CSE203

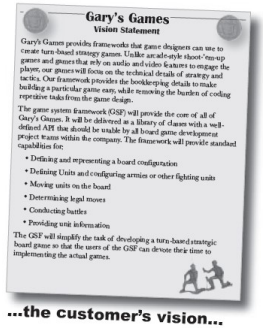
Principles of Software Design and Development

07. Architecture

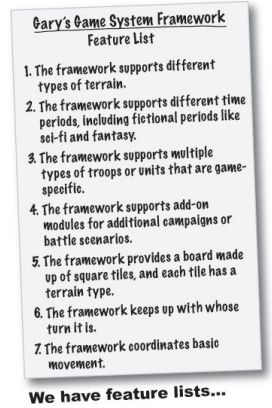
Bringing Order to Chaos

1

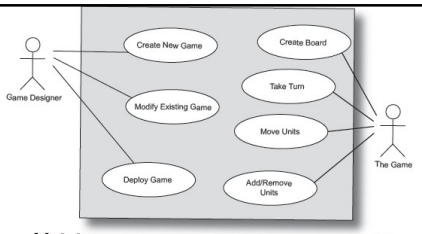
Feeling a little overwhelmed?



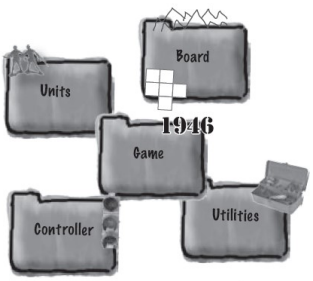
...the customer's vision...



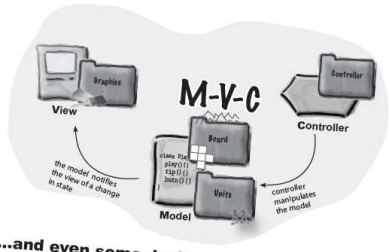
We have feature lists...



...high-level views of what we need to build...



...individual modules to code...



...and even some design patterns to apply.

OK, even if I do know how to handle all these individual pieces, where the heck am I supposed to start? Can you at least tell me what to do FIRST?

Remember our programmer friend? Here's how we left him

2

We need an architecture

We already knew this... architecture helps us design these big systems.

Architecture is your design structure, and highlights the most important parts of your app, and the relationships between those parts.

- figuring out individual pieces is not enough
- need to know
  - how those pieces fit together,
  - which ones are more important than others;
- then you'll know what you should work on *first*.

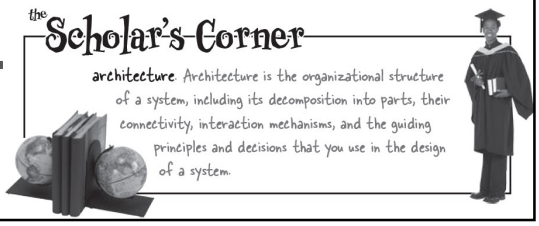
Now this is what we need... how do we figure out what's most important, so we can build those parts of our app first?

Our use case diagram was the start of this, but it's still pretty unclear how all the modules interact.

All this is particularly important when you're working with other programmers... you have to all understand the same architecture.

the Scholar's Corner

architecture. Architecture is the organizational structure of a system, including its decomposition into parts, their connectivity, interaction mechanisms, and the guiding principles and decisions that you use in the design of a system.



3

Architecture takes a big chaotic mess...

Ever get this feeling? You've got lots of important diagrams and plans, but everything is just a huge mess.

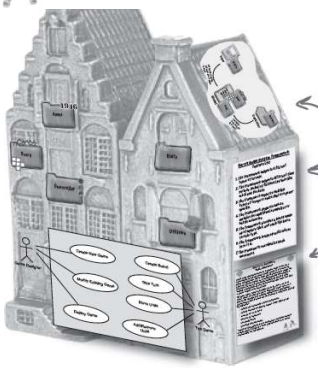
I have **no clue** what to do with all of this stuff.

These lists and patterns should help, but it's hard to know how it all fits together.

Wow... **now** I see how it all fits together!

...and helps us turn it into a well-ordered application

This is what we want... to use all the information we have to create a nice, well-constructed application.



All the diagrams and patterns are used to build the customer exactly what they want, all within a flexible, reusable design.

4

Great software in 3 easy steps

1. Make sure your software does what the customer wants it to do.

This step focuses on the customer. Make sure the app does what it's supposed to do FIRST. This is where getting good requirements and doing some analysis comes in.

2. Apply basic OO principles to add flexibility.

Once your software works, you can look for any duplicate code that might have slipped in, and make sure you're using good OO programming techniques.

Got a good object-oriented app that does what it should? It's time to apply patterns and principles to make sure your software is ready to use for years to come.

3. Strive for a maintainable, reusable design.

12

We know that these three steps will help us tackle each of these individual pieces of the game system framework.

These three steps apply when you're working on really big applications, too. So we need to start with what the customer wants an app to do, before we get into details about the actual design of the app.

Really BIG application

5

Let's start with functionality

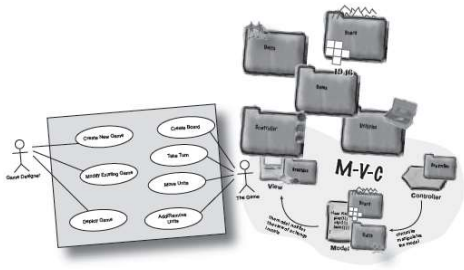
All of these features are about functionality... they focus on what the system has to do, not on what principles or patterns you use to build the system.

But which of these are the most important?

Gary's Game System Framework
Feature List

1. The framework supports different types of terrain.
2. The framework supports different time periods, including fictional periods like sci-fi and fantasy.
3. The framework supports multiple types of troops or units that are game-specific.
4. The framework supports add-on modules for additional campaigns or battle scenarios.
5. The framework provides a board made up of square tiles, and each tile has a terrain type.
6. The framework keeps up with whose turn it is.
7. The framework coordinates basic movement.

We'll come back to these other diagrams and patterns later... but right now, we're focusing solely on the functionality of the system.



6

Sharpen your pencil

What do YOU think are the most important features?

Even though our feature list has only seven things on it, there's a lot of work in those seven features. It's your job to figure out which features you think are the most important, and then in what order you'd work on those things.

Write down the 4 things you'd do first, in order, in these blanks.

