

Software Engineering, especially as it applies to your CS Capstone Project

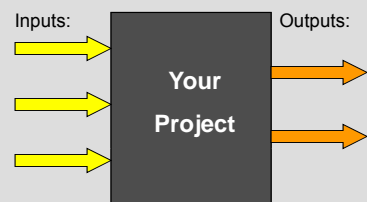
Mike Bailey

Oregon State University



Oregon State University
Computer Graphics

Requirements Document



- Express desired behaviors
- Treat your project as a "Black Box", with defined inputs that produce defined outputs
- Express entities and relationships between entities
- Express states and transitions between states
- Express limits and constraints
- ***Do not give implementation details unless they are explicitly part of the client's specification to you***



Oregon State University
Computer Graphics

mjb - October 13, 2009

Requirements Document: Address All Stakeholders

- Clients
- Clients' customers (current and future)
- Domain experts
- Lawyers
- Marketing
- Technology experts
- Security experts

Clearly all of these don't apply to you
now, but some day ...



Oregon State University
Computer Graphics

mjb - October 13, 2009

Break the Requirements into Three Categories

1. Must have – *Essential*
2. Important – *Desirable*
3. Nice if have time -- *Optional*

Requirements Should be Testable

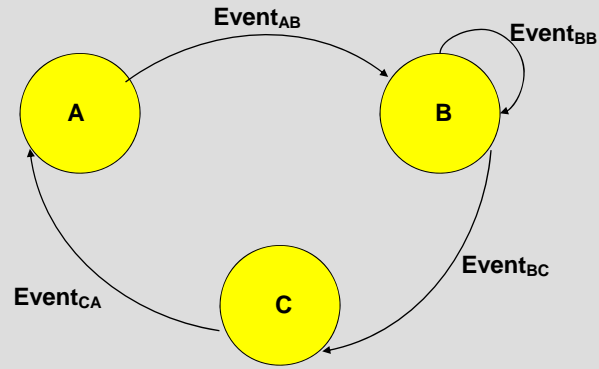
- “Fast” – **NO !**
- “≥ 30 frames per second” – **YES !**



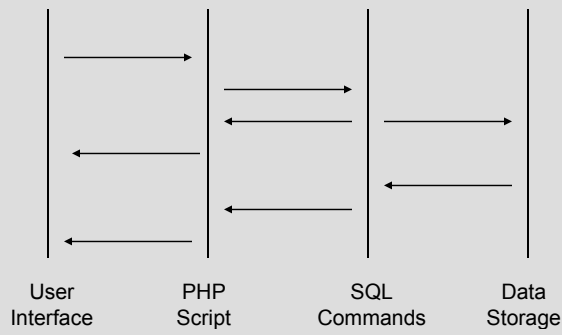
Oregon State University
Computer Graphics

