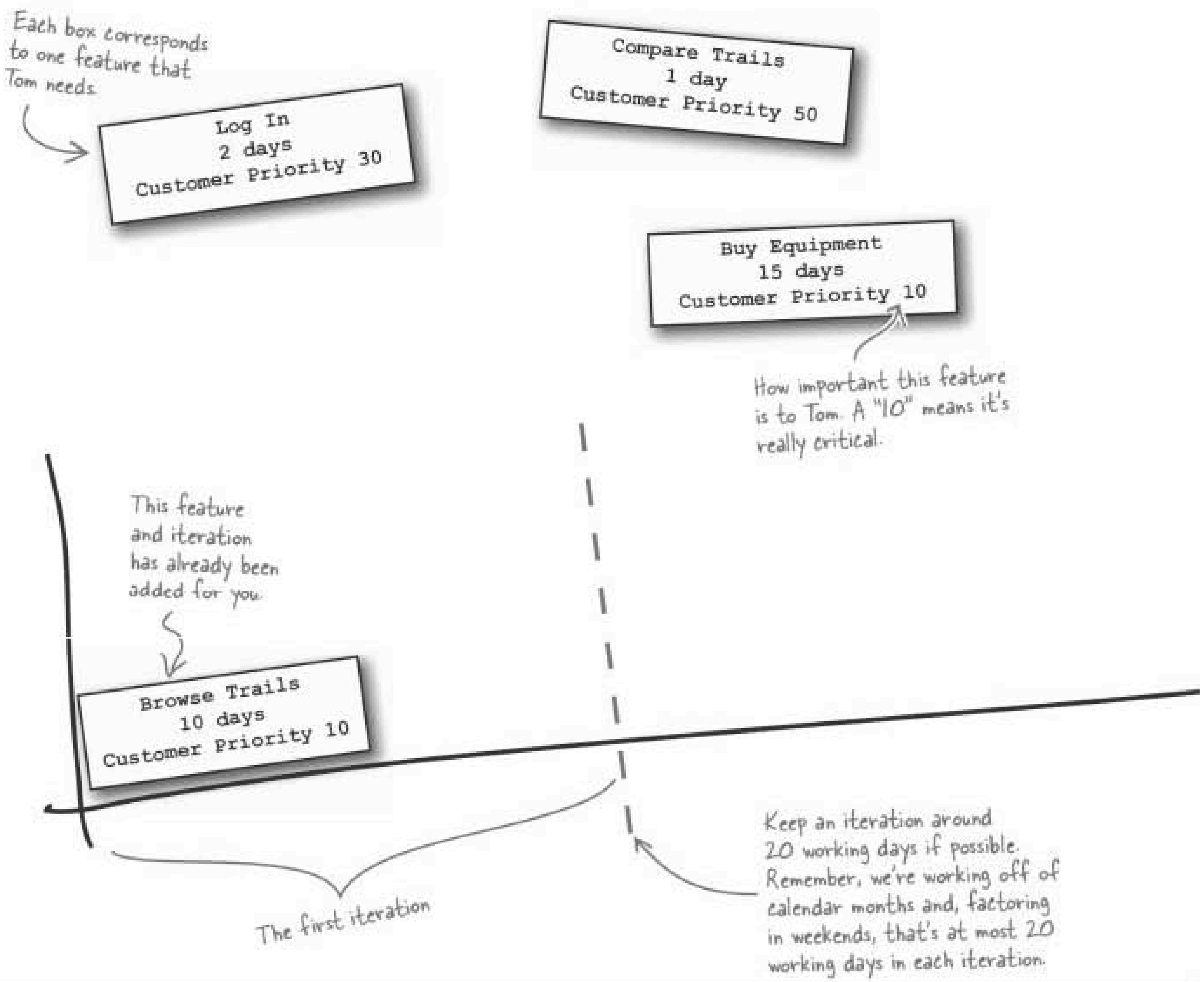
But suppose you didn't look at iteration as just a way to write big software. Think of iteration as little cycles, where you're gathering requirements, designing, writing code, and testing. Each cycle produces working, quality software:


Complete running software at the end of each iteration means we can ask the customer "Are we doing OK?" often.
Ideas in..
R
D
C
T
R
D
C
T
R
D
Requirements
Design
Code
Test (what's done)
An iteration contains all the stages of a complete process.


Here it is...the part where you write every line of every bit of functionality. TONS of code.
Now you test EVERYTHING. This phase could last for weeks or longer on its own, too. And it assumes you got all the requirements right.
This is the first time that your customer can give you feedback. Hmmm...
Code
Test
Final Software Out
Too late to make changes now; this had better be right.
Software gets bigger and more complete with each iteration and also factors in what the customer didn't like in the previous iteration.
C
T
R
D
C
T
R
D
C
T
Final Software Out
You've checked this software at the end of every iteration, so there's a much better chance this is what the customer wants.