1. _____
2. _____
3. _____
4. _____

Gary's Game System Framework
Feature List

1. The framework supports different types of terrain.
2. The framework supports different time periods, including fictional periods like sci-fi and fantasy.
3. The framework supports multiple types of troops or units that are game-specific.
4. The framework supports add-on modules for additional campaigns or battle scenarios.
5. The framework provides a board made up of square tiles, and each tile has a terrain type.
6. The framework keeps up with whose turn it is.
7. The framework coordinates basic movement.

You've got to handle all of these features, but it's up to you to figure out the order you should tackle them in.

7

The things in your application that are **really important** are **architecturally significant**, and you should focus on them **FIRST.**

Wait a second... if architecture is about the **relationships** between the parts of an application, why are we talking about the individual parts? Shouldn't we be talking about how the parts work together?

You gotta start somewhere!

It's hard to talk about the relationships between parts of a system if you don't have any of the parts themselves.

So architecture isn't just about the relationships between parts of your app; it's also about figuring out which parts are the most important, so you can start building those parts first.

8

The three Qs of architecture

The essence of a system is what that system is at its most basic level.

1. Is it part of the essence of the system?

- Is the feature really core (essence) to what a system actually is?
- Can you imagine the system without that feature?

Note from marketing: suggest replacing profanity with "heck."

2. What the [redacted] does it mean?

If you're not sure what the description of a particular feature really means, it's probably pretty important that you pay attention to that feature.

3. How the "heck" do I do it?

Focus your attention early on features that seem really hard to implement, or are totally new programming tasks for you.

9

BE the Architect

To the right, you'll find the feature list we figured out in the last chapter. Your job is to play like you're the architect, and figure out what's architecturally significant by using the three Qs of architecture we just talked about.

We decided that the board was core to the game... without a board, there really isn't a game!

WHAT'S SIGNIFICANT?	WHY?
The board for the game	Q1
Game-specific units	Q1, Q2
The framework coordinates basic movement	Q3 (and may be Q2)

We thought that troops were essential to the game... and we're not sure what "game-specific" might really mean. So two Qs applied here.

Write down which of the three Qs applies (you can write more than one, if you need to)

This seems a little vague, but it's not something we're sure about how to do. Definitely worth spending some time up front figuring out what this means, and what we need to do.

Gary's Game System Framework
Feature List

1. The framework supports different types of terrain.
2. The framework supports different time periods, including fictional periods like sci-fi and fantasy.
3. The framework supports multiple types of troops or units that are game-specific.
4. The framework supports add-on modules for additional campaigns or battle scenarios.
5. The framework provides a board made up of square tiles, and each tile has a terrain type.
6. The framework keeps up with whose turn it is.
7. The framework coordinates basic movement.

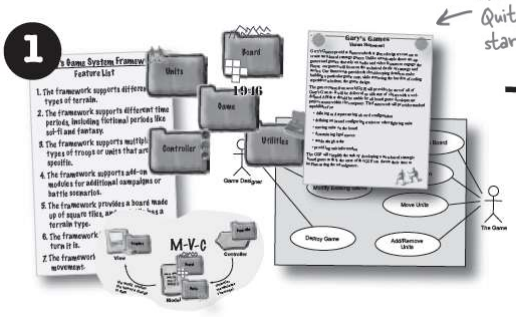
10

We've got a lot less chaos now...

Remember all this? Quite a mess when we started out...

But then we focused on making the system do what it is supposed to do.

1



2

Gary's Game System Framework
Feature List

1. The framework supports different types of terrain.
2. The framework supports different time periods, including fictional periods like sci-fi and fantasy.
3. The framework supports multiple types of troops or units that are game-specific.
4. The framework supports add-on modules for additional campaigns or battle scenarios.
5. The framework provides a board made up of square tiles, and each tile has a terrain type.
6. The framework keeps up with whose turn it is.
7. The framework coordinates basic movement.

3

Gary's Game System Framework
KEY Features

1. The board for the game—essence of the system
2. Game-specific units—essence, and what does this mean?
3. Coordinating movement—what is it, and how do we do it?

Finally, we've narrowed that down to just a few key features to focus on.

...but there's still plenty left to do which one should you work on first ?

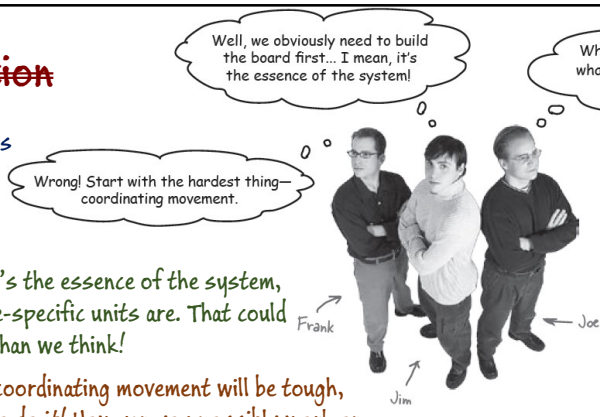
11

Cubicle Argument ~~Conversation~~

Jim: What in the world are you guys thinking? What good is it starting With anything that isn't the essence of the system?

Joe: That's ridiculous. Even if that's the essence of the system, you've got to figure out what game-specific units are. That could take weeks to write if it's harder than we think!

Frank: Maybe... but we *know* that coordinating movement will be tough, because we don't have a clue how to do it! How can you possibly work on anything else when you *know* the movement deal is going to be difficult?



- ❑ Jim (build the board)
- ❑ Joe (build game-specific units)
- ❑ Frank (build movement engine)

? So who do you think is right

Joe: But the game-specific units might be difficult, too! We just don't know, and that's my point. We've got to figure out the parts of the system we don't know anything about, or they could be real trouble!

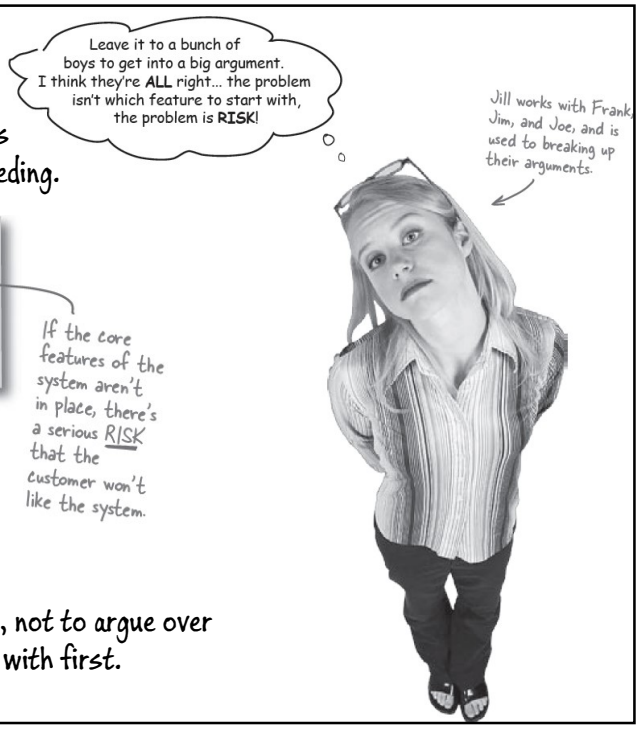
Jim: You guys go on and write movement engines and deal with units. Me, I'm gonna write a board, because... well... something tells me Gary will want to see a *board* for his *board game system*. And I'm not about to leave the board for later... I'm taking it on first.