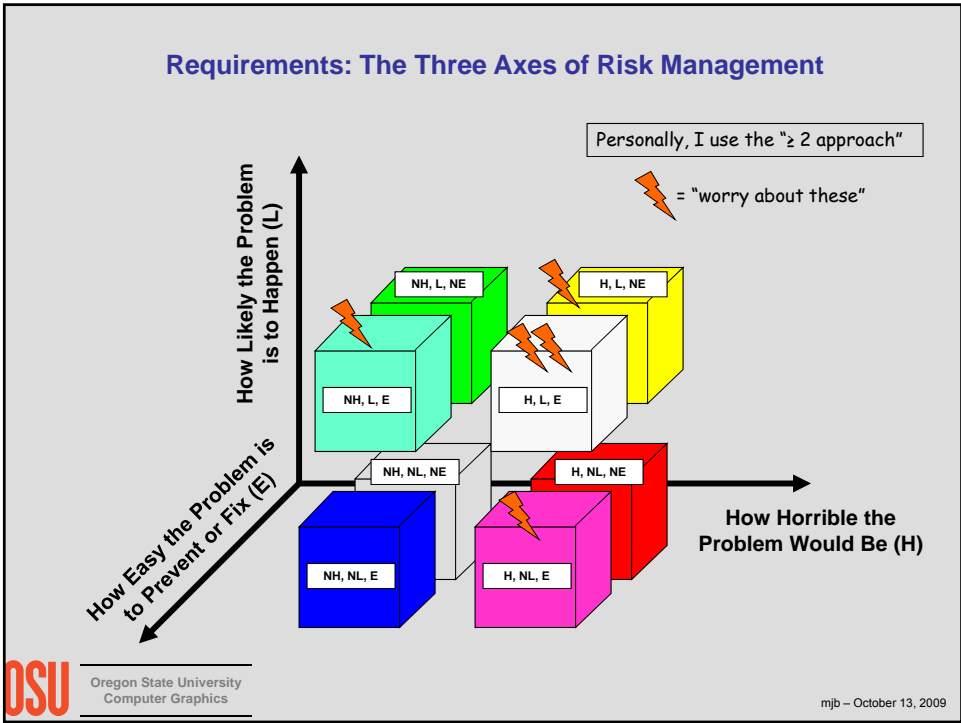
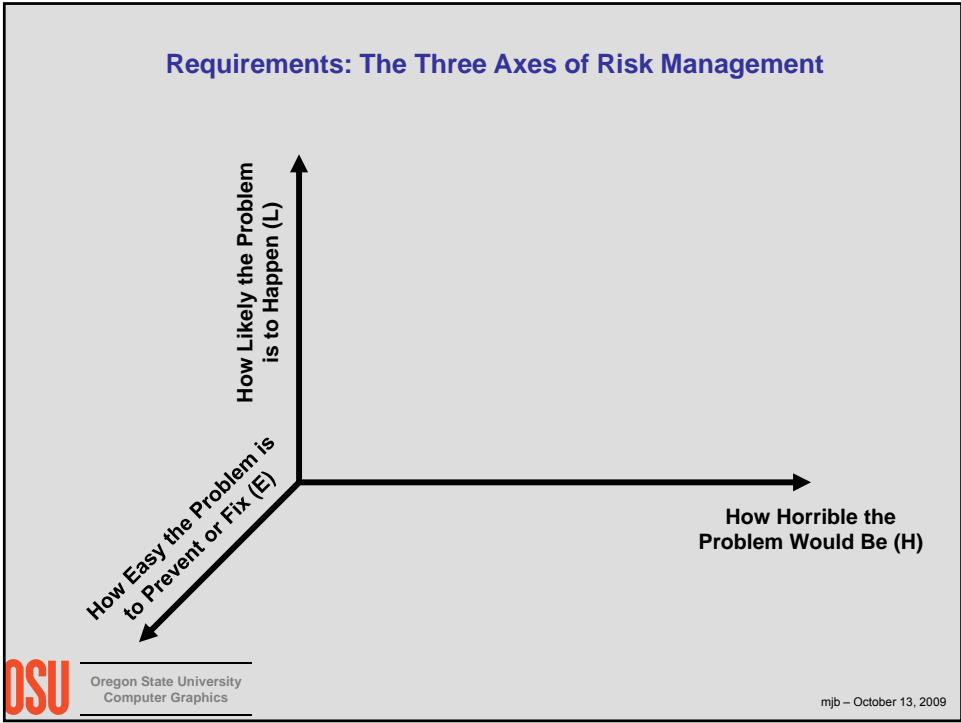
mjb - October 13, 2009

Requirements: The State Diagram



Requirements: The Event Trace Diagram





Requirements: Risk Management

Examples of Risks to consider in your CS 46* Project Planning:

- Client is unavailable for an extended time
- Technology you need is unavailable
- Technology you need takes too long to figure out
- Technology you need doesn't work as expected
- Client changes requirements
- Debugging time was underestimated
- Learning time was underestimated

Do Not Include:

- Team member gets hit by a bus
- Hangovers
- Other classes
- Losing files . . .



Oregon State University
Computer Graphics

mjb – October 13, 2009

. . . Losing Files

Losing your files because you failed to back them up is not a risk that I will tolerate, and neither should you!

Do:

- Backup to your OSU account(s).
- Backup to someone else's computer.
- Backup to an online service.
PC Magazine recently rated several:
http://www.pcmag.com/print_article2/0,1217,a%253D226992,00.asp

EMC recently came to campus and made a free 2 GB offer to OSU students:
<http://mozy.com/students>

- Keep multiple versions

Do not:

- Trust backups to the memory stick you carry around in your backpack
- Trust backups to the pile of DVDs on the same table as your home computer



Oregon State University
Computer Graphics

mjb – October 13, 2009

Requirements: Risk Management

What to do about a risk:

1. Avoid it – don't do that part of the project **NO !**
2. Contain it – set aside sufficient resources **YES !**
3. Mitigate it – take steps beforehand to soften or handle the problem **YES !**

In your Requirements Document, make note of:

- Risk Transitions – what would make this situation happen?
- Transition Indicators – how you will know it has started to happen?
- Mitigation – what upfront plans can you make in case it happens?
- Contingency planning – what will you do when it happens?



Oregon State University
Computer Graphics

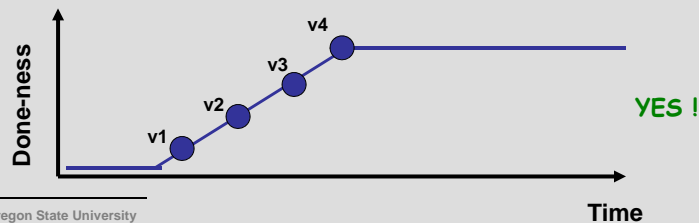
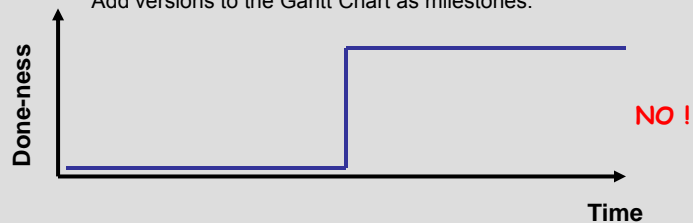
mjb – October 13, 2009

