


## A Review of Software Engineering

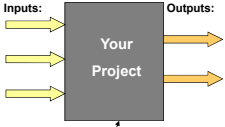
*Especially as it applies to your CS Capstone Project*

Mike Bailey  
mjb@cs.oregonstate.edu  
Oregon State University



DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Requirements Document 101



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

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Requirements Document 101: 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 ...

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Break the Requirements into Three Categories

- Must have – *Essential*
- Important – *Desirable*
- Nice if have time -- *Optional*

#### Requirements Should be Testable

- "It needs to be Fast" – **NO !**
- "It needs to run at least 30 frames per second" – **YES !**

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Do a Requirements Status Table

Req. #	Requirement	Status	Comments
1		Pending	
2		Complete	
3		Deleted	
4		Added	
5		Modified	
6			
7			

Use this format in your Requirements Document now, even though all requirements will be *Pending*.  
I am going to ask each team to turn in a sheet like this at the start of the Winter and Spring quarters and in your Final Report, so keep this file around.

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Requirements: Estimating Time to Complete a Task

O = *wildly* optimistic time to completion  
N = nominal time to completion  
P = *wildly* pessimistic time to completion

Mean and standard deviation time to completion:

$$\mu = \frac{O + 4N + P}{6} \quad \sigma = \frac{P - O}{6}$$

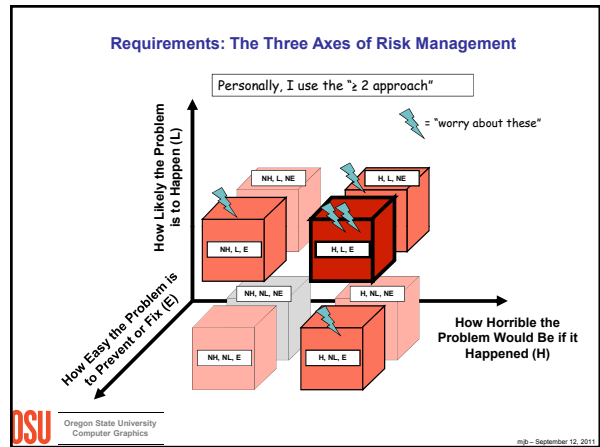
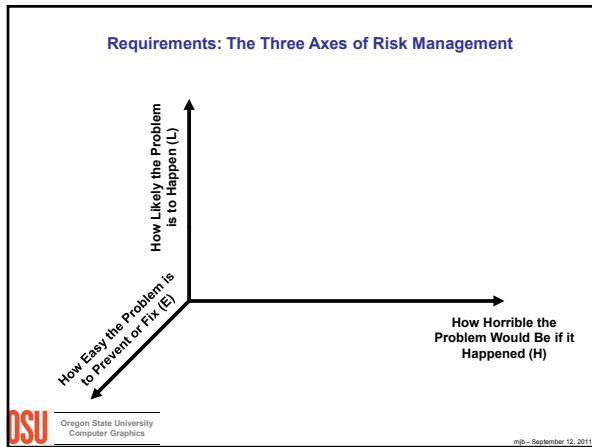
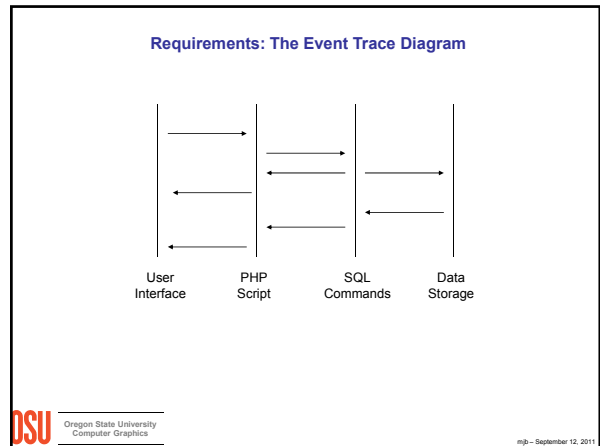
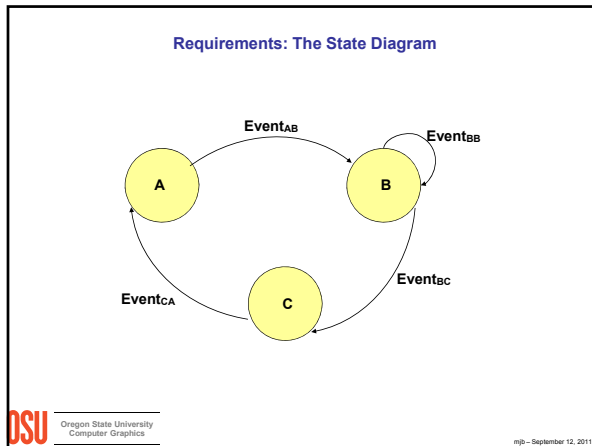
#### Estimating Time to Complete a Group of Tasks