Frank: You're both nuts. While you're putting off the hard tasks, I'm gonna make sure the things that I don't have any real idea about are taken care of, right away.

12

The reason that these features are architecturally significant is that they all introduce RISK

It doesn't matter which one you start with—as long as you are working towards reducing the RISKS in succeeding.



Gary's Game System Framework
KEY Features

1. The board for the game—essence of the system
2. Game-specific units—essence, and what does this mean?
3. Coordinating movement—what is it, and how do we do it?

If the core features of the system aren't in place, there's a serious RISK that the customer won't like the system.

Since we don't know what this means, it could be a ton of work, and that's a RISK in meeting schedules and deadlines.

This is something we're not sure how to do, so there's a RISK that we won't figure it out, or it will take a really long time.

The point here is to REDUCE RISK, not to argue over which key feature you should start with first.

13

Architecture Puzzle

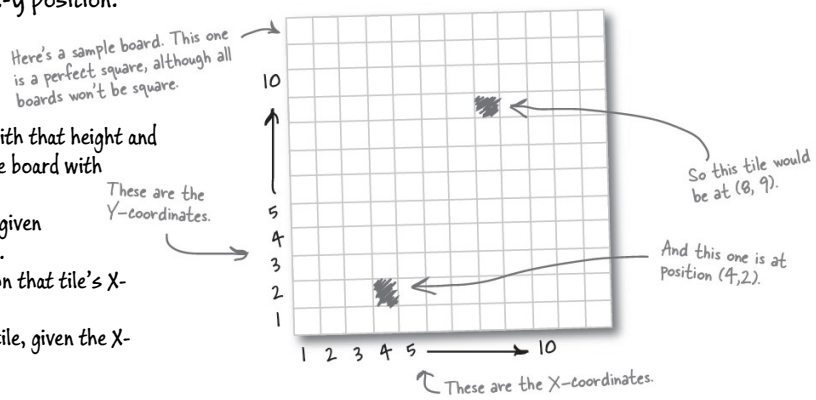
The problem:

You need a **Board** base type that game designers can use to create new games. The board's height and width are supplied by the game designers for their games. Additionally, the board can return the tile at a given position, add units to a tile, and return all the units at a given X-Y position.

Here are some detailed requirements that we got from Gary and his design team.

Your task:

1. Create a new class called **Board.java**.
2. Add a constructor to **Board** that takes in a width and height, and creates a new board with that height and width. The constructor also needs to fill the board with square tiles, one at each X-Y position.
3. Write a method that will return the tile at a given position, given that tile's X- and Y-position.
4. Write methods to add units to a tile based on that tile's X- and Y-position.
5. Write a method to return all the units on a tile, given the X- and Y-position of the tile.



14

Scenario Scramble

This is the risk we're trying to reduce or eliminate using a scenario.

Write a scenario for the Board interface you just coded.

You've coded a Board interface based on a few requirements we gave you, but it's your job to figure out if we forgot anything—before Gary sees your work and finds a mistake. Your job is to take the fragments of a scenario from the bottom, and put them into an order that makes sense on the bulletin board. The scenario should run through a realistic portion of a game. When you're done, see if you left out anything on the Board interface, and if you did, add the missing functionality into your code. You may not need all the scenario fragments; good luck!

Player 1 moves subs to (2, 2).

Player 1 moves artillery onto (4, 5).

Game requests terrain at (4, 5).

Game designer creates board with a height and width.

Player 1 battles Player 2.

Player 1's units lose the battle.

Game requests units from (4, 5).

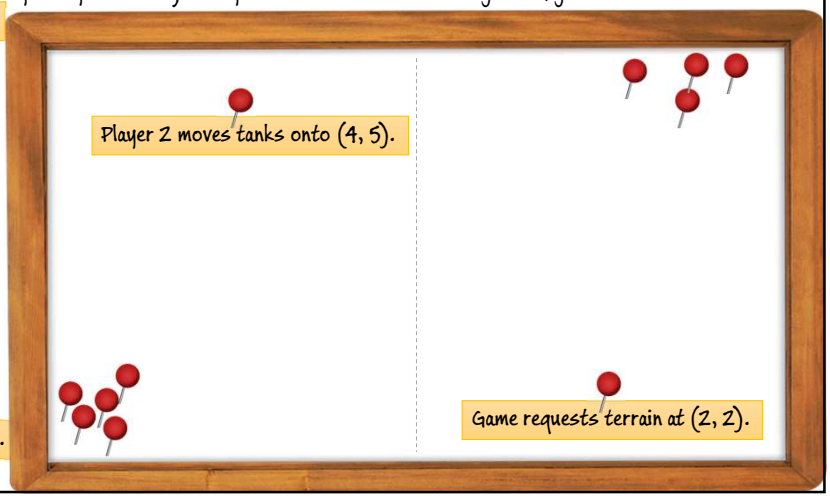
Player 1's units are removed from (4, 5).

Player 2's units win the battle.

Game designer supplies height and width.

Game designer creates a new board.

Player 2 moves army onto (4, 5).



