As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
   - From your *bookmark*, or from the *Lessons* tab in Angel
5. Open the ***Slides*** for today if you wish

Software Development Processes

- Waterfall, Spiral, Agile
- Exercise using Waterfall:

the *Day of Year from Month/Day* problem

Strings

- String operations
- Lists and strings
- String encoding
- String formatting

Session 6

CSSE 120 – Fundamentals of Software Development

Outline – Software Development ***Process***, then ***Strings***

- **Software Development *Process***
  - Three historical processes:
    - Waterfall
    - Spiral
    - Agile
  - Exercise: *Day of Year from Month/Day*
    - Using Waterfall, appropriate for low complexity / short duration development like today's exercise

- ***Strings***
  - String operations
  - Lists and strings
  - String encoding
  - String formatting

Plus in-class time working on the above concepts, continued as homework.

Month, Day → Day of year

- We need a program that, given a particular month and day in that month, calculates the "ordinal value" (i.e., day of the year) for that month/day.
  - For example, given **March 6** as our month/day, our program should determine that it is the **65th** day of the year (in a non-leap year).
  - Banks often need this "ordinal value" to compute interest, penalties and so forth.

Modern software engineering uses:

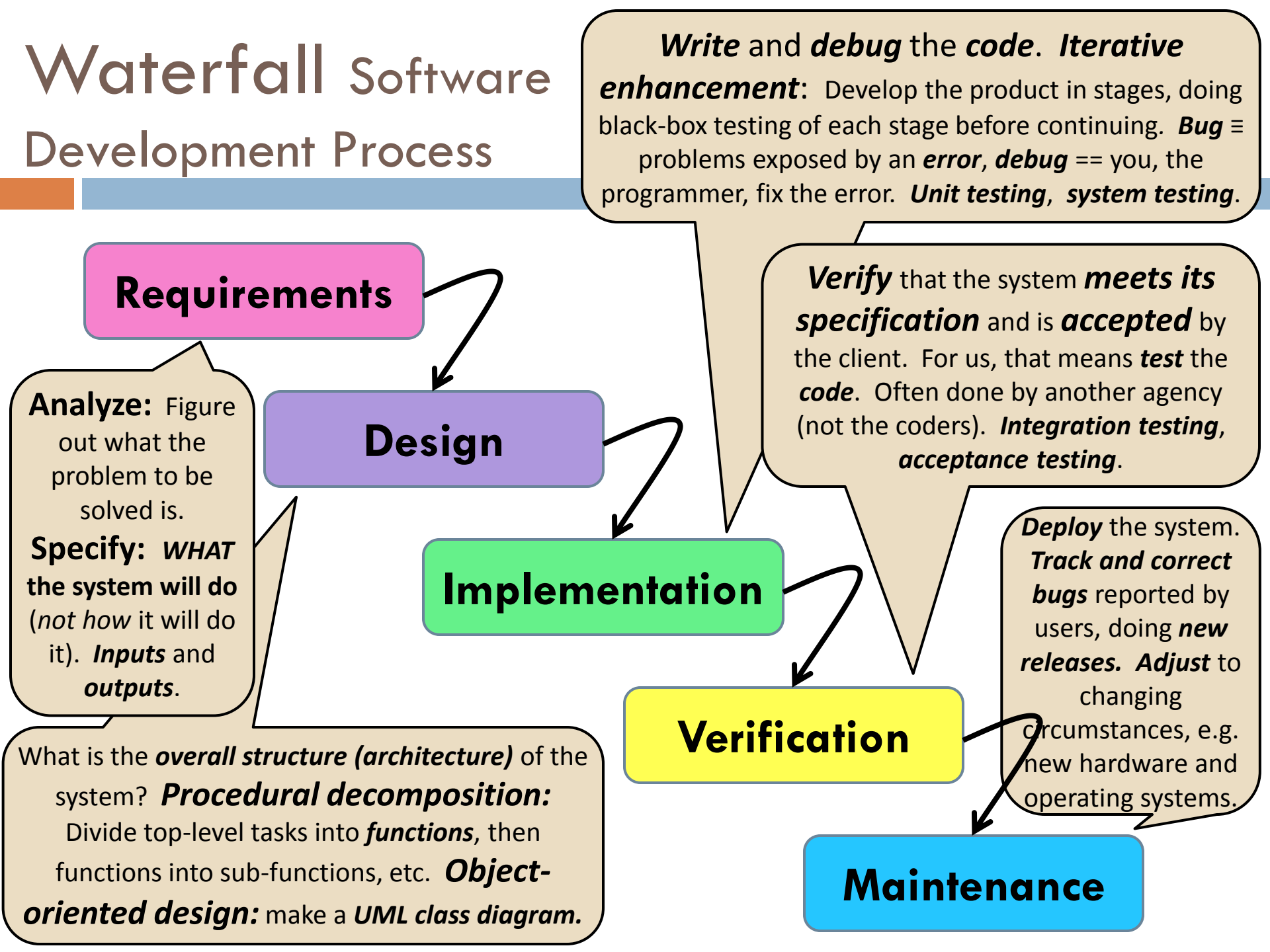
Powerful *TOOLS*, e.g.