Risk Management

Building incrementally helps mitigate risk

1. *The client gets to the "it's not what I wanted" stage sooner*
2. *You get to the "it's not going to work as we expected it to" stage sooner*

Add versions to the Gantt Chart as milestones:

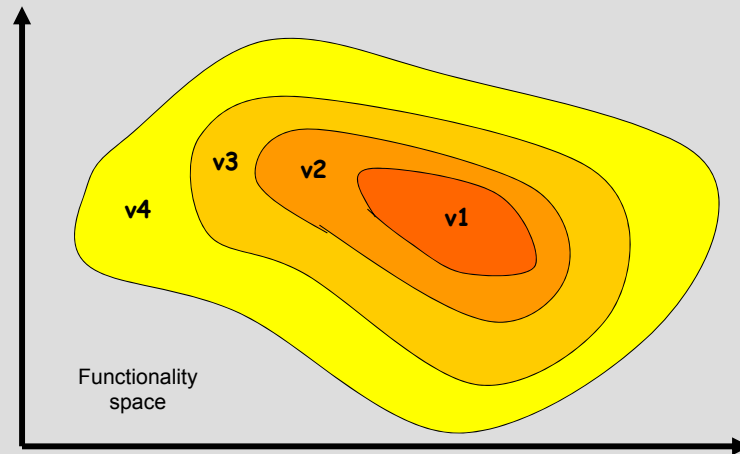


Oregon State University
Computer Graphics

mjb – October 13, 2009

Risk Management

Think of it this way:



Oregon State University
Computer Graphics

mjb - October 13, 2009

Project Management

Work Breakdown Schedule (WBS)

- List all tasks
- Don't yet worry about durations or dependencies
- Use Post-It notes on a wall or white board to start

Activity Graphs

- Add dependencies – what tasks can't start until other tasks end?
 1. Sometimes due to functionality-availability reasons
 2. Sometimes due to people-availability reasons

PERT chart

- Add durations and milestones
- Shows "slack time" of each task, i.e., how much can each task slip before the entire project slips
- Critical Path analysis (0 slack at each node)



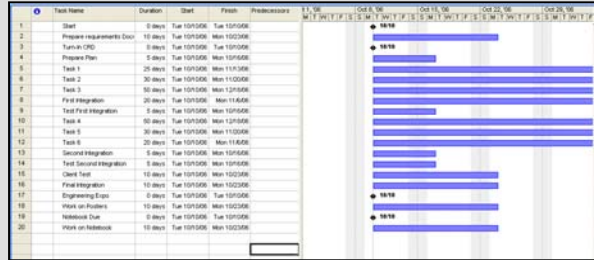
Oregon
Computer Graphics

- The Gantt chart must come from the PERT chart, if it's to have any meaning

mjb - October 13, 2009

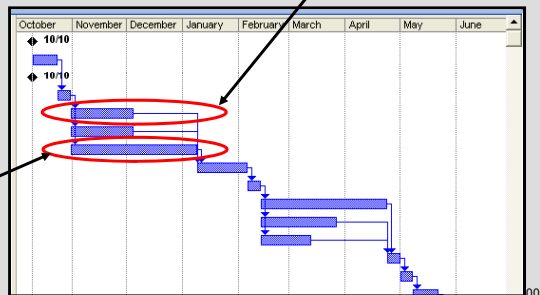
Project Management: Software

Work Breakdown Schedule: tasks and durations



Much slack here. Could this person help with the task that has no slack?

Dependencies added:



No slack in the schedule, i.e., the Critical path



Oregon State University
Computer Graphics

009

Testing and Debugging: Plan for it in the First Place

Make your code readable, and even pronounceable:

- if((flag&1) != 0) **OK**
- if(DONE(flag)) **Better**
- if(done) **Better**
- if(itsDone) **Best**

What if you really do need to test the flag?

```
#define DONE(f)  ( (f&1) != 0 )
...
if( DONE(flag) )
```

```
#define itsDone ( (flag&1) != 0 )
...
if( itsDone )
```



Oregon State University
Computer Graphics

mjb - October 13, 2009

Testing and Debugging: Plan for it in the First Place

Litter your code with instrumentation

```
bool Verbose;  
...  
if( Verbose )  
    fprintf( stderr, "Termination condition in iteration #%d\n", iter );
```

For debugging, always print to standard error, which is *unbuffered*, instead of standard output, which is *buffered*!

Or,

```
#define DebugPrint(s,x)    fprintf( stderr, s, x )  
...  
DebugPrint( "Termination condition in iteration #%d\n", iter );
```

And then in production mode:

```
// #define DebugPrint(s,x)    fprintf( stderr, s, x )  
  
#define DebugPrint(s,x)
```

This avoids the overhead of the if-test



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging

- Orthogonal testing – test *only one feature* at a time

Remember that the "only one feature at a time" includes the computer itself, amount of memory available, background load, etc. !

