Why Plan? 14 Why We Should Plan First, we don’t plan so we can predict the future. Business and software are changing too rapidly for prediction to be possible. Even if it was possible to predict what we needed in three years, it wouldn’t necessarily help because between now and then we need so many different things. The more obvious it is that you should do something, the more important it is to ask why. Clearly you must do some planning when tackling a serious software development project. Therefore before you start planning you have to understand why you need to it. Without that answer how can you tell if you succeed? We plan because: ✧ We need to ensure that we are always working on the most important thing we need to do. ✧ We need to coordinate with other people ✧ When unexpected events occur we need to understand the consequences for the first two. The first is the obvious reason for planning. There’s nothing more frustrating than working hard on a part of the system, only to find that it doesn’t really matter and gets scrapped in the next release of the sys- 15 tem. Time spent doing one thing is time not spent doing something else, so if that something else is important then we may fail. Say it’s two o’clock and we’re in Boston. We want to drive up to Acadia, but we’d also like to get haircuts and hit Freeport for camping gear. Last time we drove up to Acadia it took us five hours with no stops. So we see some options. If we shoot straight up to Acadia we can be there by seven. If we want to stop for dinner on the way, say an hour, and be there by eight. To get haircuts we’d need another hour, so that would be nine. Visiting Freeport is another hour. We look at what’s most important to us, if we want to be fed, equipped, not too late. and care less about our appearance we might decide to drop the haircut. A plan helps us see our options. Coordination is why everyone else wants us to plan. We get a call from our wives to meet for dinner in Bar Harbor. Since it’s two we know we can do it if we drive right up, stop in Freeport, and be there around eight. Software is full of such coordination. Marketing announcements, financial period ends, or management promises. Planning allows us to get an idea of what is reasonable. But planning is only as good as the estimates that the plans are based on, and estimates always come second to actuals. If we hit a horrible traffic jam all the planning in the world can’t make that dinner date. The real world has this horrible habit of intruding on the best laid plans. But planning still helps us since it allows us to consider the effects of the unexpected event. Leaving at two we hit bad traffic and don’t get to Portland until four-thirty. We know we usually get there after an hour, so our experience (and plan) tells us to call our friends to put dinner back to half-past eight and drop the visit to Freeport. Planning allows us both to adjust what we do and to coordinate with others. But the key value is to do this as soon as you know the effect of the event. Our wives would much rather know about our delay at four-thirty than at eight, and it would be really annoying to spend time in Freeport and only later realize that we’ve really screwed up dinner with our Cindies. (We don’t even want to contemplate the consequences of that, in comparison software failures are minor events....) 16 What we need in planning Planning is something that people do at various scales. You might plan your day's activities. The team plans out its tasks for the iteration Development and business lay out a plan for the next year. Senior managers develop plans for a large organization. When carrying out the plan you have to understand the scale in which you plan. If you are driving from Boston to Acadia, you won’t plan every curve in the road, but you will want to figure out which roads to take and when to change from one to another. You’re not going to expect to arrive to the minute, but we know there is some limit of lateness that requires the apologetic phone call. In order to carry out the coordination it’s vital to have an accurate picture of how far you are along the plan. On a road trip this is fairly straightforward. You can measure mileage, take into account the nature of the roads, and come up with a rough schedule with significant points along the way. If you are very late arriving at Portland, you can easily tell, and thus estimate the delay in reaching Bar Harbor. Software’s virtual nature again conspires against this property. With all the degrees of freedom it can be very difficult to find out whether you are 70% done or 30% done. It’s like taking a road trip where you don’t know whether you’ve gone 30 miles or 300 miles. Without any frame of reference you feel uncomfortable. If your dinner date doesn’t know how far you’ve gone, they’re uncomfortable too. Therefore any software planning technique must try to regain this visibility, so everyone involved in the project can really see how far along a project is. This means that you need clear milestones, ones that cannot be fudged, and clearly represent progress. They must also be things that everyone involved in the project, including the customer, can understand and learn to trust. Plans are about figuring out a likely course of events, and figuring the consequences of the inevitable changes. We need different plans, at different scales. Yet all the plans must be both simple to build and simple to keep up to date. Large and complex plans are not helpful because they cost too much to build and maintain. Since plans involve coordination, they must be comprehensible to everyone who is affected by the plan -- another reason for simplicity. 17 Finally they must be honest, and make it difficult to anyone, including development, to be fooled by reports of progress unrelated to reality. The Planning Trap It’s the final paragraph that gives us a hint as to why planning can be a trap. This is because there is another reason why people plan: ✧ To demonstrate they are in control of events You’ll notice our pejorative use of the third person for this reason. Controlling events is an oxymoron: you can’t control events, you can only control their consequences. And even then the amount of control you have is limited. Events cause plans to change. Once you hit that traffic jam then either dinner or Freeport are affected. You can’t just carry on with the plan and pretend everything is the same. That would be stupid. Yet we’ve seen this happen plenty of times. If things don’t go according to plan, then the planner is afraid they will be blamed. That fear causes the determination to say that the plan is still on track. They might admit to themselves that the plan is off track, but if they make the plan complicated enough they can even hide that. The key thing is to say to the outside world that everything is still going according to plan. But now the plan is diverging from reality and turning into an illusion. Worse still, the planner spends energy trying to maintain the illustion. Developers gradually lose motivation- if the plan’s an illusion than why try to follow it? The hope is that it will all sort itself out in the end. Occasionally that may happen. More often the gap between illusion and reality grows until at some point the illusion is unsustainable. At this point things get ugly. The customer is angry because they have made their own plans based on the illusion, perhaps made some promises that they can’t keep. The programmers are angry because they’ve worked hard, done as well as any programmers could do, but now are being shouted at for not doing the impossible and making the illusion real. Event happen and plans change. If things are going exactly according to plan, that’s usually a sign of trouble. The worst thing that can hap- 18 pen to a project is the divergence between the plan and reality. So don’t fall into that trap. Keep your plans honest, and expect them to always change. Rufus Your name is Bob. The date is January 3rd, 2001, and your head still aches from the recent millenial revelry. You are sitting in a conference room with several managers and a group of your peers. You are a project team leader. Your boss is there, and he has brought along all of his team leaders. His boss called the meeting. “We have a new project to develop.” Says your bosses boss. Call him BB. The points in his hair are so long that they scrape the ceiling. Your boss' points are just starting to grow, but he eagerly awaits the day when he can leave Brylcream stains on the acoustic tiles. BB describes the essence of the new market they have identified and the product they want to develop to exploit this market. "We must have this new project up and working by fourth quarter - October first." BB demands. "Nothing is of higher priority; so we are cancelling your current project.” The reaction in the room is stunned silence. Months of work are simply going to be thrown away. Slowly, a murmur of objection begins to circulate around the conference table. His points give off an evil green glow as BB meets the eyes of everyone in the room. One by one that insidious stare reduces each attendee to quivering lumps of protoChapter 4 Rufus and Rupert 20 plasm. It is clear that he will brook no discussion on this matter. Once silence has been restored, BB says: “We need to begin immediately. How long will it take you to do the analysis?" You raise your hand. Your boss tries to stop you, but his spitwad misses you and you are unaware of his efforts. "Sir, we can't tell you how long the analysis will take until we have some requirements." "The requirements document won't be ready for three or four weeks." BB says, his points vibrating with frustration. "So, pretend that you have the requirements in front of you now. How long will you require for analysis?" No one breathes. Everyone