Exercise

It's time to bring iterations into play on Tom's Trails. Each of the features that Tom wants for Trails Online has had estimates added to specify how long it will take to actually develop. Then we figured out how important each is to Tom and then assigned a priority to each of them (10 being the highest priority, 50 being the lowest). Take each feature and position them along the project's timeline, adding an iteration when you think it might be useful.


Each box corresponds to one feature that Tom needs.
Log In
2 days
Customer Priority 30
Compare Trails
1 day
Customer Priority 50
Buy Equipment
15 days
Customer Priority 10
How important this feature is to Tom. A "10" means it's really critical.
This feature and iteration has already been added for you.
Browse Trails
10 days
Customer Priority 10
The first iteration
Keep an iteration around 20 working days if possible. Remember, we're working off of calendar months and, factoring in weekends, that's at most 20 working days in each iteration.

Oh, one other thing. Tom doesn't want customers to be able to buy equipment unless they've logged in to the web site. Be sure and take that into account in your plan.

Each feature has an estimate to show how long it should take to develop that feature (in actual working days).

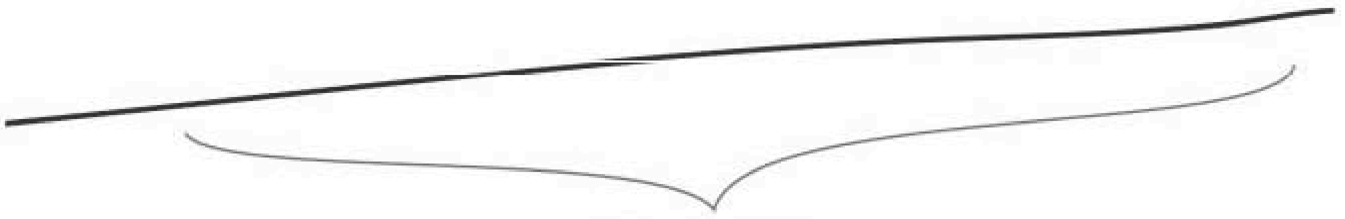
List Equipment
7 days
Customer Priority 10

Add a Review
2 days
Customer Priority 20

View Reviews
3 days
Customer Priority 20

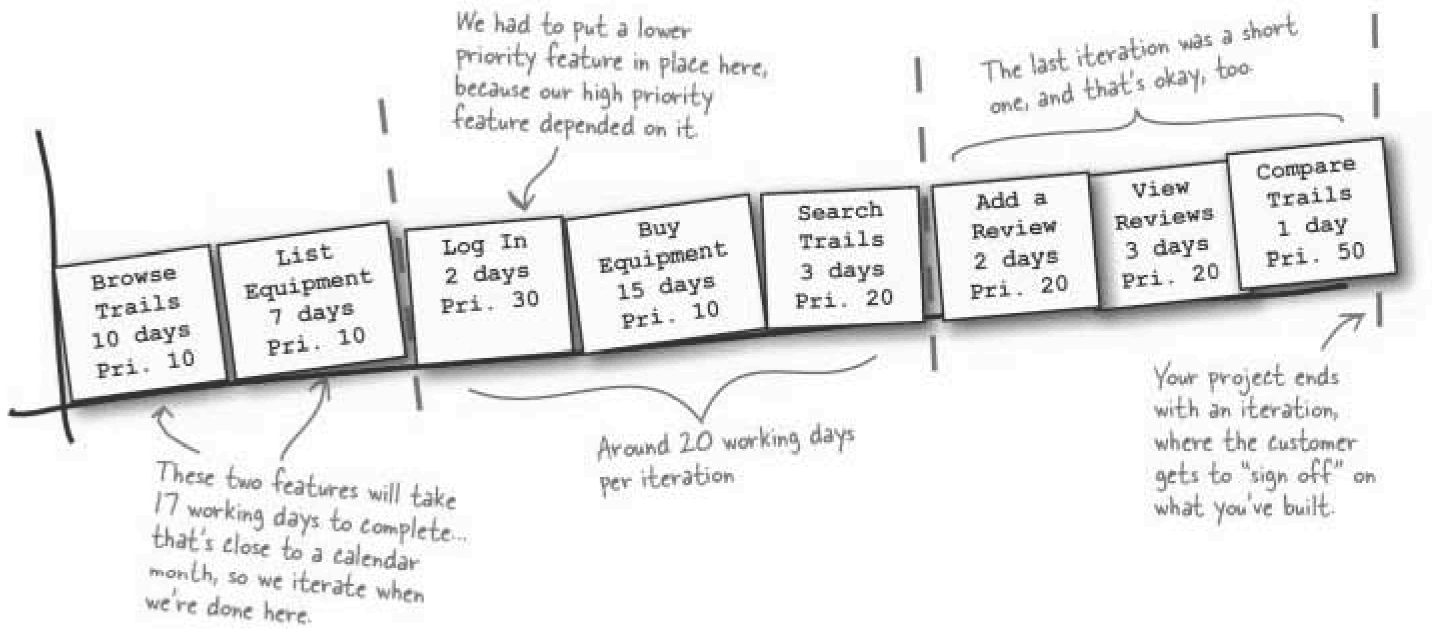
Search Trails
3 days
Customer Priority 20

10 is high priority, 50 is low. We'll look at why these priorities are in increments of 10 in Chapter 3.