- Bisection testing – narrow down where the bug is
- Regression testing of new versions – run old tests, did you break anything?
- Revision control / change control – able to go back to working versions
- Stress tests – how does it do at the limits?
- Have the debugging messages tell what the inputs were as well as what went wrong
- Write the messages to a file so you can retrieve or print them later

File I/O is buffered -- flush the file often !!

```
fprintf( fileptr, "x = %f\n", x );  
fflush( fileptr );
```

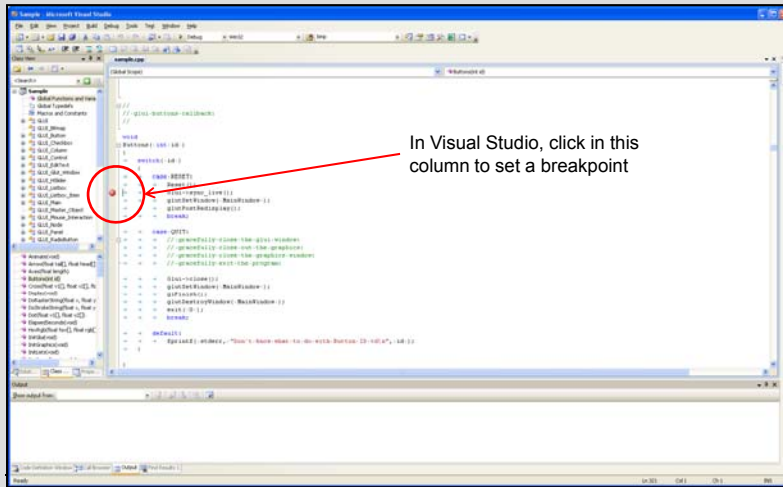


Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging

Use Debuggers to set breakpoints and examine values when those breakpoints have been hit

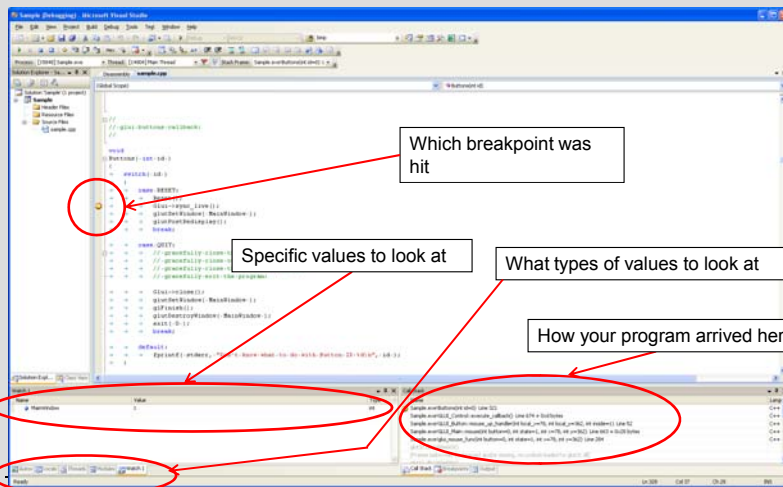


Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging

Use Debuggers to set breakpoints and examine values when those breakpoints have been hit



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging: Make your Program Readable in the First Place

Gunning's "Fog Index" (doesn't exactly apply to programming, but it's interesting):

$$F = 0.4 \times \frac{\# \text{ words}}{\# \text{ sentences}} \times (\% \text{ words} \geq 3 \text{ syllables}) \approx \text{GradeLevel}$$

De Young and Kampen's Program Readability:

$$R = 0.295a - 0.411b + 0.13c$$

(a larger R means "more readable")

a = the average length of variable names

b = number of lines of statements

c = the number of linearly independent paths through the code, or can also be expressed as one plus the number of branch points (if, switch, while, do, for)

In another study, the c multiplier was -0.318, which makes more sense to me...



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging: Fault Types

- Algorithm Fault
 - Computation of Precision Fault
 - Documentation Fault
 - Boundary Fault
 - Recovery Fault
 - Deadlock Fault (and Livelock Fault)
 - Race Condition Fault
- } Multithreading issues



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging: Fault Types

Algorithm Fault

- Testing for wrong condition
- Forgot to initialize variables
- Didn't test for special conditions
- Type incompatibility
- Typos

```
for( float ang = 0.; ang <= 2.*M_PI; ang += 2.*M_PI/30. )
```

Initialize *all* variables - don't count on the linker/loader to do it for you!

"I don't need to test for _____. It will never happen."

```
glBegin( GL_TRIANGLES );  
glVertex( x0, y0, z0 );  
glVertex( x1, y1, z1 );  
glVertex( x2, y2, z2 );  
glEnd;
```

```
int num = 1;  
int denom = 2;  
float percent = 100. * ( num / denom );
```

Should be:

```
float percent = 100. * ( (float)num / (float)denom );
```

Computation or Precision Fault

- Does not compute the correct result, even though the formula is correct

```
a = (x+y) + z;  
b = x + (y+z);  
if( a == b ) . . .
```



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging: Avoid Accumulated Computational Error

```
Δθ = 2.*π / 360.;  
θ = 0.;  
for( i = 0; i <= 360; i++ )  
{  
    . . .  
    θ = θ + Δθ ;  
}
```

NO !

```
Δθ = 2.*π / 360.;  
for( i = 0; i <= 360; i++ )  
{  
    θ = Δθ * (float)i / 360.;  
    . . .  
}
```

YES !