Mean and standard deviation time to completion:

$$\mu = \sum_i \mu_i \quad \sigma = \sqrt{\sum_i \sigma_i^2}$$

This comes from *The Clean Coder*

DSU Oregon State University Computer Graphics mjb - September 12, 2011



- ### Requirements: Risk Management
- Examples of Risks to consider in your CS 46\* Project Planning:**
- Technology you need is not available when you thought it would be
  - Technology you need doesn't work as expected
  - Technology you need takes too long to figure out
  - Client changes requirements
  - Debugging time was underestimated
  - Learning time was underestimated
  - Client is unavailable for an extended time
- Do Not Include:**
- Team member gets hit by a bus
  - Hangovers
  - Other classes
  - Pregnancy
  - Losing files . . .
- Yes, I've had each of these listed as risks in the past...
- OSU Oregon State University Computer Graphics mjb - September 12, 2011

- ### . . . 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 rated several:  
[http://www.pcmag.com/print\\_article/0,1217,a%253D226992,00.asp](http://www.pcmag.com/print_article/0,1217,a%253D226992,00.asp)
  - EMC has a free 2 GB offer to students:  
<http://mozy.com/landing/students>
- A lot of teams the last couple of years used *Dropbox*
- Keep multiple versions and multiple backups
- 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
- OSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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 to make sure it doesn't happen **YES !**
3. Anticipate and Mitigate it – take steps beforehand to soften or handle the problem if it does happen **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?

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Risk Management

**Building your project 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:

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Risk Management

Think of versions this way:

OSU Oregon State University Computer Graphics mjb - September 12, 2011

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

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Project Management: Software

Work Breakdown Schedule: tasks and durations

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Getting Microsoft Project

You are not required to use any *specific* project management software in CS 46\*, but you are required to use *something*.

You can get Microsoft Project for free by accessing the OSU MSDN Academic Alliance. To do that, go to the College of Engineering *teach* system:

<http://enr.oregonstate.edu/teach>

Under **External Sites**, click **MSDNAA Site Login**. Then click on **Applications (30)**.

This will allow you to download Microsoft Project 2010 for free.

Look around while you're there – there's a lot of other free software!

Note: this is not an advertisement for Microsoft Project. It just happens that Microsoft Project has the right functionality for CS 46\*, is typical of all project management software packages, is easy to use, is already installed on most OSU engineering machines, and you all can get it for free for your own PCs.

The skills you develop by using it will transfer to *any* other project management software package.

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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