Don't forget to add as many iterations as you think will be useful.

Your job was to build an iteration plan for Tom's Trails. You should have come up with something like we did, below:



We had to put a lower priority feature in place here, because our high priority feature depended on it.
The last iteration was a short one, and that's okay, too.
Browse Trails 10 days Pri. 10
List Equipment 7 days Pri. 10
Log In 2 days Pri. 30
Buy Equipment 15 days Pri. 10
Search Trails 3 days Pri. 20
Add a Review 2 days Pri. 20
View Reviews 3 days Pri. 20
Compare Trails 1 day Pri. 50
These two features will take 17 working days to complete... that's close to a calendar month, so we iterate when we're done here.
Around 20 working days per iteration
Your project ends with an iteration, where the customer gets to "sign off" on what you've built.

This is probably the only plan that serves the customer's priorities, keeps iterations at a manageable length, and gets the job done. If you came up with something different, take a hard look at why you made different choices than we did.

I decided to have the customer check on the project after each feature. That's even **better** than iterating every calendar month, right?

Your iteration length should be at the right tempo for YOUR project

An iteration helps you stay on track, and so you might decide to have iterations that are shorter or longer than 30 days. Thirty days might seem like a long time, but factor in weekends, and that means you're probably going to get 20 days of actual productive work per iteration. If you're not sure, try 30 calendar days per iteration as a good starting point, and then you can tweak for your project as needed.

The key here is to iterate often enough to catch youself when you're deviating from the goal, but not so often that you're spending all your time preparing for the end of an iteration. It takes time to show the customer what you've done and then make course corrections, so make sure to factor this work in when you are deciding how long your iterations should be.

there are no Dumb Questions

Q: The last feature scheduled for my iteration will push the time needed to way over a month. What should I do?

A: Consider shifting that feature into the next iteration. Your features can be shuffled around within the boundaries of a 20-day iteration until you are confident that you can successfully build an iteration within the time allocated. Going longer runs the risk of getting off course.

Q: Ordering things by customer priority is all fine and good, but what happens when I have features that need to be completed before other features?

A: When a feature is dependent on another feature, try to group those features together, and make sure they are placed within the same iteration. You can do this even if it means doing a lower-priority feature before a high-priority one, if it makes the high-priority feature possible.

This occurred in the previous exercise where the "Log In" feature was actually a low customer priority, but needed to be in place before the "Buy Equipment" feature could be implemented.

Q: If I add more people to the project, couldn't I do more in each of my iterations?

A: Yes, but be very careful. Adding another person to a project doesn't halve the time it takes to complete a feature. We'll talk more about how to factor in the overhead of multiple people in Chapter 2, when we talk about velocity.

Q: What happens when a change occurs and my plan needs to change?

A: Change is unfortunately a constant in software development, and any process needs to handle it. Luckily, an iterative process has change baked right in...turn the page and see what we mean.

The customer WILL change things up

Tom signed off on your plan, and Iteraton 1 has been completed. You're now well into your second iteration of development and things are going great. Then Tom calls...

Things are really starting to look great, but I had some thoughts after that last iteration. I think it's really important that Tom's Trails Online has a mailing list, so my customers can communicate with each other.

Remember, if your software doesn't do what the customer wants, you're not going to go very far in software development.

It's up to you to make adjustments

Tom's new idea means three new features, all high-priority. And we don't even know how long they'll take, either. But you've got to figure out a way to work these into your projects.

```
Join Mailing List
Customer Priority 20
```

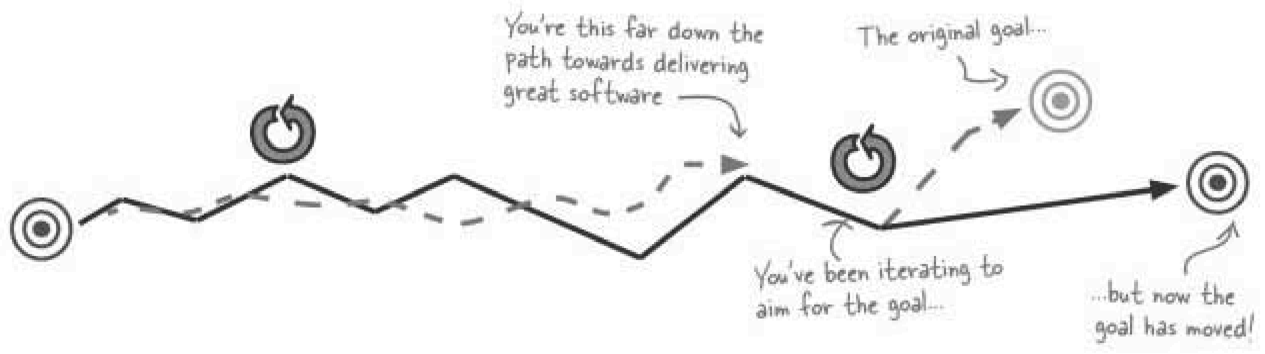
```
Post to Mailing List
Customer Priority 20
```

```
Organize a Trekking Google Group
Customer Priority 20
```