Oregon State University
Computer Graphics

mjb - October 13, 2009

Debugging: Fault Types

Documentation Fault

- Does one thing, but the documentation says it should do another

Boundary Fault

- Does not handle limit conditions correctly

Example: more data is being read from a file than the size of the array created to store it.

Recovery Fault

- Needs to gracefully handle an exception, but doesn't



Oregon State University
Computer Graphics

mjb - October 13, 2009

Multithreaded Programming

A "thread" is an independent path through the program code. Each thread has its own program counter, registers, and stack. But, since each thread is executing some part of the same program, each thread has access to the same static and dynamic memory. Each thread is scheduled and swapped just like any other process.

When is it good to use Multithreading?

- Where specific tasks can become blocked, waiting for something
- Where specific tasks can be CPU-intensive
- Where specific tasks must respond to asynchronous I/O, including the UI
- Where specific tasks have higher or lower priority than other tasks
- Where performance can be gained by overlapping I/O
- To manage independent behaviors in interactive simulations
- When you want to use data-level parallelism with multicore CPU chips

We will have a couple of classes on multicore / multithreading in the WQ!

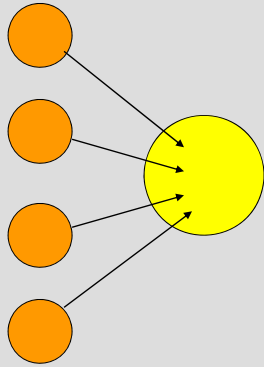


Oregon State University
Computer Graphics

mjb - October 13, 2009

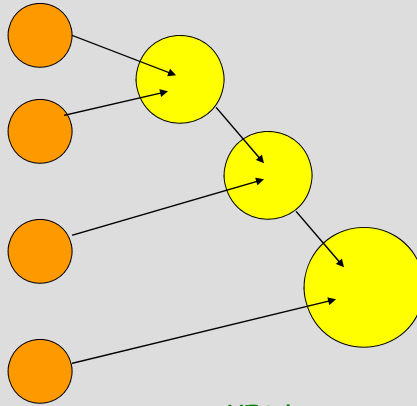
Debugging: Styles

“Big Bang” Approach



NO !

“Sandwich” (Layer) Approach



YES !



Oregon State University
Computer Graphics

mjb – October 13, 2009

Extreme Programming

- Customers: create “stories”
- Programmers: implement stories
- Work in 2-week intervals
- Daily meetings (no chairs?)
- Customer (or customer’s “agent”) is *always* present

Pair Programming

- One keyboard, one mouse, two programmers
- Two chained monitors recommended
- One is the Driver (or Pilot)
- One is the Navigator
- Periodically change roles



Oregon State University
Computer Graphics

mjb – October 13, 2009

Agile Programming, I

1. Value individuals over process (face-to-face interaction)
2. Value having working software over comprehensive documentation
3. Value customer collaboration over contract negotiation
4. Satisfy customer with early and continuous delivery of good software
5. Value responding to change over following a rigid plan

Recognize that conditions change over time:

- Business needs
- Market conditions
- Requirements realizations
- Technical advances
- Competition
- Economic realities
- What people you have on the project



Oregon State University
Computer Graphics

mjb – October 13, 2009

Agile Programming, II

- 2-4 week “sprint” to address the highest priority current need
- Daily “scrum” – no chairs, on-site customer or customer spokesman present
- Participants: “pigs” get to talk, “chickens” must just listen
- Continuous integration

Note: “Agile Programming” does not mean “no plan”, it means a continuously-changing plan under customer control