looks around at everybody else to see if they have some idea. "If analysis takes any longer than April first, then we have a problem. Can you finish the analysis by then?" Your boss visibly gathers his courage building to the ejaculation: "We'll find a way, sir!" His points grow 3mm; and your headache increases by two Tylenol. "Good." BB Smiles. "Now, how long will it take to do the design?" "Sir," you say. Your boss visibly pales. He is clearly worried that his 3mms are at risk. "Without an analysis, it will not be possible to tell you how long design will take." BB's expression shifts beyond austere. "PRETEND, you have the analysis already!" He says, while fixing you with his vacant beady little eyes. "How long will it take you to do the design?" Two Tylenol are not going to cut it. Your boss, in a desperate attempt to save his new growth babbles: "Well, sir, with only six months left to complete the project, design had better take no longer than three months." 21 "I'm glad you agree, Smithers!" BB says, beaming. Your boss relaxes. He knows his points are secure. After awhile he starts lightly humming the Brylcream jingle. BB continues, "So, analysis will be complete by April 1st, Design will be complete by July 1st, and that gives you three months to implement the project. This meeting is an example of how well our new consensus and empowerment policies are working. Now, get out there and start working. I'll expect to see TQM plans and QIT assignments on my desk by next week. Oh, and don't forget your crossfunctional team meetings and reports will be needed for next month’s quality audit." "Forget the Tylenol." You think to yourself as you return to your cubicle. "I need bourbon." Visibly excited, your boss comes over to you and says, "Gosh, what a great meeting. I think we're really going to do some world shaking with this project." You nod in agreement, too disgusted to do anything else. "Oh," your boss continues, "I almost forgot." He hands you a thirty page document. "Remember that the SEI are coming to do an evaluation next week. This is the evaluation guide. You need to read through it, memorize it, and then shred it. It tells you how to answer any questions that the SEI auditors ask you. It also tells you what parts of the building you are allowed to take them to, and what parts to avoid. We are determined to be a CMM level 3 organization by June!" \*\*\* You and your peers start working on the analysis of the new project. This is difficult because you have no requirements. But, from the 10-minute introduction given by BB on that fateful morning, you have some idea of what the product is supposed to do. 22 Corporate process demands that you begin by creating a use case document. You and your team begin enumerating use cases and drawing oval and stick diagrams. Philosophical debates break out amongst the team. There is disagreement as to whether certain use cases should be connected with <> or <> relationships. Competing models are created, but nobody knows how to evaluate them. The debate continues, effectively paralyzing progress. After a week, somebody finds the iceberg.com website that recommends disposing entirely of <> and <> and replacing them with <

> and <>. The documents on this website,

authored by Don Sengroiux, describes a method known

as Stalwart-analysis which claims to be a step by step

method for translating use-cases into design diagrams.

More competing use-case models are created using

this new scheme; but again, nobody agrees on how to

evaluate them. And the thrashing continues.

More and more, the use-case meetings are driven by

emotion rather than reason. If it weren’t for the fact that

you don’t have requirements, you’d be pretty upset by

the lack of progress you are making.

The requirements document arrives on the 15th of February. And then again on the 20th, 25th, and every week

thereafter. Each new version contradicts the previous.

Clearly the marketing folks who are writing the requirements, empowered though they might be, are not finding

consensus.

At the same time, several new competing use-case

templates have been proposed by the various team

members. Each presents its own particularly creative way

of delaying progress. The debates rage on.

23

On March 1st, Percival Putrigence, the process proctor,

succeeds in integrating all the competing use-case forms

and templates into a single, all-encompassing form. Just

the blank form is fifteen pages long. He has managed to

include every field that appeared on all the competing

templates. He also presents a 159 page document

describing how to fill out the use-case form. All current use

cases must be rewritten according to the new standard.