- High-level languages and interpreters/compilers
- Powerful Integrated Development Environments (IDEs)
- Version Control systems
- Testing frameworks
- Diagramming applications
- Modeling languages
- Task management trackers

Powerful *PROCESSES*, e.g.

- Methodologies like:
  - Waterfall *Today's focus*
  - Spiral
  - Agile, e.g.
    - Extreme Programming
    - Scrum
- Sub-processes like:
  - Coding to a contract
  - Test-driven development

Waterfall Software Development Process



Requirements

- In the real world, you obtain requirements by interacting with the customers.
  - For our example problem (*day-of-year from month/day*), we are our own customers.
- **What is the problem that we are trying to solve?**
  - Given a particular month and day in that month, calculate the "ordinal value" (i.e., day of the year) for that month/day.
- **What is the specification of the problem? E.g.**
  - **What is the form of the input? Who supplies it?**
    - User supplies month (3-letter, lowercase) and day of month (integer). On separate lines, with prompts.
  - **What is the form of the output?**
    - Prints the right day of year, with an appropriate message that "echoes" the input.
  - **What additional constraints are there?**
    - We will NOT require handling leap years.
    - We will NOT do verification that the user enters legal inputs.
    - Real-world problems have additional constraints, like what hardware the system will require and how fast it must be developed, in what budget.

Q1

Design

- Sketch how that the program will do its work.
  - **Specification ≡ WHAT** the program will do
  - **Design ≡ HOW** the program will do it
- For our *day-of-year from month/day* problem, ask:
  - What application-domain data will we need (besides user input) to solve the problem?
    - Use ***two parallel lists*** as our in-program data:
      - One of month ***names*** ("Jan", "Feb", etc)
      - One of month ***lengths*** (31, 28, 31, 30, etc)
    - A better alternative would be to have a month *class*, and bundle each month's *name* and *length* inside the month *object*.
  - What ***algorithm*** shall we use to compute the answer?
    - Once we get the month and day as input, loop through the two lists in parallel, summing the lengths of the months.

Q2

Implement

- Checkout today's project from SVN:

  `06-StringsAndLists`

- We will work together to implement our solution in the module:

  `1-daysOfYear.py`

Design (repeated from previous slide):

What application-domain data will we need (besides user input) to solve the problem?

- Use ***two parallel lists*** as our in-program data:
  - One of month ***names*** ("Jan", "Feb", etc)
  - One of month ***lengths*** (31, 28, 31, 30, etc)

What ***algorithm*** shall we use to compute the answer?

- Once we get the month and day as input, loop through the two lists in parallel, summing the lengths of the months.