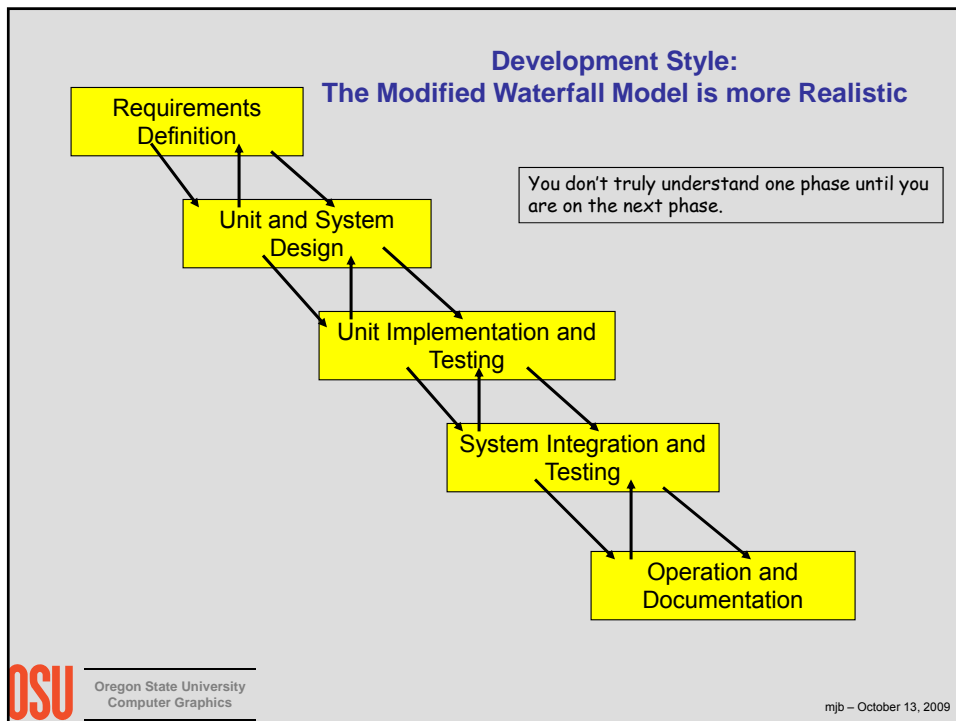
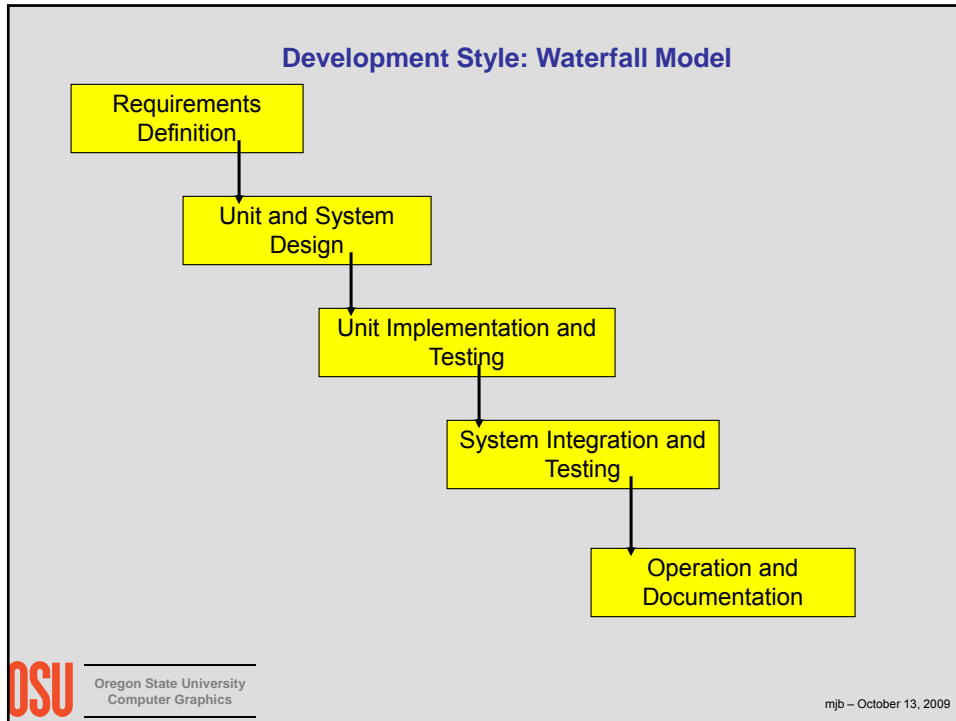
I don't mind if you want to do things this way, but:

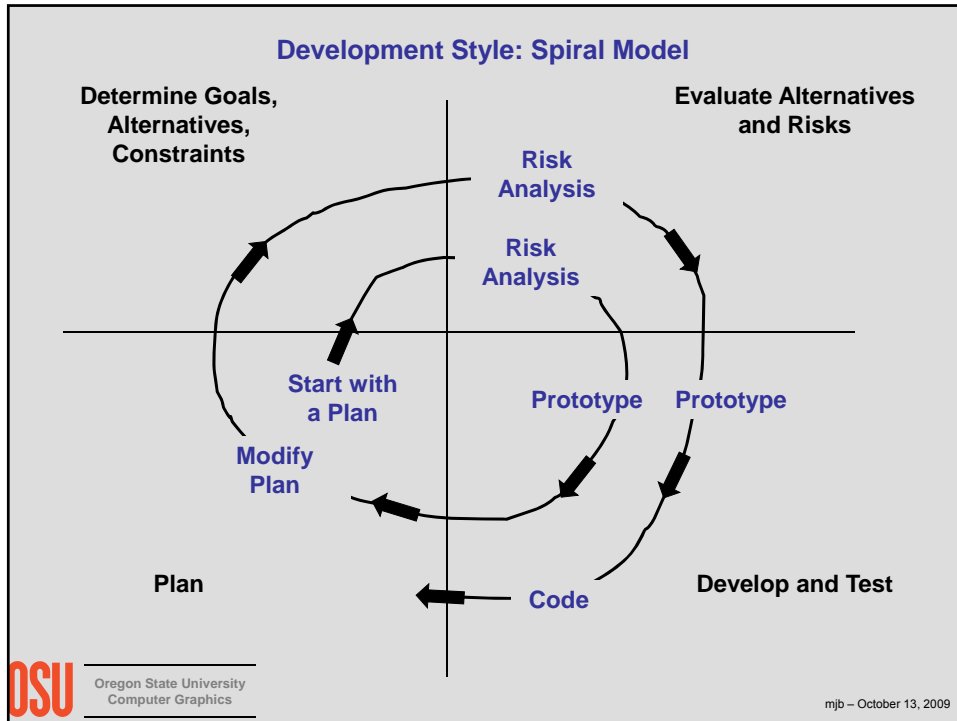
1. I want to know about it and be kept informed about how it is working
2. Let me know how you plan to always have the client present