You marvel to yourself that it now requires fifteen pages

of fill-in-the-blank, and essay questions, to answer the

question: “What should the system do when the user hit’s

return.”

The corporate process (authored by L. E. Ott, famed

author of “Holistic analysis: A progressive dialectic for software engineers.”) insists that you discover all primary usecases, 87% of all secondary use cases, and 36.274% of all

tertiary use cases before you can complete analysis and

enter the design phase. You have no idea what a tertiary

use-case is. So in an attempt to meet this requirement you

try to get your use-case document reviewed by the marketing department. Maybe they know what a tertiary

use-case is.

Unfortunately the marketing folks are too busy with

sales support to talk to you. Indeed, since the project

started, you have not been able to get a single meeting

with marketing. The best they have been able to do is

provide a never ending stream of changing and contradictory requirements documents.

While one team has been spinning endlessly on the

use-case document, another has been working out the

domain model. Endless variations of UML documents are

pouring out of this team. Every week the model is

reworked. The team members can’t decide on whether

24

to use <> or <> in the model. A huge

disagreement has been raging on the proper syntax and

application of OCL. Other’s in the team just got back

from a five day class on “catabolism”, and have been

producing incredibly detailed and arcane diagrams that

nobody else can fathom.

On March 27th, with one week to go before analysis is

to be complete, you have produced a sea of documents

and diagrams; but are no closer to a cogent analysis of

the problem than you were on January third.

•••

And then, a miracle happens.

•••

On Saturday, April 1st you check you email from home.

You see a memo from your boss to BB. It states unequivocally that you are done with the analysis!

You phone your boss and complain. "How could you

have told BB that we were done with the analysis?"

"Have you looked at a calendar lately?” he responds,

“It's April 1st!"

The irony of that date does not escape you. "But we

have so much more to think about. So much more to

analyze!, we haven’t even decided whether to use

<> or <>!"

"Where is your evidence that you are not done?"

inquires your boss impatiently.

"Whaaa...."

But he cuts you off. "Analysis can go on forever, it has to

be stopped at some point. And since this is the date it

was scheduled to stop, it has been stopped. Now, on

Monday I want you to gather up all existing analysis

materials and put them into a public folder. Release that

25

folder to Percival so that he can log it in the CM system by

Monday afternoon. Then get busy and start designing."

As you hang up the phone, you begin to consider the

benefits of keeping a bottle of bourbon in your bottom

desk drawer.

\*\*\*

They threw a party to celebrate the on-time completion of the analysis phase. BB gave a colon stirring speech

on empowerment. And your boss, another 3mm taller,

congratulated his team on the incredible show of unity

and teamwork. Finally, the CIO takes the stage and tells

everyone that the SEI audit went very well, and thanks

everyone for studying and shredding the evaluation

guides that were passed out. Level three now seems

assured, and will be awarded by June.

(Scuttlebut has it that managers at the level of BB and

above are to receive significant bonuses once the SEI

awards level 3.)

As the weeks flow by, you and your team work on the

design of the system. Of course you find that the analysis

that the design is supposedly based upon is flawed ... no,

useless... no, worse than useless. But when you tell your

boss that you need to go back and work some more on

the analysis to shore up its weaker sections, he simply

states: "The analysis phase is over. The only allowable

activity is design. Now get back to it."

So, you and your team hack the design as best you

can, unsure of whether the requirements have been

properly analyzed or not. Of course it really doesn't matter much since the requirements document is still thrashing with weekly revisions, and the marketing department

still refuses to meet with you.

26

The design is a nightmare. Your boss recently mis-read

a book named “The Finish-line” in which the author, Mark

DeThomaso, blithely suggested that design documents

should be taken down to code level detail.

“If we are going to be working at that level of detail,”

you ask, “why don’t we just write the code instead?”

“Because then you wouldn’t be designing, of course.

And the only allowable activity in the design phase is

design!”

“Besides,” he continues, “we have just purchased a

company wide license for Dandylion! This tools enables

“Round the Horn Engineering!” You are to transfer all

design diagrams into this tool. It will automatically generate our code for us! It will also keep the design diagrams

in sync with the code!”

Your boss hands you a brightly colored shrink-wrapped

box containing the Dandylion distribution. You accept it

numbly, and shamble off to your cubicle. Twelve hours,

eight crashes, a disk reformatting, and eight shots of 151

later, you finally have the tool installed on your server. You

consider the week your team will lose while attending

Dandylion training. Then you smile and think, “Any week

I’m not here, is a good week.”

Design diagram after design diagram is created by

your team. Dandylion makes it very hard to draw these

diagrams. There are dozens and dozens of deeply nested

dialog boxes with funny text fields and check boxes that

must all be filled in correctly. And then there’s the problem of moving classes between packages...

At first these diagram are driven from the use cases. But

the requirements are changing so often that the usecases rapidly become meaningless.

27

Debates rage about whether Visitor or Decorator

design patterns should be employed. One developer

refuses to use Visitor in any form claiming that it’s not a

properly object-oriented construct. Another refuses to

use multiple inheritance since it is the spawn of the devil.

Review meetings rapidly degenerate into debates

about the meaning of Object Orientation, the definition

of analysis vs. design, or when to use aggregation vs.

association.

Midway through the design cycle, the marketing folks

announce that they have rethought the focus of the system. Their new requirements document is completely

restructured. They have eliminated several major feature

areas, and replaced them with feature areas that they

anticipate customer surveys will show to be more appropriate.