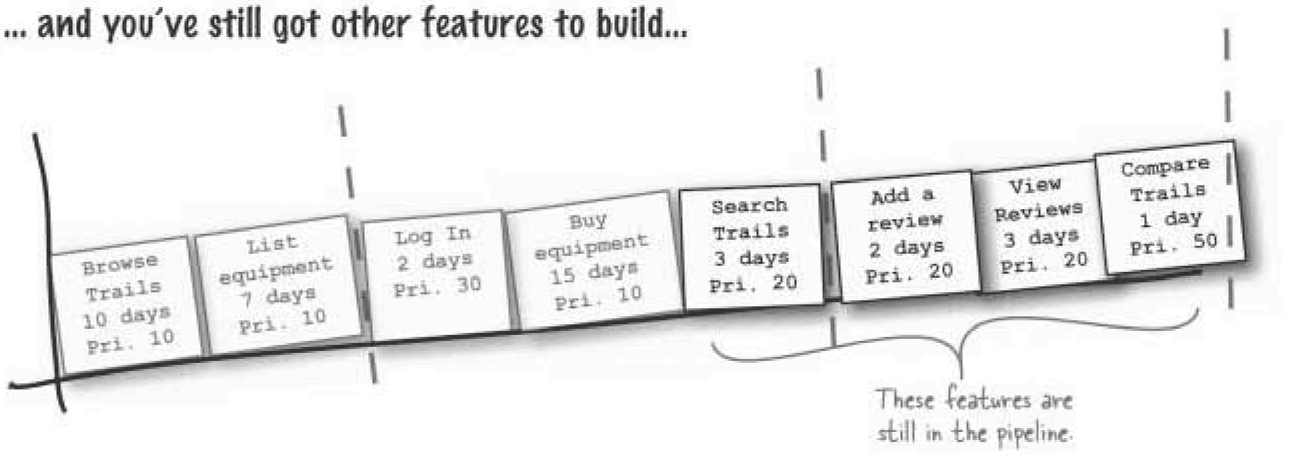
Here are three new features. Tom's given them all a priority of "20", also. Pretty important.

But there are some BIG problems...

You're already a long way into development...


You're this far down the path towards delivering great software
The original goal...
You've been iterating to aim for the goal...
...but now the goal has moved!

... and you've still got other features to build...


Browse Trails 10 days Pri. 10
List equipment 7 days Pri. 10
Log In 2 days Pri. 30
Buy equipment 15 days Pri. 10
Search Trails 3 days Pri. 20
Add a review 2 days Pri. 20
View Reviews 3 days Pri. 20
Compare Trails 1 day Pri. 50
These features are still in the pipeline.

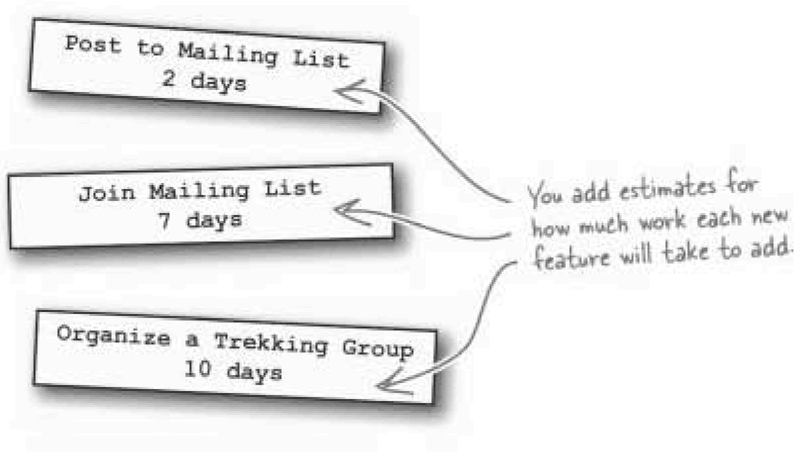
... and the deadline hasn't changed.

You are now here...
Remember the deadline from page 3? It hasn't changed, even though Tom's mind has.
A little over one month until the TrailMix conference!