Oregon State University
Computer Graphics

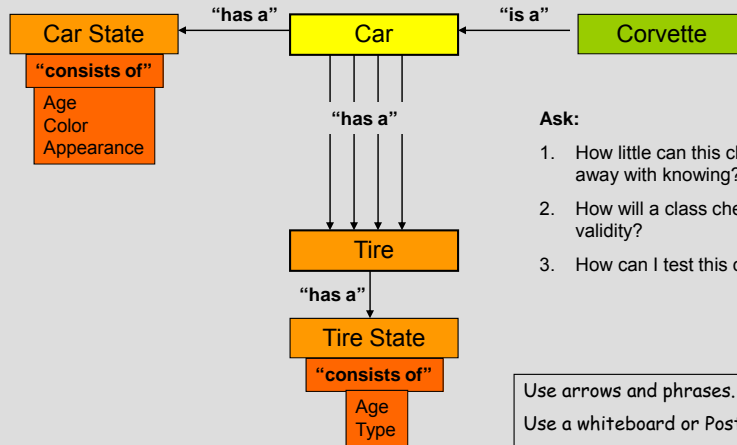
mjb – October 13, 2009





- ### Development: Object-Oriented Programming Characteristics
- An object consists of properties and behaviors, and also has:**
1. **Identity** – organized into discrete entities
 2. **Abstraction** – represents different views of the data
 3. **Classification** – group objects that have properties and behaviors in common (“is-a”)
 4. **Encapsulation** – hides implementation details
 5. **Inheritance** – start with a broad definition, then refine into more specialized sub-classes
 6. **Polymorphism** – automatically select correct method from all same-named methods; allows new classes to be coded without changing existing code.
 7. **Persistence** – an object’s name, properties, and behaviors are all retained
- OSU Oregon State University Computer Graphics mjb – October 13, 2009

Object-Oriented Programming: Draw a Picture *before* You Start Coding



Ask:

1. How little can this class get away with knowing?
2. How will a class check data validity?
3. How can I test this class?

Use arrows and phrases.
Use a whiteboard or Post-it notes.
Save the picture! Make it part of your documentation.



Oregon State University
Computer Graphics

mjb - October 13, 2009

Don't re-invent code that already exists: C++ Standard Template Library (STL)

- **Vector**: a dynamic array
- **List**: double-linked list
- **Stack/Queue**: insertion at beginning or end
- **Set**: sorted set with Boolean operators
- **Map**: associative array
- **Iterators**: input, output, forward, backward, random access



Oregon State University
Computer Graphics

mjb - October 13, 2009

Don't re-invent code that you can re-use: Design Patterns

A *design pattern* is a solution to a commonly-occurring problem in software design. By identifying certain patterns this way, we can talk about reusable solutions to common situations. Gamma, Helm, Johnson, and Vissides define:

Creational Patterns

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton

Structural Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

Behavioral Patterns

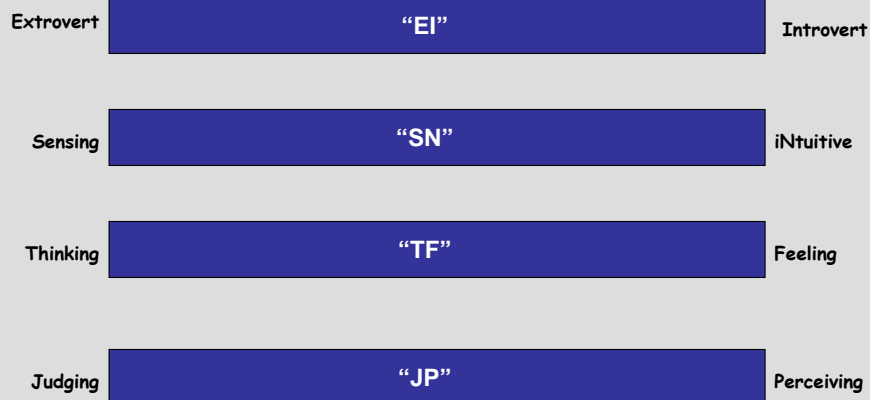
- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor



Oregon State University
Computer Graphics

mjb – October 13, 2009

Interacting in a Team: the Myers-Briggs Type Indicator (MBTI) Test



These are dimensions that each of us uses to view and interpret the world around us and come to make decisions about it.