You tell your boss that these changes mean that you

need to reanalyze and redesign much of the system. But

he says: "The analysis phase is over. The only allowable

activity is design. Now get back to it.".

You suggest that it might be better to create a simple

prototype to show to the marketing folks, and even some

potential customers. But your boss says: "The analysis

phase is over. The only allowable activity is design. Now

get back to it.".

Hack, hack, hack, hack. You try to create some kind of

a design document that might actually reflect the new

requirements documents. However, the revolution of the

requirements has not caused them to stop thrashing.

Indeed, if anything, the wild oscillations of the requirements document have only increased in frequency and

amplitude. You slog your way through them.

28

On June 15th, the Dandylion database gets corrupted.

Apparently the corruption has been progressive. Small

errors in the DB accumulated over the months into bigger

and bigger errors. Eventually the CASE tool just stopped

working. Of course the slowly encroaching corruption is

present on all the backups.

Calls to the Dandylion technical support line go unanswered for several days. Finally you receive a brief email

from Dandylion, informing you that this is a known problem, and the solution is to purchase the new version

(which they promise will be ready some time next quarter) and then re-enter all the diagrams by hand.

•••

Then, on July 1st another miracle happens! You are

done with the design!

Rather than go to your boss and complain, you stock

your middle desk drawer with some vodka.

\*\*\*

They threw a party to celebrate the on-time completion of the design phase, and their graduation to CMM

level 3. This time you find BB's speech so stirring that you

have to use the restroom before it begins.

There are new banners and plaques all over your workplace. They show pictures of eagles and mountain climbers, and they talk about teamwork and empowerment.

They read better after a few scotches. That reminds you

that you need to clear out your file cabinet to make room

for the brandy.

You and your team begin to code. But you rapidly discover that the design is lacking in some significant areas.

Actually it’s lacking any significance at all. You convene a

design session in one of the conference rooms to try to

work through some of the nastier problems. But your boss

29

catches you at it and disbands the meeting saying: "The

design phase is over. The only allowable activity is coding.

Now get back to it."

The code generated by Dandylion is really hideous. It

turns out that you and your team were using association

and aggregation the wrong way after all. All the generated code has to be edited to correct these flaws. Editing this code is extremely difficult because it has been

instrumented with ugly comment blocks that have special syntax that Dandylion needs in order to keep the diagrams in sync with the code. If you accidentally alter one

of these comments, then the diagrams will be regenerated incorrectly. It turns out that “Round the Horn Engineering” requires an awful lot of effort.

The more you try to keep the code compatible with

Dandylion, the more errors Dandylion generates. In the

end, you give up and decide to keep the diagrams up to

date manually. A second later you decide there’s no

point in keeping the diagrams up to date at all. Besides,

who has time?

Your boss hires a consultant to build tools to count the

number of lines of code that are being produced. He

puts a big thermometer graph on the wall with the number 1,000,000 on the top. Every day he extends the red

line to show how many lines have been added.

Three days after the thermometer appears on the wall,

your boss stops you in the hall. "That graph isn't growing

fast enough. We need to have a million lines done by

October 1st."

"We aren't even sh-sh-shure that the proshect will

require a m-million linezh." You blather.

"We have to have a million lines done by October 1st."

your boss reiterates. His points have grown again, and the

30

Grecian formula he uses on them creates an aura of

authority and competence. "Are you sure your comment

blocks are big enough?"

Then, in a flash of managerial insight he says: "I have it! I

want you to institute a new policy amongst the engineers.

No line of code is to be longer than 20 characters. Any

such line must be split into two or more -- preferably more.

All existing code needs to be reworked to this standard.

That'll get our line count up!"

You decide not to tell him that this will require two

unscheduled man months. You decide not to tell him

anything at all. You decide that intravenous injections of

pure ethanol are the only solution. You make the appropriate arrangements.

Hack, hack, hack, and hack. You and your team madly

code away. By August 1st your boss, frowning at the thermometer on the wall institutes a mandatory 50-hour workweek.

Hack, hack, hack, and hack. By September 1st, the

thermometer is at 1.2 million lines and your boss asks you

to write a report describing why you exceeded the coding budget by 20%. He institutes mandatory Saturdays

and demands that the project be brought back down to

a million lines. You start a campaign of re-merging lines.

Hack, hack, hack, and hack. Tempers are flaring; people are quitting; QA is raining trouble reports down on

you. Customers are demanding installation and user

manuals, salesmen are demanding advance demonstrations for special customers; the requirements document is

still thrashing, the marketing folks are complaining that

the product isn’t anything like they specified, and the

liquor store won't accept your credit card anymore.

31

Something has to give. On September 15th BB calls a

meeting.

As he enters the room, his points are emitting clouds of

steam. When he speaks, the bass overtones of his carefully manicured voice cause the pit of your stomach to roll

over. "The QA manager has told me that this project has

less than 50% of the required features implemented. He

has also informed me that the system crashes all the time,

yields wrong results, and is hideously slow. He has also

complained that he cannot keep up with the continuous

train of daily releases; each more buggy than the last!"

He stops for a few seconds, visibly trying to compose

himself. "The QA manager estimates that, at this rate of

development, we won't be able to ship the product until

December!"

Actually, you think it's more like March, but you don't

say anything.

"December!" BB roars. People duck their heads as

though he were pointing an assault rifle at them. "December is absolutely out of the question. Team leaders, I want

new estimates on my desk in the morning. I am hereby

mandating 65-hour workweeks until this project is complete. And it better be complete by Nov. 1st."

As he leaves the conference room he is heard to mutter: "Empowerment - Bad!"