Iteration handles change automatically^(well, sort of)

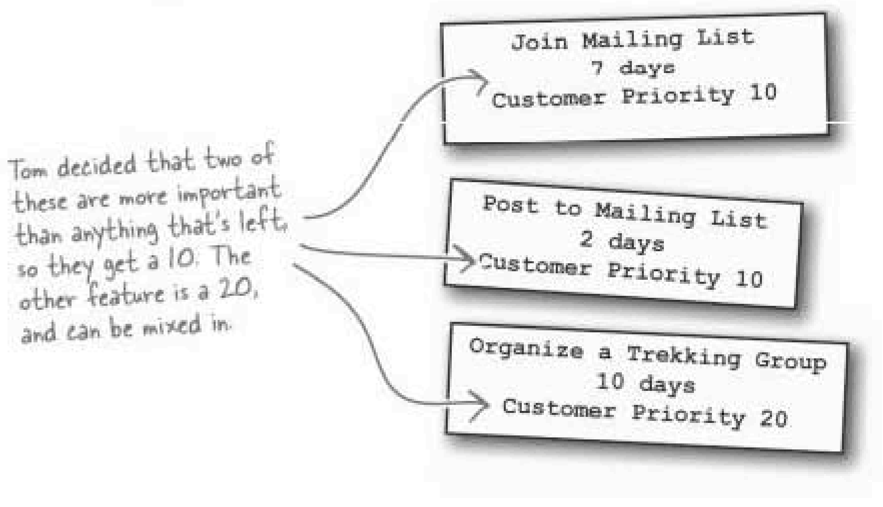
Your iteration plan is already structured around short cycles, and is built to handle lots of individual features. Here's what you need to do:

1. **Estimate the new features**
   First, you need to estimate how long each of the new features is going to take. We'll talk a lot more about estimation in a few chapters, but for now, let's say we came up with these estimates for the three new features:

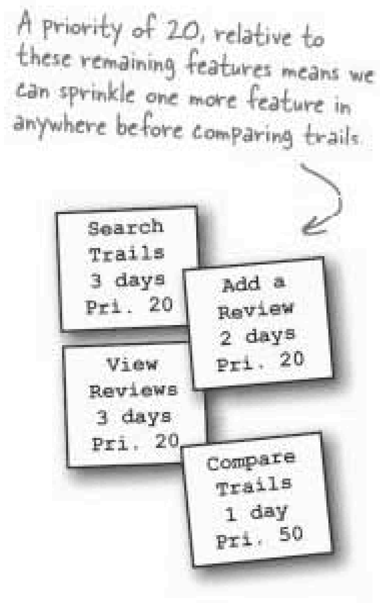


Post to Mailing List
2 days
Join Mailing List
7 days
Organize a Trekking Group
10 days
You add estimates for how much work each new feature will take to add.

2. **Have your customer prioritize the new features**
   Tom already gave everything a priority of "20," right? But you really need him to look at the other features left to implement as well, and prioritize in relation to those.



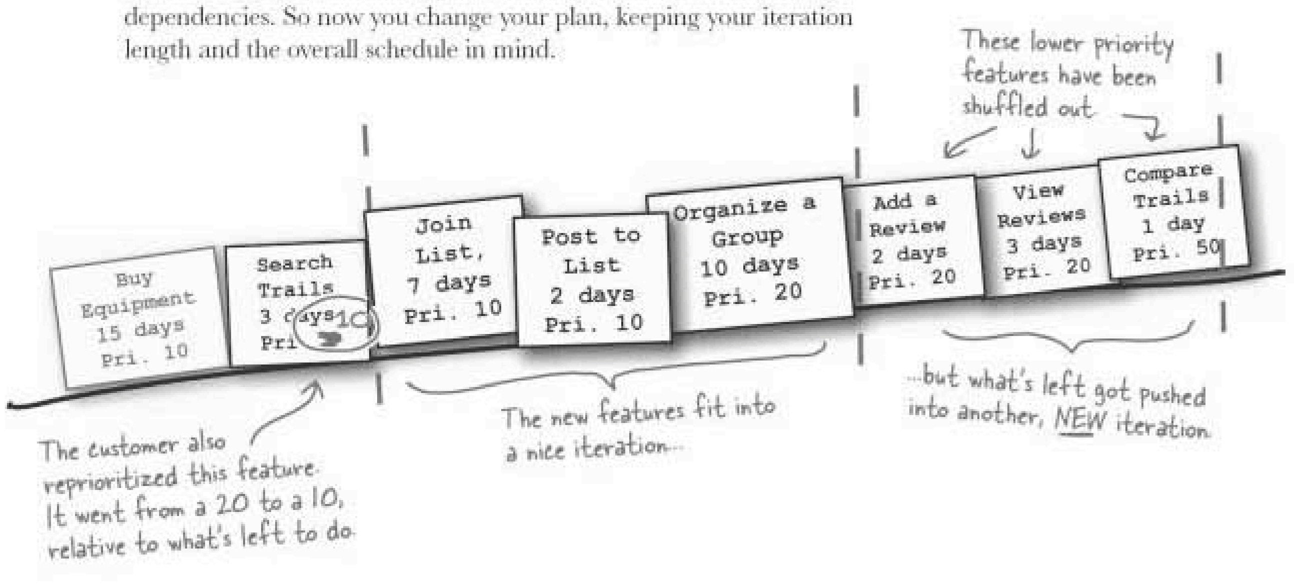
Join Mailing List
7 days
Customer Priority 10
Post to Mailing List
2 days
Customer Priority 10
Organize a Trekking Group
10 days
Customer Priority 20
Tom decided that two of these are more important than anything that's left, so they get a 10. The other feature is a 20, and can be mixed in.