Are you in the Pydev perspective? If not:

- **`Window ~ Open Perspective ~ Other`**
  then **`Pydev`**

Messed up views? If so:

- **`Window ~ Reset Perspective`**

No SVN repositories view (tab)? If it is not there:

- **`Window ~ Show View ~ Other`**
  then **`SVN ~ SVN Repositories`**

In your SVN repositories view (tab), expand your repository *(the top-level item) if not already expanded.*

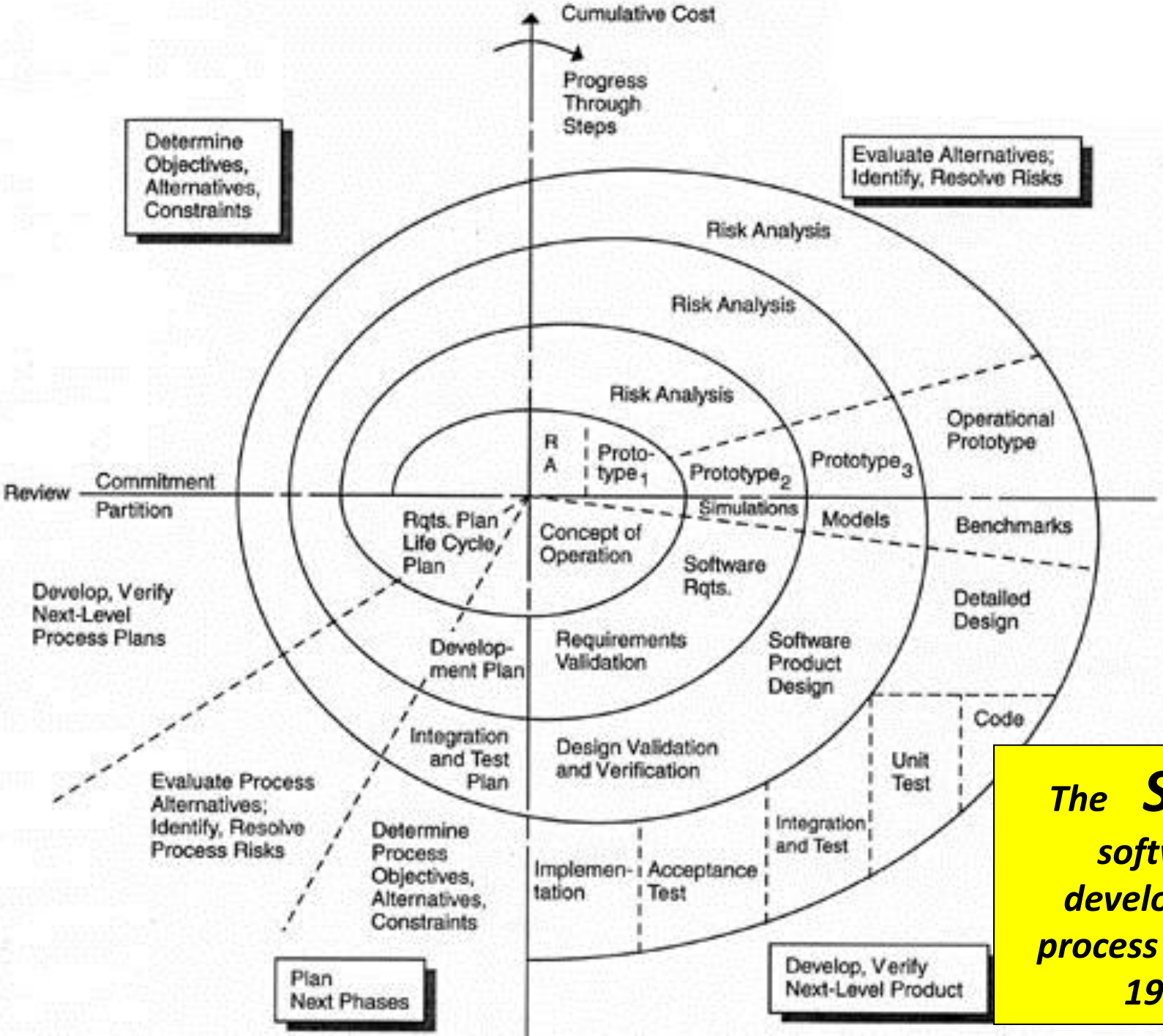
- If no repository, perhaps you are in the wrong Workspace. Get help as needed.