\*\*\*

Your boss is bald; his points are mounted on BB's wall.

The fluorescent lights reflecting off his pate momentarily

dazzle you.

"Do you have anything to drink?" he asks. Having just

finished your last bottle of Boone's Farm, you pull a bottle

of Thunderbird from your bookshelf and pour it into his

32

coffee mug. "What's it going to take to get this project

done?" he asks.

"We need to freeze the requirements, analyze them,

design them, and then implement them." You say callously.

"By Nov. 1st?" your boss exclaims incredulously. "No way!

Just get back to coding the damned thing." He storms

out, scratching his vacant head.

A few days later you find that your boss has been transferred to the corporate research division. Turnover has

skyrocketed. Customers, informed at the last minute that

their orders cannot be fulfilled on time, have begun to

cancel their orders. Marketing is reevaluating whether or

not this product aligns with the overall goals of the company, etc., etc. Memos fly, heads roll, policies change,

and things are, overall, pretty grim.

Finally, by March, after far too many 65-hour weeks, a

very shaky version of the software is ready. In the field,

bug discovery rates are high, and the technical support

staff are at their wit's end trying to cope with the complaints and demands of the irate customers. Nobody is

happy.

In April, BB decides to buy his way out of the problem

by licensing a product produced by Rupert industries and

redistributing it. The customers are mollified, the marketing folks are smug, and you are laid off.

Rupert

Rupert Industries. Project: ~Alpha~

Your name is Robert. The date is January 3rd, 2001. The

quiet hours spent with your family this holiday have left

you refreshed and ready for work. You are sitting in a con-

33

ference room with your team of professionals. The manager of the division called the meeting.

“We have some ideas for a new project” says the division manager. Call him Russ. He is a high strung British

chap with more energy than a fusion reactor. He is ambitious and driven; but understands the value of a team.

Russ describes the essence of the new market opportunity the company has identified, and introduces you to

Jay, the marketing manager who is responsible for defining the products that will address it.

Addressing you, Jay says: “We’d like to start defining

our first product offering as soon as possible. When can

you and you team meet with me?”

You reply: “We’ll be done with the current iteration of

our project this Friday. We can spare a few hours for you

between now and then. After that, we’ll take a few people from the team and dedicate them to you. We’ll begin

hiring their replacements, and the new people for your

team immediately.”

“Great”, says Russ, “But I want you to understand that it

is critical that we have something to exhibit at the trade

show coming up this July. If we can’t be there with something significant, we’ll lose the opportunity.”

“I understand.” you reply. “I don’t yet know what it is

that you have in mind, but I’m sure we can have something by July. I just can’t tell you what that something will

be right now. In any case, you and Jay are going to have

complete control over what we developers do, so you

can rest assured that by July you’ll have the most important things that can be accomplished in that time ready

to exhibit.”

Russ nods in satisfaction. He knows how this works. Your

team has always kept him advised and allowed him to

steer their development. He has the utmost confidence

that your team will work on the most important things first;

and that they will produce a high quality product.

~ ~ ~

“So Robert,” says Jay at their first meeting, “How does

your team feel about being split up?”

“We’ll miss working with each other”, you answer, “but

some of were getting pretty tired of that last project and

34

are looking forward to a change. So, what are you guys

cooking up?”

Jay beams. “You know how much trouble our customers currently have...” And he spends a half hour or so

describing the problem and possible solution.

“OK, wait a second” you respond. “I need to be clear

about this.” And so you and Jay talk about how this system might work. Some of Jay’s ideas aren’t fully formed.

You suggest possible solutions. He likes some of them. You

continue discussing.

During the discussion, as each new topic is addressed,

Jay writes user story cards. Each card represents something that the new system has to do. The cards accumulate on the table and are spread out in front of you. Both

you and Jay point at them, and pick them up, and make

notes on them as you discuss the stories. The cards are

powerful mnemonic devices that you can use to represent complex ideas that are barely formed.

At the end of the meeting you say: “OK, I’ve got a general idea of what you want. I’m going to talk to the team

about it. I imagine there are some experiments they’ll

want to run with various database structures and presentation formats. Next time we meet, it’ll be as a group, and

we’ll start identifying the most important features of the

system.

A week later your nascent team meets with Jay. They

spread the existing user story cards out on the table and

begin to get into some of the details of the system.

The meeting is very dynamic. Jay presents the stories in

the order of their importance. There is much discussion

about each one. The developers are concerned about

keeping the stories small enough to estimate and test. So

they continually ask Jay to split one story into several

smaller stories. Jay is concerned that each story has a

clear business value and priority, so as he splits them, he

makes sure this stays true.

The stories accumulate on the table. Jay writes them,

but the developers make notes on them as needed.

Nobody tries to capture everything that is said; the cards

are not meant to capture everything; they are just

reminders of the conversation.

35

As the developers become more comfortable with the

stories, they begin writing estimates on them. These estimates are crude and budgetary, but they give Jay an

idea of what the story will cost.