A priority of 20, relative to these remaining features means we can sprinkle one more feature in anywhere before comparing trails.
Search Trails
3 days
Pri. 20
Add a Review
2 days
Pri. 20
View Reviews
3 days
Pri. 20
Compare Trails
1 day
Pri. 50

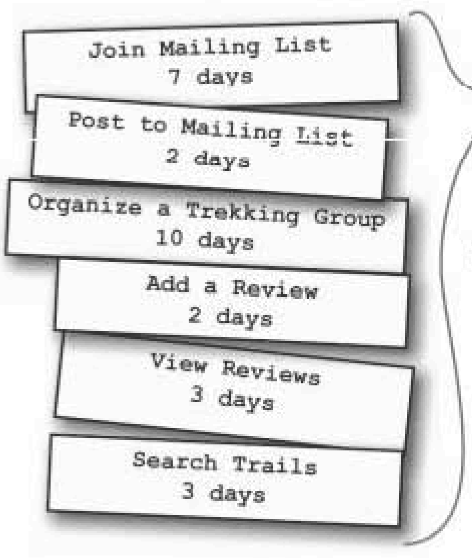
3 **Rework your iteration plan**

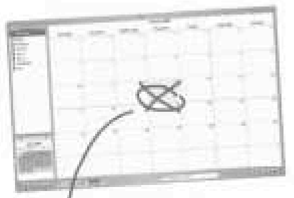
The ordering is set based on prioritization, and there aren't any dependencies. So now you change your plan, keeping your iteration length and the overall schedule in mind.


These lower priority features have been shuffled out
Buy Equipment 15 days Pri. 10
Search Trails 3 days Pri. 10
Join List, 7 days Pri. 10
Post to List 2 days Pri. 10
Organize a Group 10 days Pri. 20
Add a Review 2 days Pri. 20
View Reviews 3 days Pri. 20
Compare Trails 1 day Pri. 50
The customer also reprioritized this feature. It went from a 20 to a 10, relative to what's left to do.
The new features fit into a nice iteration...
...but what's left got pushed into another, NEW iteration.

4 **Check your project deadline**

Remember the TrailMix Conference? You need to see if the work you've got left, including the new features, still can get done in time. Otherwise, Tom's got to make some hard choices.





(Days of work left)

–

(Days left before deadline)

= Can you do it?

If this number is negative, you're in good shape.

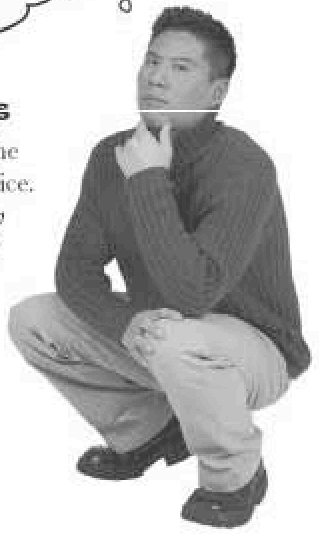
You're about to hit me with a big fancy development process, aren't you? Like if I use RUP or Quick or DRUM or whatever, I'm magically going to start producing great software, right?

A process is really just a sequence of steps

Process, particularly in software development, has gotten a bit of a bad name. A process is just a sequence of steps that you follow in order to do something—in our case, develop software. So when we've been talking about iteration, prioritization, and estimation, we've really been talking about a software development process.

Rather than being any formal set of rules about what diagrams, documentation, or even testing you should be doing (although testing is something we'd definitely recommend!), a process is really just what to do, and when to do it. And it doesn't need an acronym...it just has to work.

We don't really care what process you use, as long as it has the components that ensure you get great, quality software at the end of your development cycle.


It seems like iteration could be applied to **any** process, right?

Iteration is more than a process

Regardless of the actual steps involved in the process you choose, iteration is a best practice. It's an approach that can be applied to ***any*** process, and it gives you a better chance of delivering what is needed, on time and on budget. Whatever process you end up using, iteration should be a major part.

The right software development process for YOU is one that helps YOU develop and deliver great software, on time and on budget.