Oregon State University
Computer Graphics

mjb – October 13, 2009

MBTI Pairs of Preferences

Attitudes: Extraversion (E) / Introversion (I)

The terms *Extravert* and *Introvert* are used in a different sense when discussing the MBTI than we normally use these terms.

The preferences for **Extraversion** and **Introversion** are sometimes referred to as *attitudes*. Briggs and Myers recognized that each of the functions can show in the external world of behavior, action, people and things (*extraverted attitude*) or the internal world of ideas and reflection (*introverted attitude*).

People with a preference for Extraversion draw energy from action: they tend to act, then reflect, then act further. If they are inactive, their level of energy and motivation tends to decline.

Conversely, those whose preference is Introversion become less energized as they act: they prefer to reflect, then act, then reflect again. People with Introversion preferences need time out to reflect in order to rebuild energy.

The Introvert's flow is directed inward toward concepts and ideas and the Extravert's is directed outward towards people and objects. There are several contrasting characteristics between Extraverts and Introverts: Extraverts desire breadth and are action-oriented, while introverts seek depth and are thought-oriented.



Oregon State University
Computer Graphics

http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator 009

MBTI Pairs of Preferences

Functions: Sensing (S) / iNtuition (N) and Thinking (T) / Feeling (F)

Each person uses one of these four functions more dominantly and proficiently than the other three; however, all four functions are used at different times depending on the circumstances.

Sensing and **intuition** are the information-gathering (Perceiving) functions. They describe how new information is understood and interpreted. Individuals who prefer *Sensing* are more likely to trust information that is in the present, tangible and concrete: that is, information that can be understood by the five senses. They tend to distrust hunches that seem to come out of nowhere. They prefer to look for details and facts. For them, the meaning is in the data.

On the other hand, those who prefer *iNtuition* tend to trust information that is more abstract or theoretical, that can be associated with other information (either remembered or discovered by seeking a wider context or pattern). They may be more interested in future possibilities. They tend to trust those flashes of insight that seem to bubble up from the unconscious mind. The meaning is in how the data relates to the pattern or theory.



Oregon State University
Computer Graphics

http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator 009

MBTI Pairs of Preferences

Functions: Sensing (S) / iNtuition (N) and Thinking (T) / Feeling (F)

Each person uses one of these four functions more dominantly and proficiently than the other three; however, all four functions are used at different times depending on the circumstances.

Thinking and **Feeling** are the decision-making (Judging) functions. The Thinking and Feeling functions are both used to make rational decisions, based on the data received from their information-gathering functions (Sensing or iNtuition). Those who prefer *Thinking* tend to decide things from a more detached standpoint, measuring the decision by what seems reasonable, logical, causal, consistent and matching a given set of rules. Those who prefer *Feeling* tend to come to decisions by associating or empathizing with the situation, looking at it 'from the inside' and weighing the situation to achieve, on balance, the greatest harmony, consensus and fit, considering the needs of the people involved.

People with a Thinking preference do not necessarily, in the everyday sense, 'think better' than their Feeling counterparts; the opposite preference is considered an equally rational way of coming to decisions (and, in any case, the MBTI assessment is a measure of preference, not ability). Similarly, those with a Feeling preference do not necessarily have 'better' emotional reactions than their Thinking counterparts.



Oregon State University
Computer Graphics

http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator 009

MBTI Pairs of Preferences

Lifestyle: Judgment (J) / Perception (P)

Myers and Briggs recognized that people also have a preference for using either the **Judging** function (Thinking or Feeling) or their **Perceiving** function (Sensing or iNtuition) when relating to the outside world (extraversion). Types with a preference for *Judging* show the world their preferred judging function (Thinking or Feeling).

So TJ types tend to appear to the world as logical, and FJ types as empathetic. Judging types prefer to "have matters settled." Those types ending in P show the world their preferred *Perceiving* function (Sensing or iNtuition). So SP types tend to appear to the world as concrete and NP types as abstract. Perceiving types prefer to "keep decisions open."

For Extraverts, the J or P indicates their *dominant* function; for Introverts, the J or P indicates their *auxiliary* function. Introverts tend to show their dominant function outwardly only in matters "important to their inner worlds".



Oregon State University
Computer Graphics

http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator 009

Software Engineering References

- Shari Lawrence Pfleeger and Joanne Atlee, *Software Engineering Theory and Practice*, Prentice Hall, 2006.
- Tom Demarco and Timothy Lister, *Waltzing with Bears*, Dorset House Publishing, 2003.
- Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
- Frank Tsui and Orlando Karam, *Essentials of Software Engineering*, Jones and Bartlett, 2007.
- Ian Sommerville, *Software Engineering*, Addison-Wesley, 2004.



Oregon State University
Computer Graphics

mjb – October 13, 2009