At the end of the meeting, it is clear that there are many more stories that could be discussed. It is also clear that the most important stories have been addressed, and that they represent several months worth of work. Jay closes the meeting by taking the cards with him and promising to have a proposal for the first release in the morning. ~ ~ ~ The next morning you reconvene the meeting. Jay chooses five cards and places them on the table. “According to your estimates, these cards represent about one perfect team-week’s worth of work. The last iteration of the previous project managed to get one perfect team-week done in three real weeks. If we can get these five stories done in three weeks, we’ll be able to demonstrate them to Russ. That will make him feel very comfortable about our progress.” Jay is pushing it. The sheepish look on his face lets you know that he knows it too. You reply, “Jay, this is a new team, working on a new project. It’s a bit presumptuous to expect that our velocity will be the same as the previous team’s. However, I met with the team yesterday afternoon, and we all agreed that our initial velocity should, in fact, be set to one perfect-week for every three realweeks. So you’ve lucked out on this one.” “Just remember,” you continue, “that the story estimates and the story velocity are very tentative at this point. We’ll learn more when we plan the iteration, and even more when we implement it.” Jay looks over his glasses at you as if to say “Who’s the boss around here anyway.”, and then smiles and says “Yeah, don’t worry, I know the drill by now.” Jay then puts fifteen more cards on the table. He says, “If we can get all these cards done by the end of March, we can turn the system over to our beta test customers. And we’ll get good feedback from them.” 36 You reply, “OK, so we’ve got our first iteration defined; and we have the stories for the next three iterations after that. These four iterations will make our first release.” “So,” says Jay, Can you really do these five stories in the next three weeks?” “I don’t know for sure Jay,” you reply, “Let’s break them down into tasks and see what we get.” So Jay, you, and your team spend the next several hours taking each of the five stories that Jay chose for the first iteration and breaking them down into small tasks. The developers quickly realize that some of the tasks can be shared between stories, and that other tasks have commonalities that can probably be taken advantage of. It is clear that potential designs are popping into the developers’ heads. From time to time they form little discussion knots and scribble UML diagrams on some cards. Soon, the whiteboard is filled with the tasks that, once completed, will implement the five stories for this iteration. You start the sign up process by saying: “OK, let’s sign up for these tasks.” “I’ll take the initial database generation.” says Pete, “That’s what I did on the last project, and this doesn’t look very different. I estimate it at two of my perfect mandays.” “OK, well then I’ll take the login screen.” says Joe. “Aw darn,” says Elmo, the junior member of the team, “I’ve never done a GUI, and I kinda wanted to try that one.” “Ah the impatience of youth.” Joe says sagely, with a wink in your direction, “You can assist me with it, young Jedi.” To Jay: “I think it’ll take me about three of my perfect man-days.” One by one the developers sign up for tasks and estimate them in terms of their own perfect man-days. Both you and Jay know that it is best to let the developers volunteer for tasks, than it is to assign the tasks to them. You also know full well that you daren’t challenge any of the developer’s estimates. You know these guys, and you trust them. You know they are going to do the very best they can. 37 The developers know that they can’t sign up for more perfect man-days than they finished in the last iteration they worked on. Once each developer has filled his schedule for the iteration, they stop signing up for tasks. Eventually, all the developers have stopped signing up for tasks. But, of course, there are still tasks left on the board. “I was worried that might happen.” you say, “OK, there’s only one thing to do, Jay. We’ve got too much to do in this iteration. What stories or tasks can we remove.” Jay sighs. He knows that this is the only option. Working overtime at the beginning of a project is insane; and projects where he’s tried it have not fared well. So Jay starts to remove the least important functionality. “Well, we really don’t need the login screen just yet. We can simply start the system in the logged in state.” “Rats!” cries Elmo. “I really wanted to do that.” “Patience, Grasshopper.” says Joe. “Those who wait for the bees to leave the hive, will not have lips too swollen to relish the honey.” Elmo looks confused. Everyone looks confused. “So...”, Jay continues, “I think we can also do away with...” And so, bit by bit the list of tasks shrinks. Developers who lose a task, sign up for one of the remaining ones. The negotiation is not painless. Several times Jay exhibits obvious frustration and impatience. Once, when tensions are especially high, Elmo volunteers to “Work extra hard to make up some of the missing time.” You are about to correct him when, fortunately, Joe looks him in the eye and says, “When once you proceed down the dark path, forever will it dominate your destiny.” In the end, an iteration acceptable to Jay is reached. It’s not what Jay wanted. Indeed, it is significantly less. But it’s something the team feels that they can achieve in the next three weeks. And, after all, it still addresses the most important things that Jay wanted in the iteration. “So, Jay,” you say when thing had quieted down a bit. “When can we expect acceptance tests from you?” 38 Jay sighs. This is the other side of the coin. For every story the development team implements, Jay must supply a suite of acceptance tests that prove that they work. And the team needs these long before the end of the iteration, since they will certainly point out differences in the way Jay and the developers imagine the system’s behavior. “I’ll get you some example test scripts today.” Jay promises. “I’ll add to them every day after that. You’ll have the entire suite by the middle of the iteration.” ~~~ The iteration begins on Monday morning with a flurry of CRC sessions. By mid-morning all the developers have assembled into pairs and are rapidly coding away. “And now, my young apprentice,” Joe says to Elmo, “you shall learn the mysteries of Test First Design!”. “Wow, that sounds pretty rad.” Elmo replies. “How do you do it?” Joe beams. It’s clear that he has been anticipating this moment. “Laddy-buck, what does the code do right now?” “Huh?”, replied Elmo, “It doesn’t do anything at all, there is no code.” “So, consider our task, can you think of something the code should do?” “Sure.” Elmo said with youthful surety, “First, it should connect to the database.” “And thereupon, what must needs be required to connecteth the database?” “You sure talk wierd.” laughed Elmo. “I think we’d have to get the database object from some registry and call the Connect() method. “Ah. Astute young wizard. Thou perceivest correctly that we requireth an object within which we can cacheth the database object.” “Is ‘cacheth’ really a word?” “It is when I say it! So, what test can we write that we know the database registry should pass?” Elmo sighs. he knows he’ll just have to play along. “We should be able to create a database object and pass it to the registry in a Store() method. And then we should 39 be able to pull it out of the registry with a Get() method and make sure it’s the same object.” “Oh, well said, my pre-pubescent sprite!” “Hay!” “So, now, let’s write a test function that proves your case.” “But shouldn’t we write the database object and registry object first?” “Ah, you’ve much to learn my young impatient one. Just write the test first.” “But it won’t even compile!” “Are you sure? What if it did?” “Uh...” “Just write the test, Elmo. Trust me.” And so Joe, Elmo, and all the other developers began to code their tasks, one test case at a time. The room in which they worked was a-buzz with the conversations between the pairs. The murmur was punctuated by an occasional high-five when a pair managed to finish a task or a difficult test case. As development proceeded, the developers changed partners once or twice a day. Each developer got to see what all the others were doing, and so knowledge of the code spread generally throughout the team. Whenever a pair finished something significant; whether a whole task, or just an important part of a task, they integrated what they had with the rest of the system. Thus, the code base grew daily, and integration difficulties were minimized. The developers communicated with Jay on a daily basis. They’d go to him whenever they had a question about the functionality of the system, or the interpretation of an acceptance test case. Jay, good as his word, supplied the team with a steady stream of acceptance test scripts. The team read these carefully and thereby gained a much better understanding of what Jay expected the system to do. By the beginning of the second week, there was enough functionality to demonstrate to Jay. Jay watched eagerly as the demonstration passed test-case after testcase. 40 “This is really cool.” Jay said as the demonstration finally ended. “But this doesn’t seem like one third of the tasks. Is your velocity slower than anticipated?” You grimace. You’d been waiting for a good time to mention this to Jay; but now Jay was forcing the issue. “Yes, unfortunately we are going slower than we had expected. The new application server we are using is turning out to be a pain to configure. Also it takes forever to reboot; and we have to reboot it whenever we