Your software isn't complete until it's been RELEASED

You added the new features, and now you and your team have finished the project on time and on schedule. At every step of the way, you've been getting feedback from the customer at the end of each iteration, incorporating that feedback, and new features, into the next iteration. Now you can deliver your software, and ***then*** you get paid.

Tom isn't talking about your software running on your machine... he cares about the software running in the real world.

Excellent. I'm already getting calls, and people love the new site. And our orders are up this week, mostly off of new customers that saw our online demo at TrailMix. Nice work.

Iteration helped you reach an achievable goal that captured what your customer needed.

there are no Dumb Questions

Q: What happens when the customer comes up with new requirements and you can't fit all the extra work into your current iteration?

A: This is when customer priority comes into play. Your customer needs to make a call as to what really needs to be done for this iteration of development. The work that cannot be done then needs to be postponed until the next iteration. We'll talk a lot more about iteration in the next several chapters.

Q: What if you don't *have* a next iteration? What if you're already on the last iteration, and then a top priority feature comes in from the customer?

A: If a crucial feature comes in late to your project and you can't fit it into the last iteration, then the first thing to do is explain to the customer why the feature won't fit. Be honest and show them your iteration plan and explain why, with the resources you have, the work threatens your ability to deliver what they need by the due date.

The best option, if your customer agrees to it, is to factor the new requirement into another iteration on the end of your project, extending the due date. You could also add more developers, or make everyone work longer hours, but be wary of trying to shoehorn the work in like this. Adding more developers or getting everyone to work longer hours will often blow your budget and rarely if ever results in the performance gains you might expect (see Chapter 3).

Tools for your Software Development Toolbox

Software Development is all about developing and delivering great software. In this chapter, you learned about several techniques to keep you on track. For a complete list of tools in the book, see Appendix ii.

Development Techniques

Iteration helps you stay on course

Plan out and balance your iterations when (not if) change occurs

Every iteration results in working software and gathers feedback from your customer every step of the way

Here are some of the key techniques you learned in this chapter...

...and some of the principles behind those techniques.

Development Principles

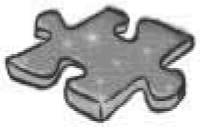
Deliver software that's needed

Deliver software on time

Deliver software on budget

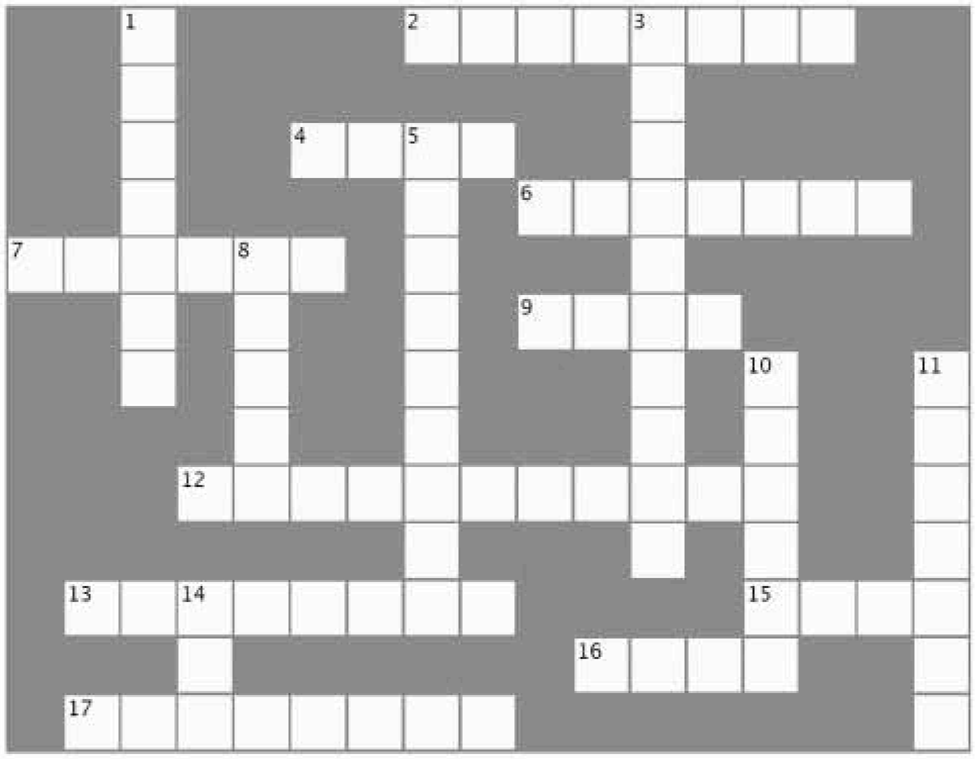
BULLET POINTS

- The **feedback** that comes out of each **iteration** is the best tool for ensuring that your software meets the needs of your customers.
- An iteration is a complete project in **miniature**.
- Successful software is not developed in a vacuum. It needs **constant feedback** from your customer using iterations.
- Good software development delivers great software, **on time** and **on budget**.
- It's always better to deliver **some** of the features **working perfectly** than all of the features that don't work properly.
- Good developers develop software; **great developers** ship software!