15

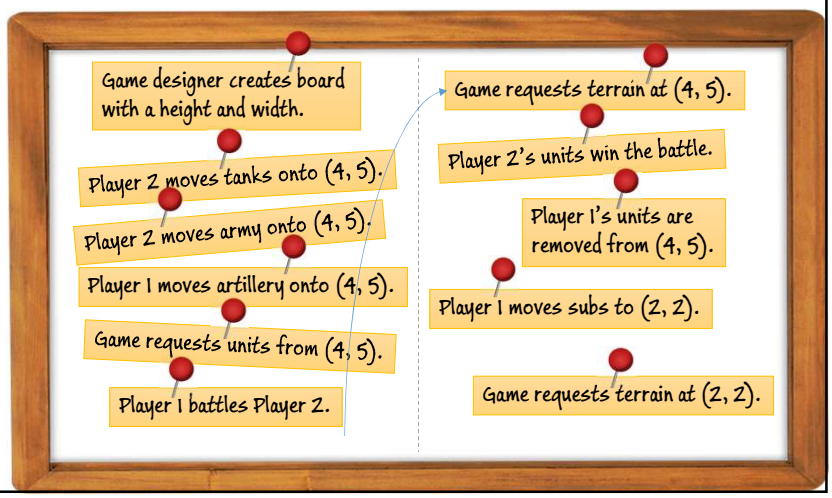
Scenario Scramble Solution

Write a scenario for the Board interface you just coded.

Player 1's units lose the battle.

Game designer supplies height and width.

Game designer creates a new board.



16

Architecture Puzzle Solution

```
package cse.gsf.board;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import bim309.gsf.unit.Unit;

public class Board {
  private int width, height;
  private List tiles;
  public Board(int width, int height) {
    this.width = width;
    this.height = height;
    initialize();
  }
  private void initialize() {
    tiles = new ArrayList(width);
    for (int i=0; i<width; i++) {
      tiles.add(i, new ArrayList(height));
      for (int j=0; j<height; j++) {
        ((ArrayList)tiles.get(i)).add(j, new Tile());
      }
    }
  }
```

We put the Board class in a board-specific package. This lines up with the modules we decided on

Here's a class we'll create in a minute, since we need it to finish up Board.

This constructor was laid out in the requirements. It takes the width and height in, and then calls initialize() to set up the board.

We represented the grid on the board as an array of arrays, using width and height as the dimensions.

At each coordinate, we add a new instance of Tile. We'll have to write that class to make Board work as well... check the next page for how we defined Tile.


Board.java

17

Architecture Puzzle Solution

These methods are pretty self-explanatory, and were part of the requirements

```
public Tile getTile(int x, int y) {
  return (Tile)((ArrayList)tiles.get(x-1)).get(y-1);
}

public void addUnit(Unit unit, int x, int y) {
  Tile tile = getTile(x, y);
  tile.addUnit(unit);
}

public void removeUnit(Unit unit, int x, int y) {
  Tile tile = getTile(x, y);
  tile.removeUnit(unit);
}

public void removeUnits(int x, int y) {
  Tile tile = getTile(x, y);
  tile.removeUnits();
}

public List getUnits(int x, int y) {
  return getTile(x, y).getUnits();
}
}
```

We decided to let the Tile class handle these operations, and just delegate adding and removing units to that class.

You should have figured out that we need a way to remove units from the scenario exercise. This is what was missing in our original requirements.

Here's another place where we delegate to the Tile class. Since a tile stores the units on it, it's really the tile's job to handle retrieving those units.


Board.java

18

The Tile and Unit classes

```
package cse.gsf.unit;
public class Unit {
  public Unit() { }
}
```

Unit.java

We made Unit as absolutely simple as possible. There are lots of details to be added later, but we don't need those details to make Board work right now.

Tile is in the same package as the Board class... they're tightly related.

```
package cse.gsf.board;
import java.util.LinkedList;
import java.util.List;
import cse.gsf.unit.Unit;

public class Tile {
  private List units;

  public Tile() {
    units = new LinkedList();
  }

  protected void addUnit(Unit unit) {
    units.add(unit);
  }

  protected void removeUnit(Unit unit) {
    units.remove(unit);
  }
}
```

Tile has a list of the units on it at any given time.

These are the methods that Board uses to manipulate units. They're protected, so only classes in headfirst.gsf.board can access them.

Tile.java

Keep the right focus

You don't need to worry about everything that **Tile** and **Unit** will eventually need to do. Your focus is on making **Board** and its key features work, not on completing **Tile** or **Unit**. That's why we left **Unit** so bare, and added only a few methods to **Tile**.

19

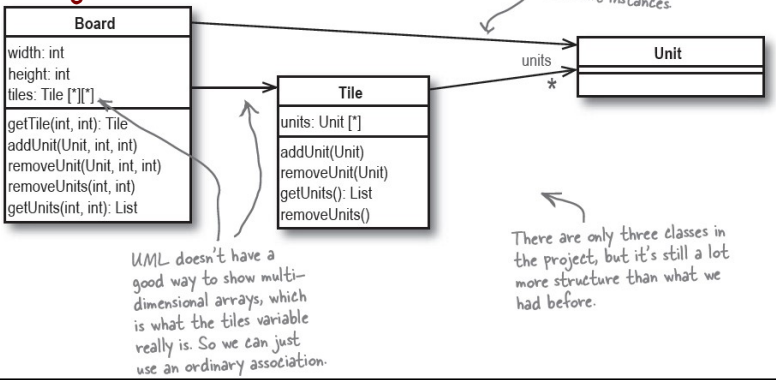
More order, less chaos

Gary's Game System Framework
KEY Features

1. The board for the game—essence of the system
2. Game-specific units—essence, and what does this mean?
3. Coordinating movement—what is it, and how do we do it?

We've got the basic Board written, so we've handled this key feature enough to move on to another.

We've got structure now, too...



Even though Board doesn't have any variables of type Unit, it's still associated to Unit because of its methods that take in Unit instances.

UML doesn't have a good way to show multi-dimensional arrays, which is what the tiles variable really is. So we can just use an ordinary association.

There are only three classes in the project, but it's still a lot more structure than what we had before.

20

Which feature should we work on next?

With some basic classes written, we can look at how classes interact, as well as begin to build on what we've already done.

We've got a Unit class now, so why don't we tackle "game-specific units" next? Besides, we can also look at how Board and Unit interact.

Architecture is your design structure, and highlights the most important parts of your app, and the relationships between those parts.

You really *can't* talk about the relationships between parts if you don't have two parts that *have* a relationship.

21

Game-specific units... what does that mean?

LET'S TALK TO THE CUSTOMERS (THE GAME DESIGNERS) TO LEARN WHAT "GAME SPECIFIC UNITS" MEAN?

Strategy is the key for our games. We use an advanced combat system where each unit has an attack strength, defense strength, and experience modifier.

Our customers are all about air battles, so we don't even need troops. I just want to be able to create a bunch of different types of planes, with different speeds, weapons, and that kind of thing.

Our games are realistic and long-term... we even keep up with the ages and relationships between characters in our games.

