“You’ve got to be very careful if you don’t know where you’re going, because you might not get there.”

Yogi Berra

Thanks to Professor Kirstie Hawkey for providing these slides!
There is no Silver Bullet

Primary thing to remember with SDLC methods!
It does NOT mean that there is no one method that will work in all cases
It means: “There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity”

SDLC Model

A framework that describes the activities performed at each stage of a software development project.
Aside: You Own a Construction Company

Not a small one – you build skyscrapers

Do you just start pouring concrete for the foundation or do you do a lot of planning? Why?
Aside: You Own a Construction Company

You may have built skyscrapers before, but each one is effectively custom built. Clients are paying a lot of money.

You’re paying a lot of money.

Materials and labor, among other costs.

Once you’ve started, it’s going to be very difficult to change things up.
Aside: You Own a Construction Company

What if the customer doesn’t like the finished product?

For that matter, how will the customer anticipate the finished product?

For that matter, how do you know what the customer wants in the first place?

They might actually want things that are different than what they think they want

And may not realize until it’s built
Aside: You Own a Construction Company

If you’ve built similar buildings before, then you might have a good idea of the time and cost.
Aside: You Own a Construction Company

But what if this building requires things that have never been done before?
Aside: You Own a Construction Company

But what if this building requires things that have never been done before?
Aside: You Own a Construction Company

But what if this building requires things that have never been done before?
Aside: You Own a **Software** Company

What makes you think that software doesn’t have many of the same issues?

It does. So when building large software, you had better have a good development model!

It’s no guarantee

But it greatly increases the likelihood of a successful project
Waterfall Model

**Requirements** – defines needed information, function, behavior, performance and interfaces.

**Design** – data structures, software architecture, interface representations, algorithmic details.

**Implementation** – source code, database, user documentation, testing.
Plan: How do we chart our entire course if we don’t know what’s ahead?

Build: Whatever happens, just keep paddling!

Test: I wish we’d designed for this scenario upfront

Launch: Patch it as best we can. No time to change course now
Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule
Waterfall Deficiencies

All requirements must be known upfront
Deliverables created for each phase are considered frozen — inhibits flexibility
Can give a false impression of progress
Does not reflect problem-solving nature of software development — iterations of phases
Integration is one big bang at the end
Little opportunity for customer to preview the system (until it may be too late)
When to use the Waterfall Model

Requirements are very well known
Product definition is stable
Technology is understood
New version of an existing product
Porting an existing product to a new platform.

High risk for new systems because of specification and design problems.
Low risk for well-understood developments using familiar technology.
V-Shaped SDLC Model

A variant of the Waterfall that emphasizes the verification and validation of the product. Testing of the product is planned in parallel with a corresponding phase of development.
V-Shaped Steps

Project and Requirements Planning – allocate resources

Product Requirements and Specification Analysis – complete specification of the software system

Architecture or High-Level Design – defines how software functions fulfill the design

Detailed Design – develop algorithms for each architectural component

Production, operation and maintenance – provide for enhancement and corrections

System and acceptance testing – check the entire software system in its environment

Integration and Testing – check that modules interconnect correctly

Unit testing – check that each module acts as expected

Coding – transform algorithms into software
V-Shaped Strengths

Emphasize planning for verification and validation of the product in early stages of product development

Each deliverable must be testable

Project management can track progress by milestones

Easy to use
V-Shaped Weaknesses

Does not easily handle concurrent events

Saw this 100 times. No one says why this is with any specificity

Does not handle iterations or phases

Does not easily handle dynamic changes in requirements

Like Waterfall model, not flexible

Does not contain risk analysis activities
When to use the V-Shaped Model

Excellent choice for systems requiring high reliability — hospital patient control applications

All requirements are known up-front

When it can be modified to handle changing requirements beyond analysis phase

Solution and technology are known
Protoyping: Basic Steps

Identify basic requirements
  Including input and output info
  Details (e.g., security) generally ignored
    That’s not a good thing, but not unique to prototyping
Develop initial prototype
  UI first
Review
  Customers/end – users review and give feedback
Revise and enhance the prototype & specs
  Negotiation about scope of contract may be necessary
Horizontal prototype

Broad view of entire system/sub-system
Focus is on user interaction more than low-level system functionality (e.g., database access)