make even the slightest change to its configuration.” Jay eyes you with suspicion. The stress of last Monday’s negotiations had still not entirely dissipated. He says: “And what does this mean to our schedule? We can’t slip it again, we just can’t. Russ will have a fit! He’ll haul us all into the woodshed and ream us some new ones.” You look Jay right in the eyes. There’s no pleasant way to give someone news like this. So you just blurt out, ”Look, if things keep going like their going, then we’re not going to be done with everything by next Friday. Now it’s possible that we’ll figure out a way to go faster. But, frankly, I wouldn’t depend upon that. You should start thinking about one or two tasks that could be removed from the iteration without ruining the demonstration for Russ. Come hell or high water we are going to give that demonstration on Friday, and I don’t think you want us to choose which tasks to omit.” “Aw for -- goodness sakes!” Jay barely manages to stifle yelling that last word as he stalks away shaking his head. Not for the first time you say to yourself: “Nobody ever promised me project management would be easy.” You are pretty sure it won’t be the last time either. ~~~ Actually, things went a bit better than you had hoped. The team did, in fact, have to drop one task from the iteration; but Jay had chosen wisely, and the demonstration for Russ went without a hitch. Russ was not impressed with the progress, but neither was he dismayed. He simply said, “This is pretty good. But remember, we have to be able to demonstrate this system at the trade show in July; and at this rate it doesn’t look like you’ll have all that much to show.” 41 Jay, whose attitude had improved dramatically with the completion of the iteration, responded to Russ by saying: “Russ, this team is working hard, and well. When July comes around I am confident that we’ll have something significant to demonstrate. It won’t be everything, and some of it may be smoke and mirrors, but we’ll have something.” Painful though the last iteration was, it had calibrated your velocity numbers. The next iteration went much better. Not because your team got more done than in the last iteration, but simply because they didn’t have to remove any tasks or stories in the middle of the iteration. By the start of the fourth iteration, a natural rhythm has been established. Jay, you, and the team know exactly what to expect from each other. The team is running hard, but the pace is sustainable. You are confident that the team can keep up this pace for a year or more. The number of surprises in the schedule diminishes to near zero; however the number of surprises in the requirements does not. Jay and Russ frequently look over the growing system and make recommendations or changes to the existing functionality. But all parties realize that these changes take time, and must be scheduled. So the changes do not cause anyone’s expectations to be violated. In March there is a major demonstration of the system to the board of directors. The system is very limited, and is not yet in a form good enough to take to the trade show; but progress is steady, and the board is reasonably impressed. The second release goes even smoother than the first. By now the team has figured out a way to automate Jay’s acceptance test scripts. They have also refactored the design of the system to the point where it is really easy to add new features and change old ones. The second release was done by the end of June, and was taken to the trade show. It had less in it than Jay and Russ would have liked; but it did demonstrate the most important features of the system. Though customers at the trade show noticed that certain features were missing, overall they were very impressed. You, Russ, and Jay 42 all returned from the trade show with smiles on your faces. You all felt as though this project was a winner. Indeed, many months later you are contacted by Rufus Inc. They had been working on a system like this for their internal operations. They have cancelled the development of that system after a death-march project; and are negotiating to license your technology for their environment. Indeed, things are looking up! Courage! What makes a King out of a slave? Courage! What makes the flag on the mast to wave? Courage! What makes the elephant charge his tusk, in the misty mist or the dusky dusk? What makes the muskrat guard his musk? Courage! What makes the sphinx the seventh wonder? Courage! What makes the dawn come up like thunder? Courage! What makes the Hottentot so hot? What puts the "ape" in apricot? What have they got that I ain't got? -- Cowardly Lion Why do we need a software process? For the same reason that we need laws, governments, and taxes: Fear. The American Declaration of Independence says: That among these [rights] are life, liberty, and the pursuit of happiness. That to secure these rights, governments are instituted among men, deriving their just powers from the consent of the governed.1 Though the profundity of these words may distract us, consider the word “secure”. We institute governments because we are afraid of losing our rights. By the same token, we institute software processes because we are afraid. Customers are afraid that: 1. Microsoft word complains about the grammar… Chapter 5 Fear 44 ✧ They won’t get what they asked for. ✧ They’ll ask for the wrong thing. ✧ They’ll pay too much for too little. ✧ They must surrender control of their career to techies who don’t care. ✧ They won’t ever see a meaningful plan. ✧ The plans they do see will be fairy-tales. ✧ They won’t know what’s going on. ✧ They’ll be held to their first decisions and won’t be able to react to changes in the business. ✧ No one will tell them the truth. Developers are afraid too. They fear that: ✧ They will be told to do more than they know how to do. ✧ They will be told to do things that don’t make sense. ✧ They are too stupid. ✧ They are falling behind technically. ✧ They will be given responsibility without authority. ✧ They won’t be given clear definitions of what needs to be done. ✧ That they’ll have to sacrifice quality for deadlines. ✧ That they’ll have to solve hard problems without help. ✧ That they won’t have enough time to succeed. Unacknowledged fear is the source of all engineering failure. If these fears are not put on the table and dealt with, then developers and customer each try to protect themselves by building walls. They refuse to share critical information: “If I tell the engineers about this, they’ll spend months trying to figure it out instead of doing what I need.” “If I tell the customer how quickly I got this done, he’ll expect me to do everything that fast.” 45 They exaggerate, tell half-truths, lie, cover-up, and work at cross purposes. They build huge useless political and procedural structures aimed at protection instead of success. In order to be successful, a development process must be instituted among customers and developers that secures certain inalienable rights. Among these are: The Customer Bill of Rights. ✧ You have the right to an overall plan, to know what can be accomplished, when, and at what cost. ✧ You have the right to get the most possible value out of every programming week. ✧ You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify. ✧ You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs. ✧ You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system reflecting investment to date. Programmer Bill of Rights ✧ You have the right to know what is needed, with clear declarations of priority. ✧ You have the right to produce quality work at all times. ✧ You have the right to ask for and receive help from peers, superiors, and customers. ✧ You have the right to make, and update your own estimates. ✧ You have the right to accept your responsibilities instead of having them assigned to you. If we are going to develop well, we must create a culture that makes it possible for programmers and customer to acknowledge their fears and accept their rights and responsibilities. Without such guarantees, we cannot be courageous. We huddle in fear behind fortress walls, building them ever stronger, adding ever more weight to the develop- 46 ment processes we have adopted. We continually add cannonades and battlements, documents and reviews, procedures and signoffs, moats with crocodiles, torture chambers, and huge pots of boiling oil. But when our fears are acknowledged and our rights are accepted, then we can be courageous. We can set goals that are hard to reach, and collaborate to make those goals. We can tear down the structures that we built out of fear and that impede us. We will have the courage to do only what is necessary and no more, to spend our time on what’s important, rather than on protecting ourselves. The driving story featured prominently in Embrace Change, but it is central to XP, so we repeat it here. If you read EC, you’ll only want to read this chapter to see if we’ve somehow managed to make the story a bit more dramatic. It was a beautiful sunny day. Kent and his mom were driving along a straight stretch of I-5 near Chico. He was about 12 years old. “It’s about time you learn how to drive,” said Mom. “Really?” Excitement bubbled in Kent’s chest. Chapter 6 Driving Software 48 “Yes. Now, what I want you to do is get the car right in between the lines and pointed absolutely straight,” said Mom. “I can do that.” Kent very carefully lines up the star on the beige Mercedes 240D dead straight to the horizon. His eyebrows raise a little at just how easy this driving thing really is. After