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

C++:  
inline bool DONE( int f )  
{  
    return (f&1) != 0;  
}



Oregon State University  
Computer Graphics

mjb - September 12, 2011

### 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 - September 12, 2011

### 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 );
```

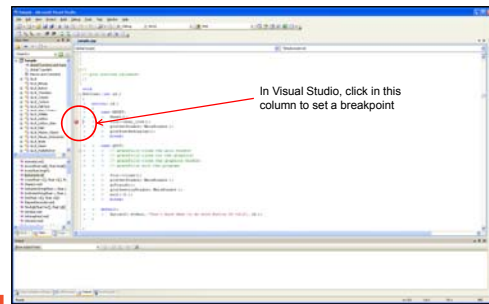


Oregon State University  
Computer Graphics

mjb - September 12, 2011

### Debugging

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

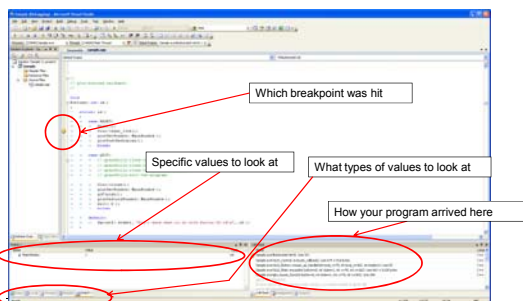


Oregon State University  
Computer Graphics

mjb - September 12, 2011

### Debugging

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



Oregon State University  
Computer Graphics

mjb - September 12, 2011

### 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 - September 12, 2011

### Debugging: Fault Types

**Algorithm Fault**

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

- Testing for wrong condition → Initialize all variables - don't count on the linker/loader to do it for you!
- Forgot to initialize variables → "I don't need to test for \_\_\_\_\_. It will never happen."
- Didn't test for special conditions
- Type incompatibility
- Types →
 

```
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 ) ...
```

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Debugging: Avoid Floating Point Accumulated Computational Error (if it's more important than speed)

```
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 !**

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Debugging: Styles

**"Big Bang" Approach**

**NO !**

**"Sandwich" (Layer) Approach**

**YES !**

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### Better Yet, Plan Testing as You Go: Tests and Coverage

This comes from *The Clean Coder*

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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 !

OSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### Agile Programming, I

1. Values customer collaboration
2. Values face-to-face interaction over a pre-determined process
3. Values having working software quickly over comprehensive documentation
4. Values early and continuous delivery of software
5. Values responding to change in requirements over following a pre-determined plan

**Agile programming recognizes that conditions change over time:**

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

DSU Oregon State University Computer Graphics mjb - September 12, 2011

### 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

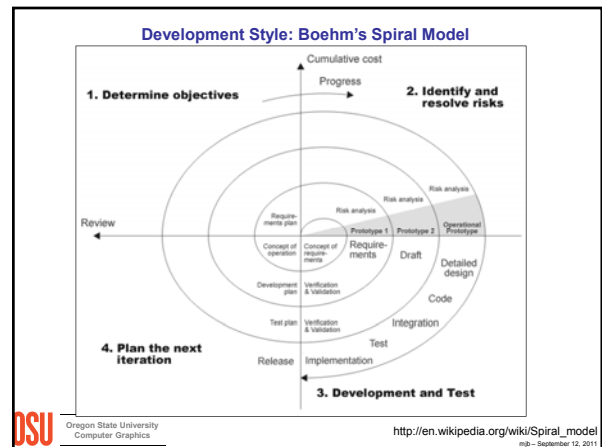
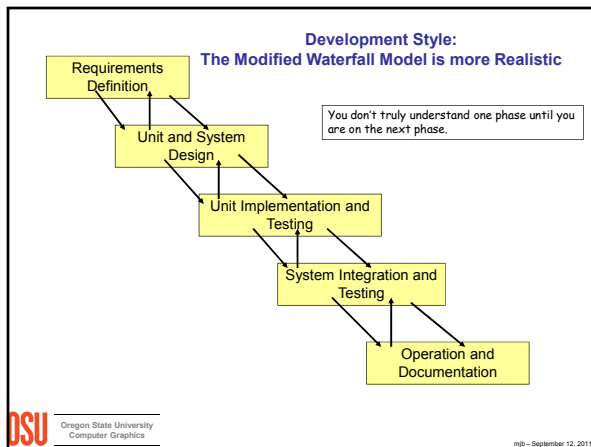
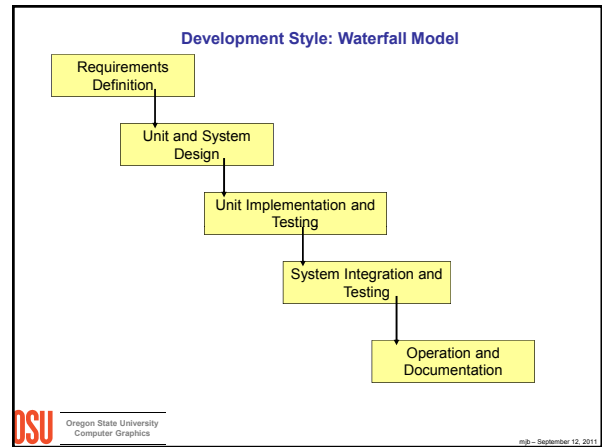
Note: "Agile Programming" does not mean "no plan", it means a continuously-changing plan under customer control