Useful for:
- Confirmation of UI requirements and system scope
- **Demonstration version of the system** to obtain buy-in from business/customers
- Develop preliminary estimates of development time, cost, effort
Dimensions of Prototyping

Vertical prototype
More complete elaboration of a single sub-system or function
Useful for:
- Obtaining detailed requirements for a given function
- Refining database design
- Obtaining info on system interface needs
- Clarifying complex requirements by drilling down to actual system functionality
Throwaway/rapid/close-ended prototyping

Creation of a model that will be discarded rather than becoming part of the final delivered software

After preliminary requirements gathering, used to visually show the users what their requirements may look like when implemented

Focus is on quickly developing the model focus is not on good programming practices

Can Wizard of Oz things
Wizard of Oz Prototyping

Requires three things:

Script: tells what is to take place
Person: Acts as end user
Human “wizard”: simulates the end product

Person may not know that the “software” is actually a human simulating behavior

WOZ name comes from Toto pulling back curtain to reveal Wizard is actually a person pulling levers
Wizard of Oz Prototyping

Purpose is to improve user experience (UX)
Fidelity of Prototype

Low-fidelity

Paper/pencil
Mimics the functionality, but does not look like it

Often implemented with interpreted scripting language (e.g., Python)
Goal is not typically optimization at this stage
Fidelity of Prototype

Medium to High-fidelity

GUI builder

“Click dummy” prototype – looks like the system, but does not provide the functionality

Or provide functionality, but have it be general and not linked to specific data

http://www.youtube.com/watch?v=VGjcFouSlp4

http://www.youtube.com/watch?v=5oL1mNbxa4&feature=related
Throwaway Prototyping steps

Write preliminary requirements
Design the prototype
User experiences/uses the prototype, specifies new requirements
Repeat if necessary
Write the final requirements
Develop the real products
Evolutionary Prototyping

A.k.a breadboard prototyping (analogous to electronics breadboard)

Goal is to build a very robust prototype in a structured manner and constantly refine it.

The evolutionary prototype forms the heart of the new system and is added to and refined.

Allow the development team to add features or make changes that were not conceived in the initial requirements.
Evolutionary Prototyping
Evolutionary Prototyping Model

Developers build a prototype during the requirements phase.
Prototype is evaluated by end users.
Users give corrective feedback.
Developers further refine the prototype.
When the user is satisfied, the prototype code is brought up to the standards needed for a final product.
EP Steps

A preliminary project plan is developed
A partial high-level paper model is created
The model is source for a partial requirements specification
A prototype is built with basic and critical attributes
The designer builds
  the database
  user interface
  algorithmic functions
The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
This loop continues until the user is satisfied
EP Strengths

Customers can “see” the system requirements as they are being gathered.
Developers learn from customers.
A more accurate end product.
Unexpected requirements accommodated.
Allows for flexible design and development.
Steady, visible signs of progress produced.
Interaction with the prototype stimulates awareness of additional needed functionality.
Incremental prototyping

Final product built as separate prototypes
At the end, the prototypes are merged into a final design
Extreme Prototyping

Often used for web applications
Development broken down into 3 phases, each based on the preceding phase
1. Static prototype consisting of HTML pages
2. Screens are programmed and fully functional using a simulated services layer
   Fully functional UI is developed with little regard to the services, other than their contract
3. Services are implemented
Prototyping advantages

Reduced time and cost
Can improve the quality of requirements and specifications provided to developers
  Early determination of what the user really wants can result in faster and less expensive software

Improved/increased user involvement
User can see and interact with the prototype, allowing them to provide better/more complete feedback and specs
Misunderstandings/miscommunications revealed
Final product more likely to satisfy their desired look/feel/performance
Disadvantages of prototyping

Insufficient analysis

Focus on limited prototype can distract developers from analyzing complete project

Think of house with lots of "add ons"

May overlook better solutions

Conversion of limited prototypes into poorly engineered final projects that are hard to maintain

Limited functionality may not scale well if used as the basis of a final deliverable

May not be noticed if developers too focused on building prototype as a model
Disadvantages of prototyping 2

User confusion of prototype and finished system

Users can think that a prototype (intended to be thrown away) is actually a final system that needs to be polished. Unaware of the scope of programming needed to give prototype robust functionality.