Software Development Cross

Let's put what you've learned to use and stretch out your left brain a bit. All of the words below are somewhere in this chapter. Good luck!



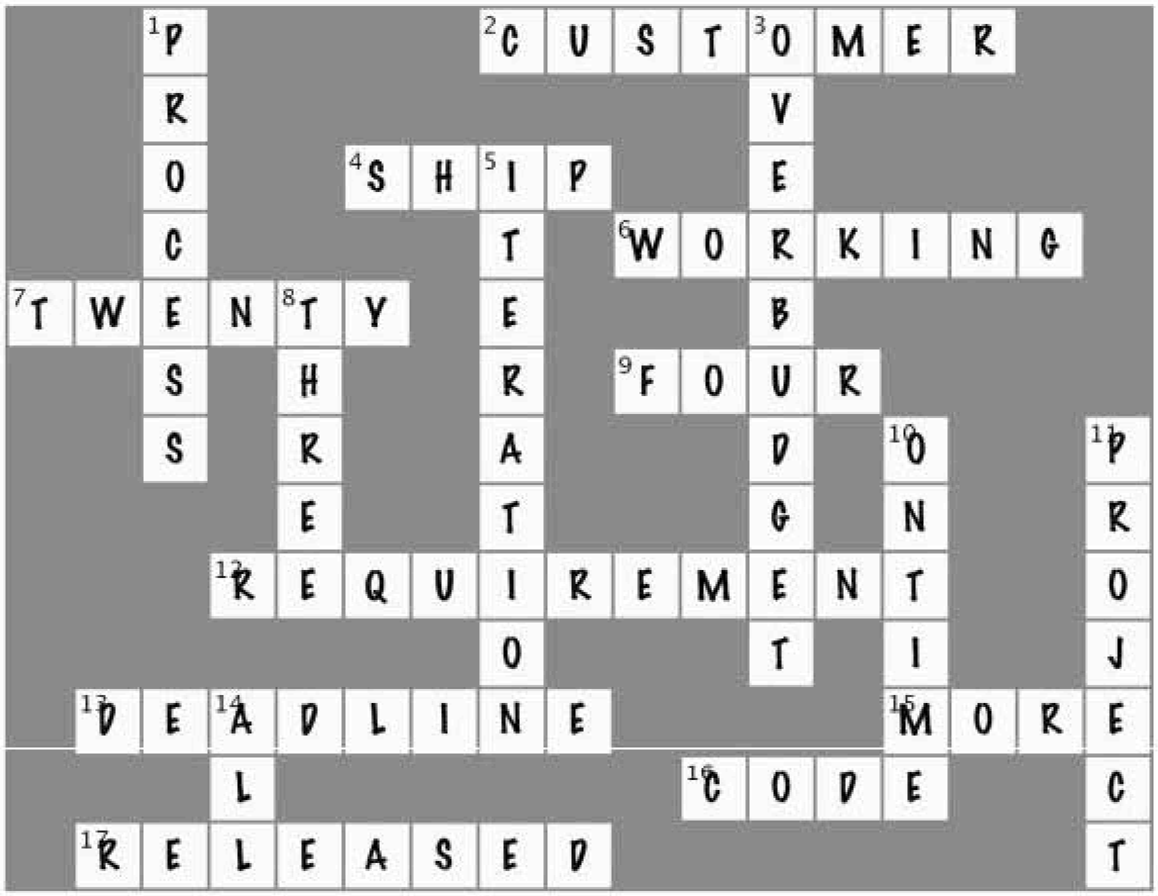
Across

2. I'm the person or company who ultimately decides if your software is worth paying for.
4. Good Developers develop, great developers ....
6. An iteration produces software that is ....
7. Aim for .... working days per iteration.
9. The number of development stages that are executed within an iteration.
12. I am one thing that your software needs to do.
13. The date that you need to deliver your final software on.
15. Iteration is .... than a process.
16. The single most important output from your development process.
17. Software isn't complete until it has been ....

Down

1. A .... is really just a sequence of steps.
3. When a project fails because it costs too much, it is ....
5. I contain every step of the software development process in micro and I result in runnable software.
8. The minimum number of iterations in a 3 month project.
10. Software that arrives when the customer needs it is ....
11. An iteration is a complete mini-.........
14. The types of software development projects where you should use iteration.

Software Development Cross Solution


1 PROCESS
2 CUSTOMER
3 OVERBUDGET
4 SHIP
5 ITERATION
6 WORKING
7 TWENTY
8 THREE
9 FOUR
10 ONTIME
11 PROJECT
12 REQUIREMENT
13 DEADLINE
14 ALL
15 MORE
16 CODE
17 RELEASED