No good war game is good without weapons... lots of different types of weapons. And our units can hold two each, so it gets really fun fast.

I build sci-fi games, big huge space and planet battles. So I need to be able to have armies with lasers and create lots of spaceships.

22

Sharpen your pencil

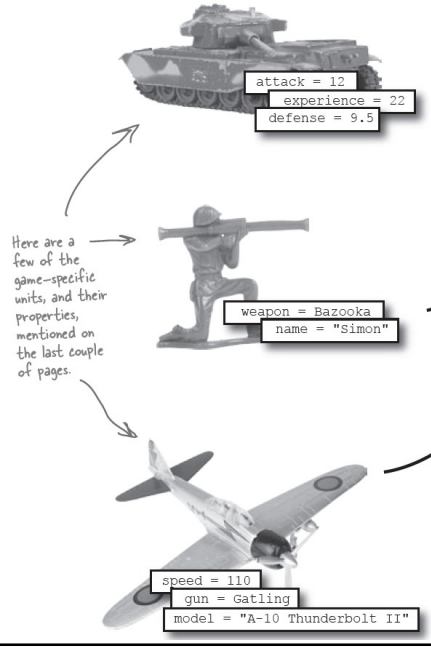
What does "game-specific units" mean?

Now that you've heard from several of the game designers who want to use Gary's game system framework, you should have a good idea of what our second key feature is all about. Write down in the blanks below your idea of what you need to do to support "game-specific units."

Each game based on the framework has different types of units, with different attributes and capabilities. So we need to be able to have properties for a unit that are different for each game, and support multiple data types for those properties.

23

Commonality revisited

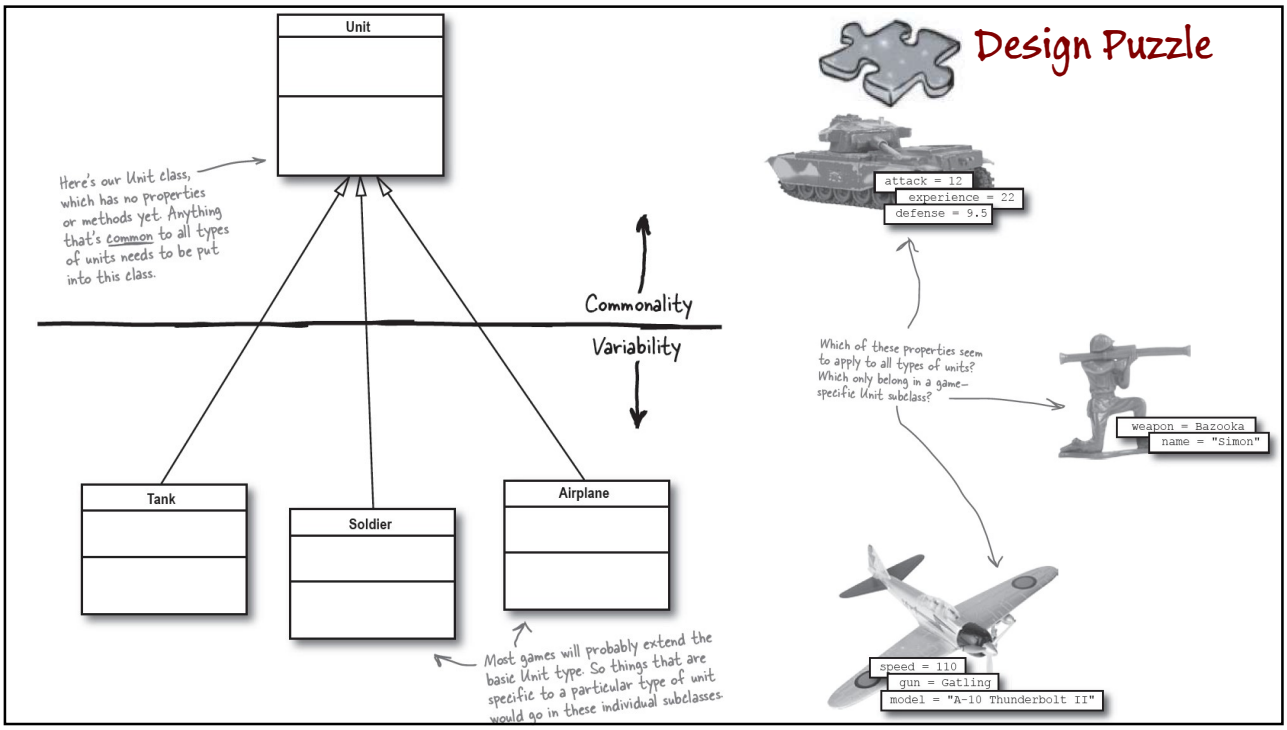


Here are a few of the game-specific units, and their properties, mentioned on the last couple of pages.

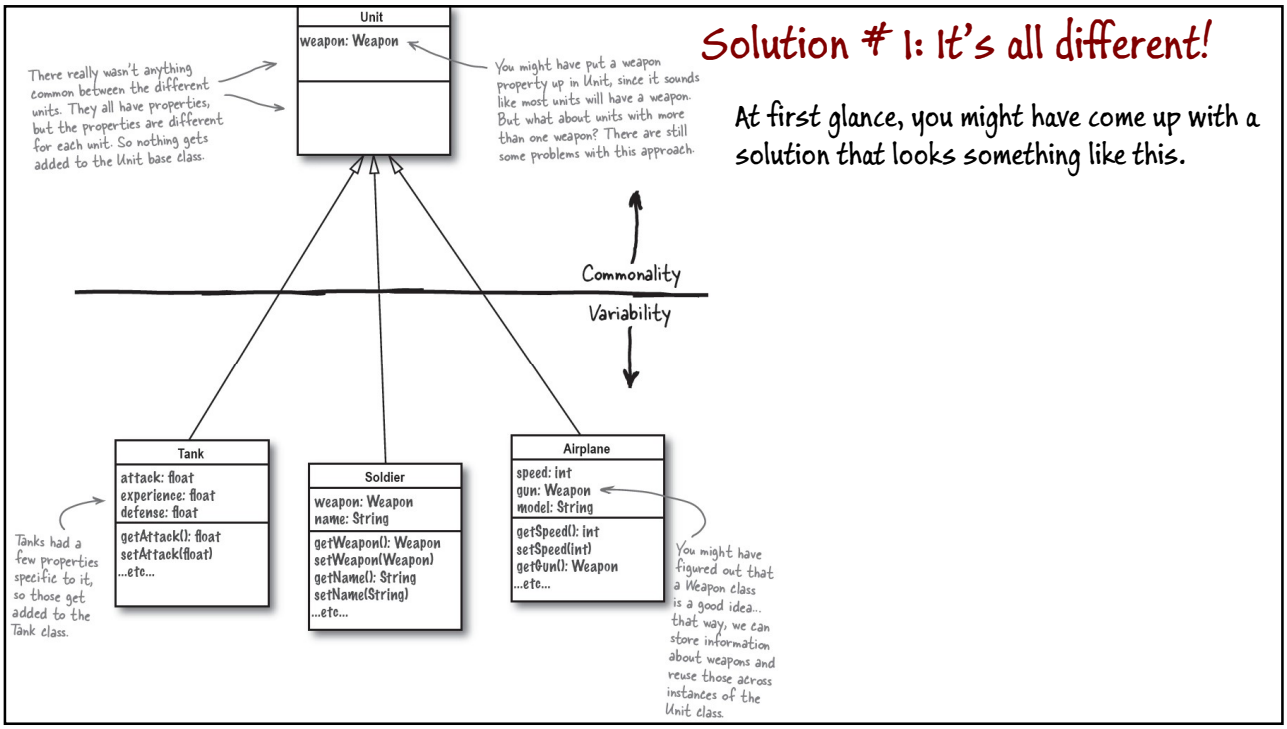
What is common among these different types of units? What basic things can we say that would apply to any game's units?