Right-click on today's project, *then select* ***`Checkout`****. Press* ***`OK`*** *as needed.*

The project shows up in the **`Pydev Package Explorer`** to the right. Expand and browse the modules under **`src`** as desired.

Software Development Processes – Waterfall

- We used Waterfall in our exercise
  - Perfectly fine for low complexity / short duration projects, but otherwise generally discredited
  - Other processes use the same Stages as Waterfall, however, in some form or another
- Next slides show two other Software Development Processes, just for your general education
  - Key in them (and all modern software development processes) is to iterate – back to the customer, then forward to reflect the changes the customer requires
  - No need to learn details of the next two slides
  - But important to recognize that for most real-world software development projects, Waterfall is NOT appropriate and iterative processes are used instead



The Spiral software development process (Boehm, 1988)

Agile software development processes

What is Agile Software Development?

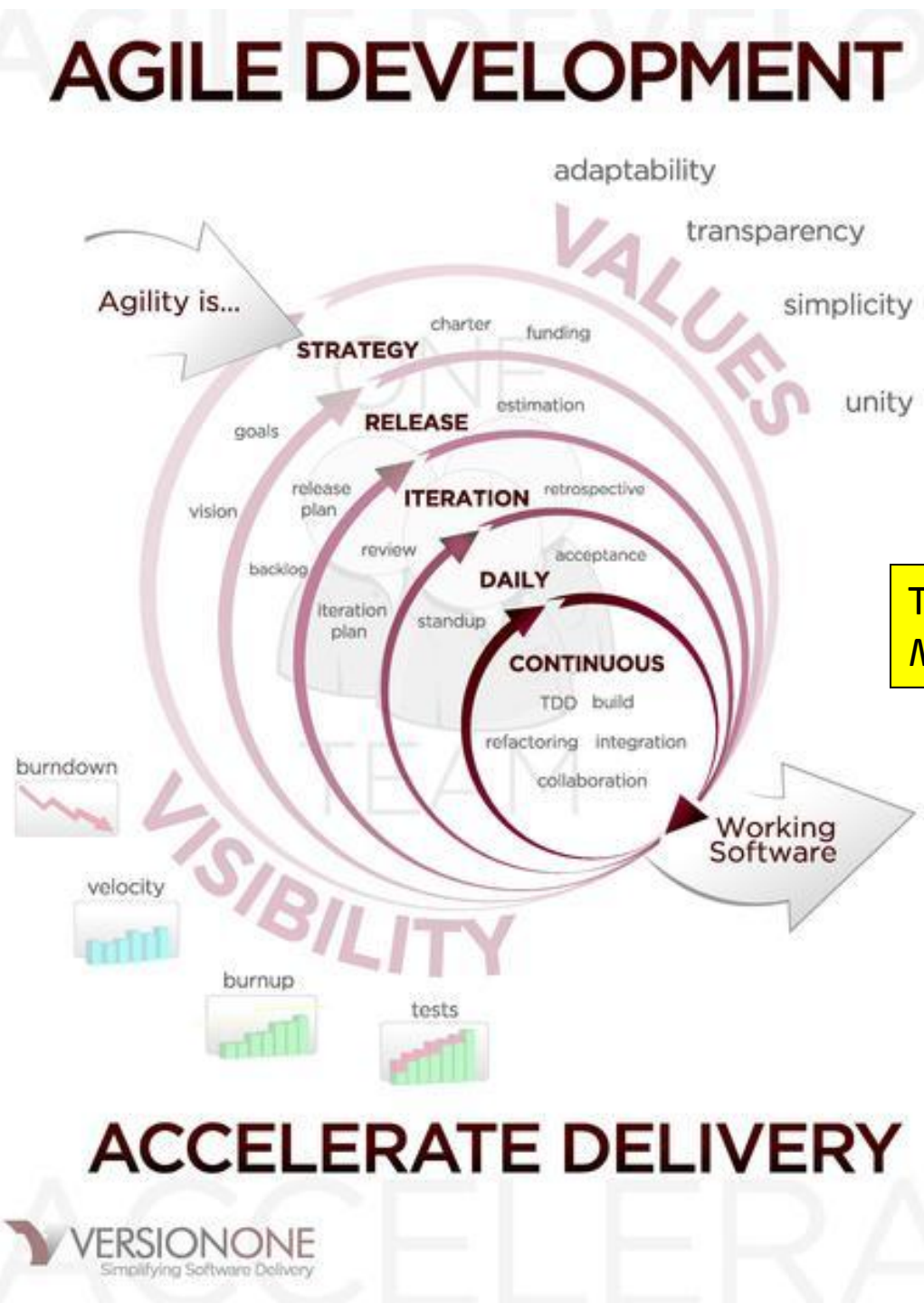
In the late 1990's several methodologies began to get increasing public attention. Each had a different combination of old ideas, new ideas, and transmuted old ideas. But they all emphasized close collaboration between the programmer team and business experts; face-to-face communication (as more efficient than written documentation); frequent delivery of new deployable business value; tight, self-organizing teams; and ways to craft the code and the team such that the inevitable requirements churn was not a crisis.