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

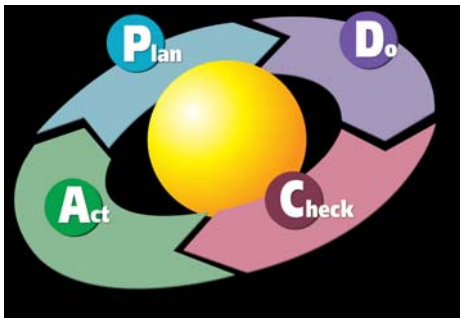
1. I want to know about it and be kept informed about how it is working
2. Let me know how you plan to always have the client present

If you can't guarantee continuous client feedback, you can't do it.

DSU Oregon State University Computer Graphics mjb - September 12, 2011



The Spiral Model Bears a Resemblance to Edwards Deming's PDCA Cycle

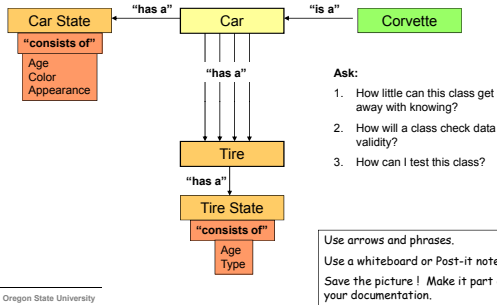


Development: Object-Oriented Programming Characteristics

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

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



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

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

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.

### 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.

OSU Oregon State University Computer Graphics [http://en.wikipedia.org/wiki/Myers-Briggs\\_Type\\_Indicator](http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator) 10/11/2011

### 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.

OSU Oregon State University Computer Graphics [http://en.wikipedia.org/wiki/Myers-Briggs\\_Type\\_Indicator](http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator) 10/11/2011

### 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.

OSU Oregon State University Computer Graphics [http://en.wikipedia.org/wiki/Myers-Briggs\\_Type\\_Indicator](http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator) 10/11/2011

### 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".

OSU Oregon State University Computer Graphics [http://en.wikipedia.org/wiki/Myers-Briggs\\_Type\\_Indicator](http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator) 10/11/2011

**ASCE American Society of Civil Engineers**



**The New Orleans Levees:  
The Worst Engineering Catastrophe in U.S. History –  
What Went Wrong and Why**

Lawrence H. Roth, P.E., G.E., F.ASCE  
Deputy Executive Director

OSU Oregon State University Computer Graphics 10/11/2011

### Ten Lessons Learned

1. **Failure to think globally** Account for issues that are beyond the bounds of the project
2. **Failure to absorb new knowledge** Establish mechanisms to incorporate changing information
3. **Failure to understand, manage, and communicate risk** Select an appropriate level of protection
4. **Failure to build quality in** Be sure project performance meets expectations of all stakeholders
5. **Failure to build resilience in** Avoid catastrophic failure
6. **Failure to provide redundancy** Think about what could go wrong and use a second line of defense where needed
7. **Failure to see that the sum of many parts ≠ a system** A chain is only as strong as its weakest link
8. **The buck couldn't find a place to stop** Make sure someone is in responsible charge
9. **Beware of the interfaces** Recognize that problems concentrate at the interfaces
10. **Follow the money** Don't let pressure for tradeoffs and low-cost solutions compromise quality, reliability, and safety

OSU Oregon State University Computer Graphics 10/11/2011

### Software Engineering References Used for These Notes

- 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.
- Frederick Brooks, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley, 2010.
- Robert Martin, *The Clean Coder*, Prentice-Hall, 2011.



Oregon State University  
Computer Graphics

m@ - September 12, 2011