We talked about commonality when we were trying to gather basic system requirements. It also applies to smaller problems, like the game-specific units.

24


Design Puzzle
Unit
Here's our Unit class, which has no properties or methods yet. Anything that's common to all types of units needs to be put into this class.
Commonality
Variability
Tank
Soldier
Airplane
Most games will probably extend the basic Unit type. So things that are specific to a particular type of unit would go in these individual subclasses.
attack = 12
experience = 22
defense = 9.5
Which of these properties seem to apply to all types of units? Which only belong in a game-specific Unit subclass?
weapon = Bazooka
name = "Simon"
speed = 110
gun = Gatling
model = "A-10 Thunderbolt II"

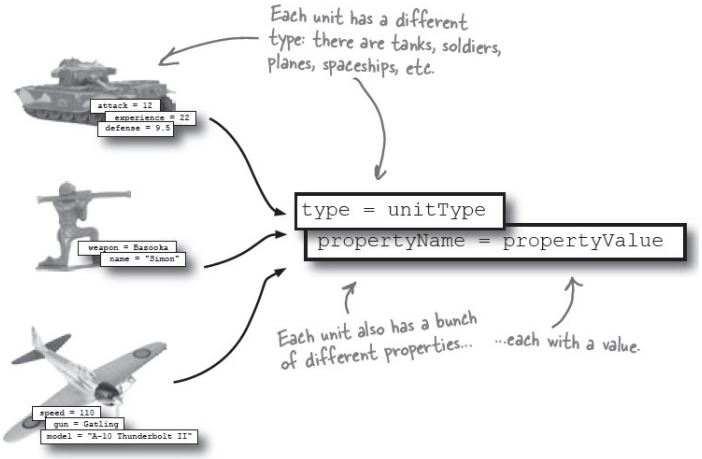
25


Solution # 1: It's all different!
At first glance, you might have come up with a solution that looks something like this.
Unit
weapon: Weapon
There really wasn't anything common between the different units. They all have properties, but the properties are different for each unit. So nothing gets added to the Unit base class.
You might have put a weapon property up in Unit, since it sounds like most units will have a weapon. But what about units with more than one weapon? There are still some problems with this approach.
Commonality
Variability
Tank
attack: float
experience: float
defense: float
getAttack(): float
setAttack(float)
...etc...
Soldier
weapon: Weapon
name: String
getWeapon(): Weapon
setWeapon(Weapon)
getName(): String
setName(String)
...etc...
Airplane
speed: int
gun: Weapon
model: String
getSpeed(): int
setSpeed(int)
getGun(): Weapon
...etc...
Tanks had a few properties specific to it, so those get added to the Tank class.
You might have figured out that a Weapon class is a good idea... that way, we can store information about weapons and reuse those across instances of the Unit class.

26

Commonality is about more than just the names of properties...

Let's take a step back from focusing on the actual names of the properties for each unit. What's really the same for each unit?

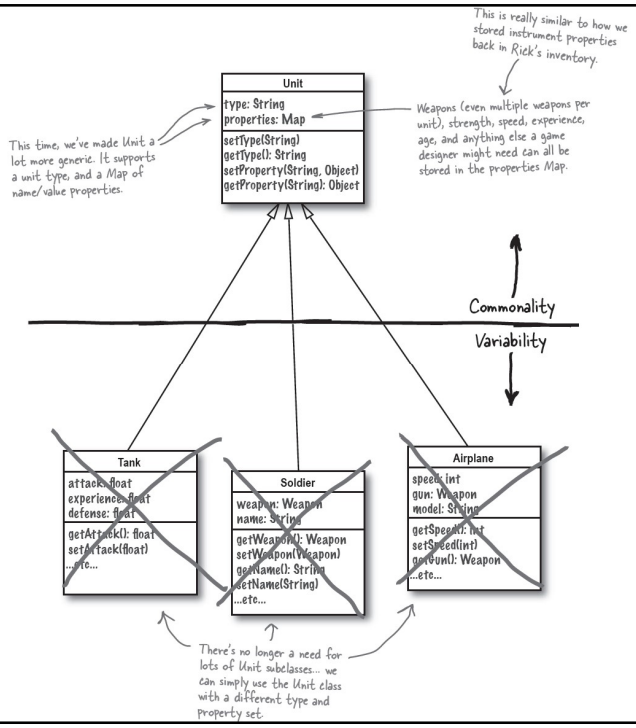


That was sort of dumb... why go through all this commonality stuff when there's nothing common between the different units? That seemed like a waste of time.

So what is common is that a unit has a type and a set of properties, each of which is a simple name/value pair.

27

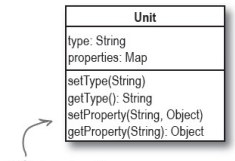
Solution # 2: It's all the same!



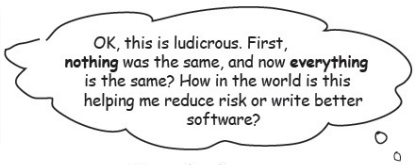
28

Commonality analysis: the path to flexible software

Wondering why we spent all that time on commonality analysis? Look back at the first solution and then again at the second solution, and then fill out the table below to see what commonality has really bought us in terms of our design:



With a single well-designed Unit class, we can support any number of different unit types.