The Manifesto for Agile Software Development

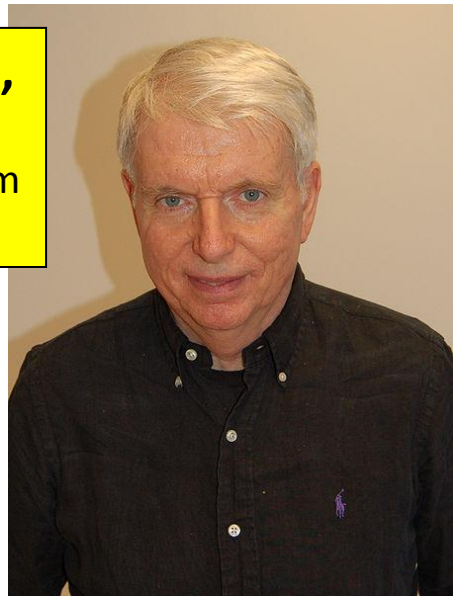
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

See http://www.agilealliance.org/ for fascinating details



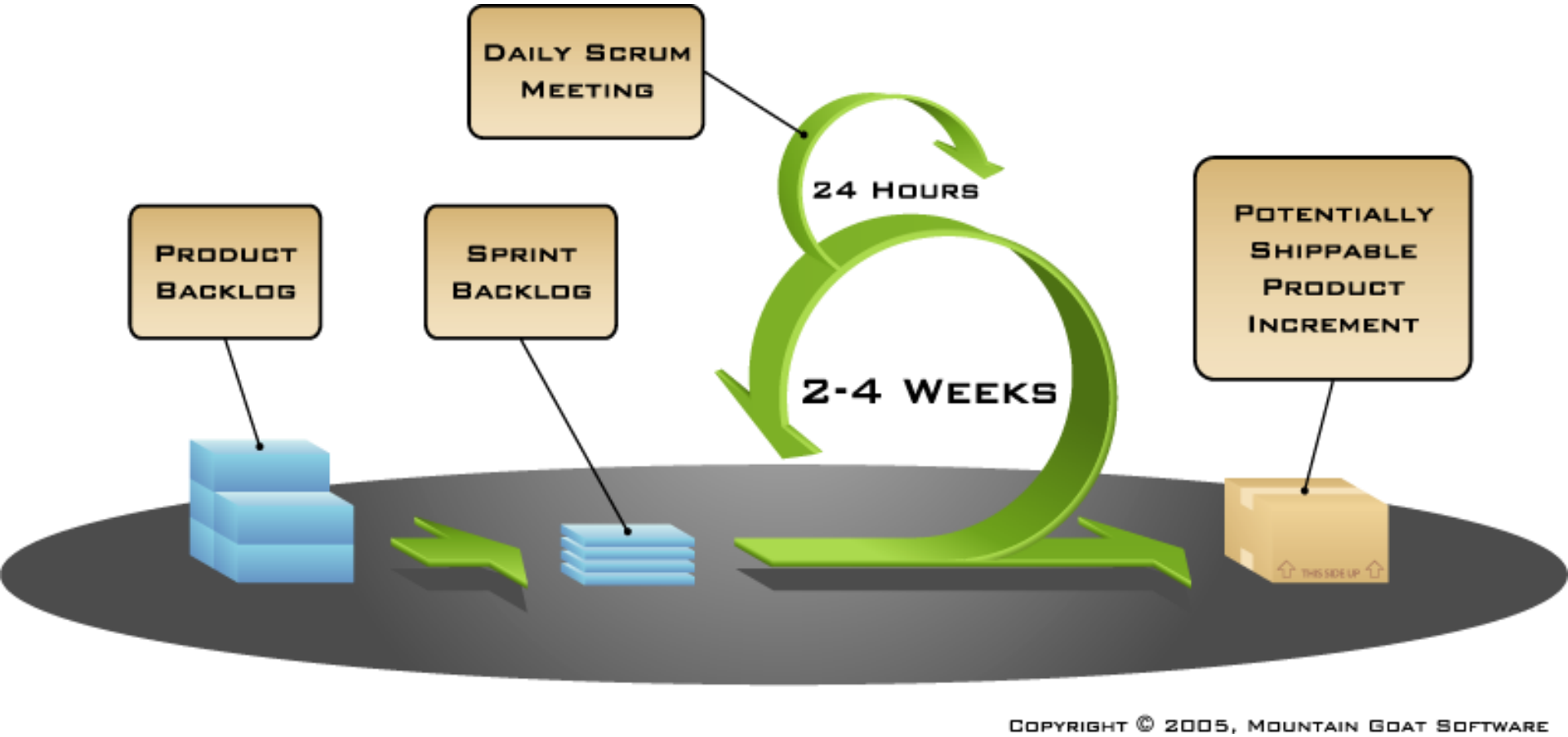
Jeff Sutherland, one of the developers of Scrum (see next slide).



Two of the 17 original signatories of the *Manifesto for Agile Software Development*

Kent Beck, co-author of the JUnit testing framework and a creator of the Extreme Programming and Test Driven Development methodologies.

*One popular **Agile** software development process – **Scrum***



Strings (sequences of characters)

- String literals (constants):
- `"One\nTwo\nThree"`
- `"Can't Buy Me Love"`
- `'I say, "Yes."  You say, "No." '`
- `"'A double quote looks like this \",' he said."`
- `"""I don't know why you say, "Goodbye,"
  I say "Hello." """`

Operating on Strings

Operations/Methods	What does each of these operation/method do?
s1 + s2	Concatenates two strings e.g. "xyz" + "abc" ⇒ "xyzabc"
*s * <int>*	Replicates string *s* <*int*> times e.g. "xyz" * 4 ⇒ "xyzxyzxyzxyz"
s.capitalize()	*Copy* of *s* with only 1st letter capitalized
s.lower()	*Copy* of *s* with all lower case characters
s.reverse()	*Copy* of *s* will all characters reversed
s.split()	List of the words in *s* (split on spaces by default)

Some more string methods

Methods	What does each of these operation/method do?
s.count(sub)	Returns the number of occurrences of *sub* in *s*
s.find(sub)	Returns the first position (index, 0-based) where *sub* occurs in *s*
s.title()	*Copy* of *s* with first character of each word capitalized
s.replace(old, new)	*Copy* of *s* where all occurrences of *old* in s have been replaced with *new*
s.lstrip()	*Copy* of s with leading white space removed
s.strip()	*Copy* of s with leading and trailing white space removed
s.join(list)	Concatenate *list* into a string, using *s* as the separator between items in the list