a moment, his eyes drift to a roadside sign. \*\*\*ggggrrrrrrccccchhhh\*\*\* (hey, you try to write down the sound combining wheels on gravel and a pre-adolescent yelp). Kent’s mouth goes dry, his heart pounds. “Okay,” says Mom, concealing a smile, “that’s not how you drive a car. Driving a car is not about getting the car pointed in the right direction. Driving a car is about constantly making little corrections. You drift a little this way, you steer a little that way. This way, that way, as long as you are driving.” You don’t drive software development by getting your project pointed in the right direction (The Plan). You drive software development by seeing that you are drifting a little this way and steering a little that way. This way, that way, as long as you develop the software. One very vocal opponent of XP once used the phrase: “Ready…Fire…Aim!” The intent was clearly pejorative. How can you hit a target unless you aim first? The point, however, is that we are not trying to hit a target. Instead, we are trying to maximize the benefit of a process. The driving metaphor helps us once again. Your first act as you get into the car is not turning the wheel so that it points towards your destination. Your first act is usually to turn on the ignition. Indeed, the initial direction of motion has little to do with your destination, and much more to do with your local circumstances. You might want to back out of your garage before heading for Peoria. Though you probably have a destination in mind, and probably also have a route planned, that route and destination are always subject to change. The radio may warn you of heavy traffic, causing you to change your route. Your spouse may call on the cell phone and ask you to pick up some milk, causing you to modify your destination. Software development is a process. It can go well, or it can go badly. To keep it going well we must continually direct it. To direct it we must frequently assess the direction it is going, compare this to the direction 49 we want it to go, and then make careful adjustments. Thus, good project management can be characterized by: “Ready… Fire… Aim… Aim… Aim… Aim… Aim…”