Good design will always reduce risk.

Number of unit types	Number of unit classes – Solution #1	Number of unit classes – Solution #2
3	4	1
5	6	1
10	11	1
25	26	1
50	51	1
100	101	1

This first row is what we've looked at so far: 3 different unit types.

100 may seem like a lot, but in massive war games, it's not so far-fetched.

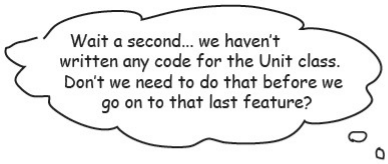
With Solution #1, you always had a Unit base class, and a subclass for each unit type.

With Solution #2, the single Unit class supported all types of units, with any number of different types of properties and attributes.

? Which solution do you think is better Why

29

And still more order...

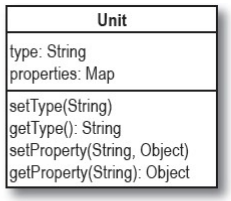


Gary's Game System Framework
KEY Features

1. The board for the game—essence of the system
2. Game-specific units—essence, and what does this mean?
3. Coordinating movement—what is it, and how do we do it?

We're focusing on doing just the things that reduce risk.

Remember, the point of architecture is to **reduce risk**, and to **create order**.

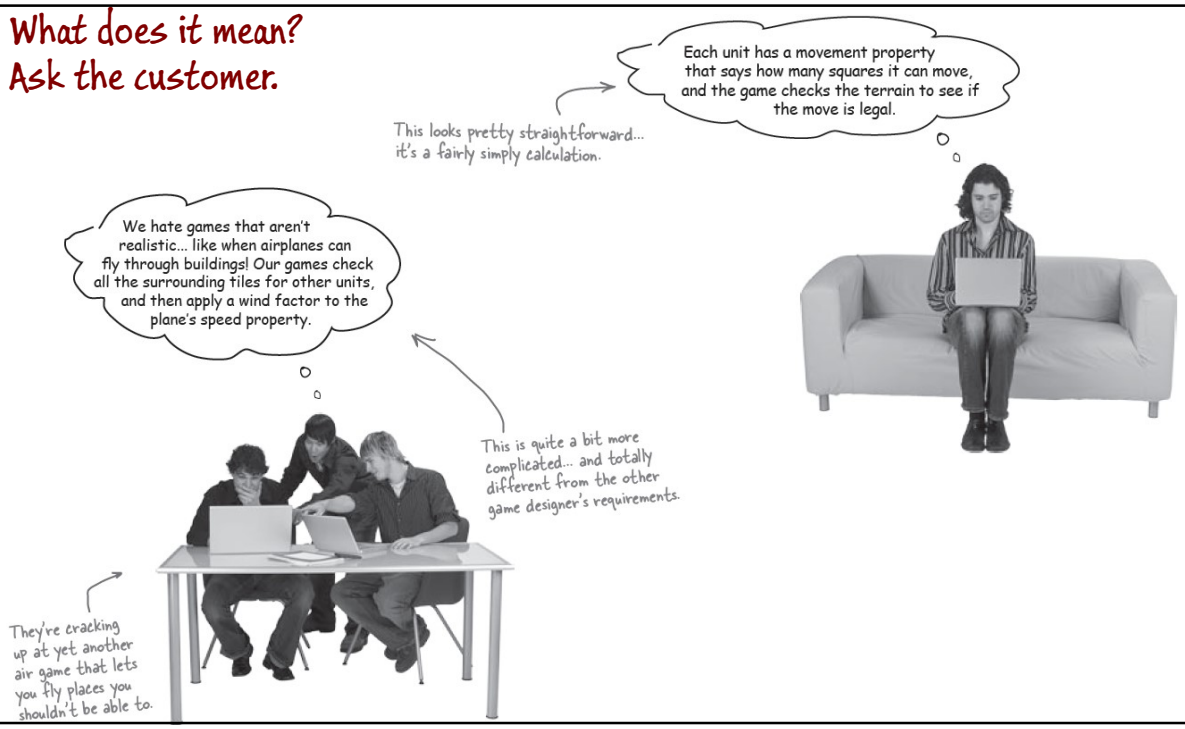


This class diagram is all you need at this point. It gives you the structure of the Unit class, and answers the question, "What does 'game-specific units' mean?"

Sometimes the best way to write great code is to hold off on writing code as long as you can.

30

What does it mean? Ask the customer.



31

Do you know what "coordinating movement" means?

Listening to the customers should have given you a pretty good idea of what the third key feature of Gary's game system framework is all about. Write what you think that feature really means in the blanks below:

_____

_____

Now do some commonality analysis

Next you need to try and figure out what's common about the different movement scenarios that the customers have been talking about. Are there some basic things that apply to all the different types of movement? If you think that there are, write those common things in the blanks below:

_____

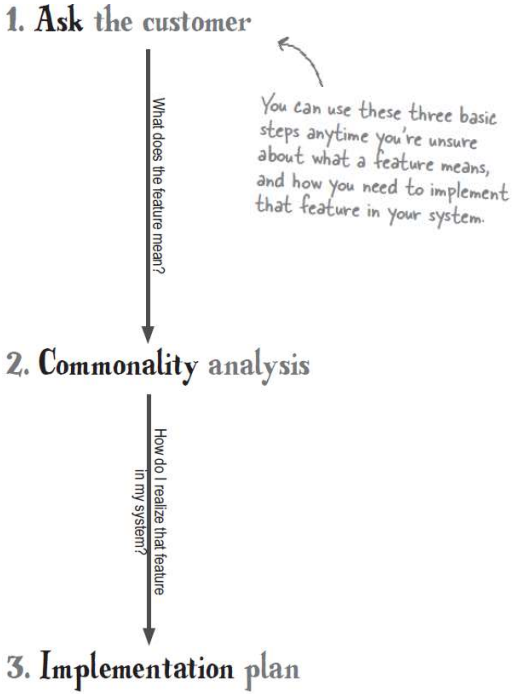
_____

So now what would you do?