Practice with string operations

- Many of the operations listed in the book, while they work in Python 2.5, have been superseded by newer ones
- + is used for String concatenation: "xyz" + "abc"
- * is used for String duplication: "xyz " * 4
  - **>>> franklinQuote = 'Who is rich? He who is content. ' + 'Who is content? Nobody.'**
  - **>>> franklinQuote.lower()**

    'who is rich? he who is content. who is content? nobody.'
  - **>>> franklinQuote.replace('He', 'She')**

    'Who is rich? She who is content. Who is content? Nobody.'

    >>> franklinQuote.find('rich')

Q5-6

Strings are immutable sequences

- Lists are mutable:

```
colors = ["red", "white", "blue"]
colors[1] = "grey"
colors.append("cyan")
```

OK

colors becomes ["red", "grey", "blue"] then ["red", "grey", "blue". "cyan"]

- A string is an **immutable** sequence of characters

```
>>> building = "Taj Mahal"
>>> building[2]
>>> building[1:4]
>>> building[4] = "B"
```

NOT OK.
Gives an error message when executed.

Q7-8

Strings and Lists

- A String method: **split** breaks up a string into separate words
  - `>>> franklinQuote = 'Who is rich? He who is content. ' + 'Who is content?  Nobody.'`
  - `>>> myList = franklinQuote.split(' ')`
    `['Who', 'is', 'rich?', 'He', 'who', 'is', 'content.', 'Who', 'is', 'content?', 'Nobody.']`
- A string method: **join** creates a string from a list
  - `'#'.join(myList)`
  - `'Who#is#rich?#He#who#is#content.#Who#is#content?#Nobody.'`
- What is the value of `myList[0][2]`?
- Do exercise in **2-practiceWithStringsAndLists** module

Getting a string from the user

```
>>> name = input('Enter your name:')
Enter your name:John
>>> name
'John'
>>>
```

Q10

take a break

String Representation

- Computer stores 0s and 1s
  - Numbers stored as 0s and 1s
  - What about text?
- Text also stored as 0s and 1s
  - Each character has a code number
  - Strings are sequences of characters
  - Strings are stored as sequences of code numbers
  - Does it matter what code numbers we use?
- Translating: `ord(<char>)` `chr(<int>)`

Q11-12

Consistent String Encodings

- Needed to share data between computers, also between computers and display devices
- Examples:
  - ASCII—American Standard Code for Info. Interchange
    - "Ask-ee"
    - Standard US keyboard characters plus "control codes"
    - 8 bits per character
  - Extended ASCII encodings (8 bits)
    - Add various international characters
  - Unicode (16+ bits)
    - Tens of thousands of characters
    - Nearly every written language known

Q13

String Formatting

- Allows us to format complex output for display
  - It treats a string as a template with slots --- {}
  - Provided values are plugged into each slot
  - Uses a built-in method, format(), that takes values to plug into each slot
  - <template-string>.format(<values>)
- What does each slot look like?
  - {<index>:<format-specifier>}
  - <index> tells which of the parameters is inserted in slot
  - <format-specifier> describes how this slot will be formatted

Format Specifiers

- Syntax:
  - %<width>.<precision><typeChar>
- Width gives total spaces to use
  - 0 (or width omitted) means as many as needed
  - 0*n* means pad with leading 0s to *n* **total** spaces
  - -*n* means "left justify" in the *n* spaces
- Precision gives digits after decimal point, rounding if needed.
- TypeChar is:
  - f for float, s for string, or d for decimal (i.e., int) [can also use i]
- Note: this RETURNS a string that we can print
  - Or write to a file using write(string), as you'll need to do on the homework 6 assignment (HW6)

Q14-15, *turn in quiz*

Begin HW6

- Although you do not have a reading assignment and Angel quiz, you are strongly encouraged to begin working on your homework early.