If you have an understanding of what "coordinating movement" means, and you know what things are common across all games, you should have an idea about what you need to do to the game framework to make this feature work. Write your ideas down in this final set of blanks:

_____

_____

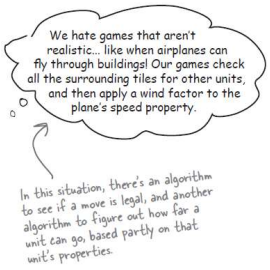


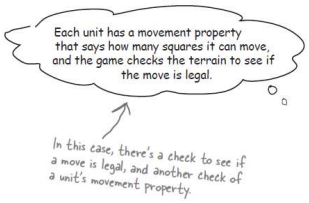
32

Is there anything common here?

Units should be able to move from one tile on the board to another. Movement is based on a calculation or algorithm specific to each game, and sometimes involves the game-specific properties of a unit.

So what exactly is common among all the different possible movement scenarios? Remember what the customers said.





What's common?	What's variable?
There's a check prior to a move to see if the move is legal.	The algorithm to check a move's legality is different for every game.
A unit's properties are used to see how far the unit can move.	The number and specific properties used are different for every game.
Factors other than the unit affect movement.	The factors that affect movement are different for every game.

See a recurring theme here?

This is where things like wind speed come into play.

33

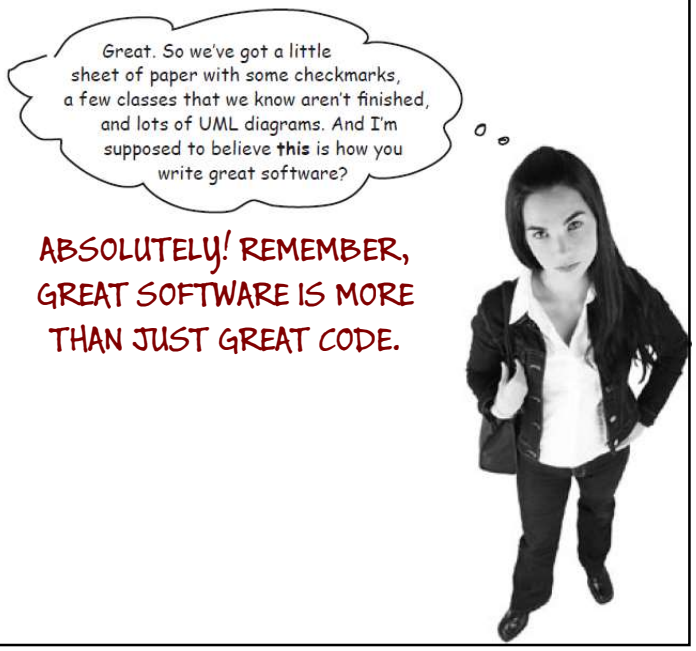
It's "different for every game"

Did you see what kept showing up in our chart? Every time we found some commonality, the variability column had the same words: "different for every game."

WHEN YOU FIND MORE THINGS THAT ARE DIFFERENT ABOUT A FEATURE THAN THINGS THAT ARE THE SAME, THERE MAY NOT BE A GOOD, GENERIC SOLUTION.

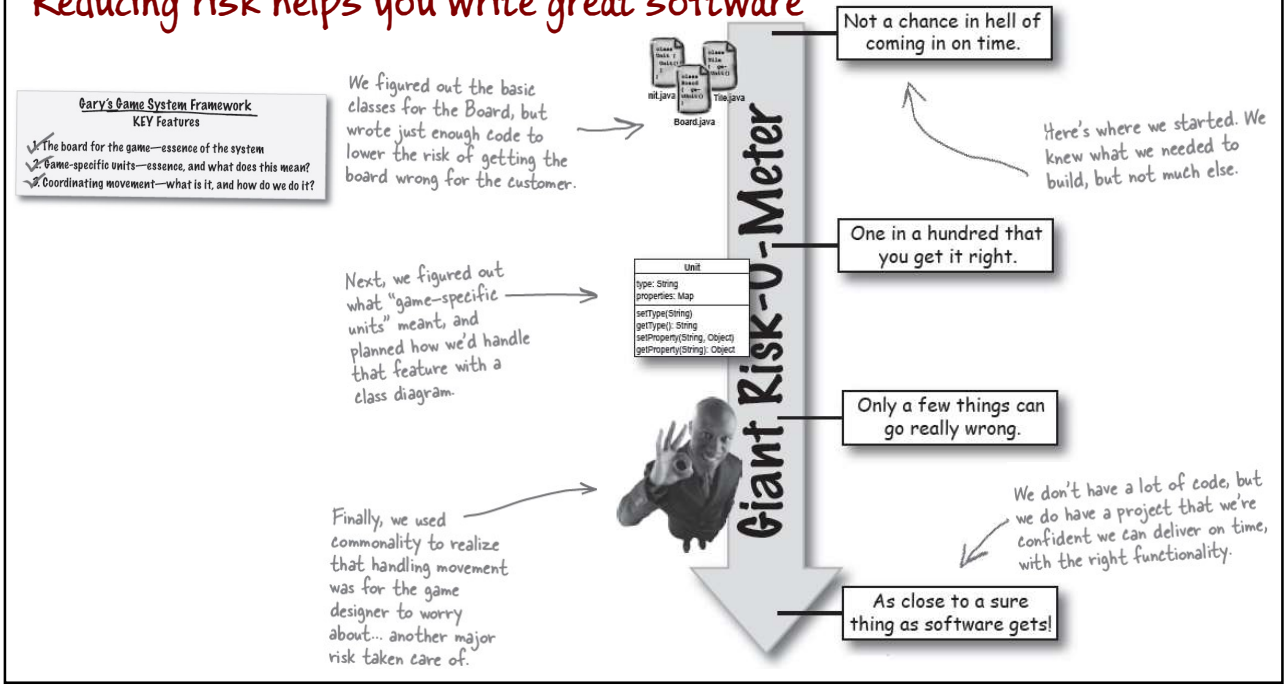
34

Customers don't pay you for great code, they pay you for great software.



35

Reducing risk helps you write great software



36

Tools for your toolbox

Bullet Points

- Architecture helps you turn all your diagrams, plans, and feature lists into a well-ordered application.
- The features in your system that are most important to the project are architecturally significant.
- Focus on features that are the essence of your system, that you're unsure about the meaning of, or unclear about how to implement first.
- Everything you do in the architectural stages of a project should reduce the risks of your project failing.
- If you don't need all the detail of a use case, writing a scenario detailing how your software could be used can help you gather requirements quickly.
- When you're not sure what a feature is, you should ask the customer, and then try and generalize the answers you get into a good understanding of the feature.
- Use commonality analysis to build software solutions that are flexible.
- Customers are a lot more interested in software that does what they want, and comes in on time, than they are in code that you think is really cool.

37