Users can become attached to features included in prototype for consideration and then removed from final specification.

Especially problematic if those features turn out to be difficult/impossible to implement at production quality (e.g., required infrastructure unavailable).
Disadvantages of prototyping 3

Developer attachment to prototype

If spend a great deal of time/effort to produce, may become attached

Might try to attempt to convert a limited prototype into a final system

Bad if the prototype does not have an appropriate underlying architecture
Disadvantages of prototyping

Excessive development time of the prototype

Prototyping supposed to be done quickly
If developers lose sight of this, can try to build a prototype that is too complex
For throw away prototypes, the benefits realized from the prototype (precise requirements) may not offset the time spent in developing the prototype — expected productivity reduced
Users can be stuck in debates over prototype details and hold up development process
Disadvantages of prototyping

Expense of implementing prototyping
Start up costs of prototyping may be high
Expensive to change development methodologies in place (re-training, re-tooling)
Slow development if proper training not in place
  High expectations for productivity unrealistic if insufficient recognition of the learning curve
Lower productivity can result if overlook the need to develop corporate and project specific underlying structure to support the technology
**Best uses of prototyping**

Most beneficial for systems that will have many interactions with end users.

The greater the interaction between the computer and the user, the greater the benefit of building a quick system for the user to play with.

Especially good for designing good human–computer interfaces.
Aside: Generation Languages

First generation (1gl): Machine language

2gl: Low-level assembly language: hardware dependent

3gl: High-level languages: C, C++, Java, Javascript, Visual Basic

4gl: Statements similar to statements in a human language: Perl, Python, PHP, Ruby, SQL

5gl: Programming languages that contain visual tools to help develop a program: Mercury, OPS5, Prolog
AGILE SOFTWARE DEVELOPMENT LIFE CYCLES
Agile SDLC’s

Speed up or bypass one or more life cycle phases
Usually less formal and reduced scope
Used for time-critical applications
Used in organizations that employ disciplined methods
Some Agile Methods

- Rapid Application Development (RAD)
- Incremental SDLC
- Scrum
- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rational Unify Process (RUP)
Agile vs Waterfall Propaganda

https://www.youtube.com/watch?v=CKD9nWVsDzc
RAPID APPLICATION DEVELOPMENT (RAD) MODEL
RAD is not Rapid Prototyping

Rapid application development (RAD) is a method for rapidly developing the final product.

As the title implies, you are rapidly developing the application.

Rapid prototyping uses a throwaway prototype in order to better learn the needs/requirements of the user.
TRADITIONAL

ANALYSIS → HIGH-LEVEL DESIGN → DETAILED DESIGN → CONSTRUCTION → TESTING → IMPLEMENTATION

RAD

ANALYSIS AND QUICK DESIGN → PROTOTYPE CYCLES → BUILD → REFINISH → DEMONSTRATE → TESTING → IMPLEMENTATION
Rapid Application Model (RAD)

Requirements planning phase  (a workshop utilizing structured discussion of business problems)

User description phase  — automated tools capture information from users

Construction phase  — productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)

Cutover phase  -- installation of the system, user acceptance testing and user training
Aside: Timeboxing

Timeboxing is a planning technique common in planning projects, where the schedule is divided into a number of separate time periods (timeboxes, normally two to six weeks long), with each part having its own deliverables, deadline and budget.
Aside: Timeboxing

- Timeboxes are used as a form of risk management, especially for tasks that may easily extend past their deadlines. The end date (deadline) is one of the primary drivers in the planning and should not be changed as it is usually linked to a delivery date of the product. If the team exceeds the deadline, the team failed in proper planning and/or effective execution of the plan. This can be the result of: the wrong people on the wrong job (lack of communication between teams, lack of experience, lack of commitment/ drive/motivation, lack of speed) or underestimation of the complexity of the requirements.

- When the team exceeds the deadline, the following actions might be taken after conferring with the Client:
  
  Dropping requirements of lower impact (the ones that will not be directly missed by the user)
  Working overtime to compensate for the time lost
  Moving the deadline
Requirements Planning Phase

Combines elements of the system planning and systems analysis phases of the System Development Life Cycle (SDLC).

Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements.

It ends when the team agrees on the key issues and obtains management authorization to continue.
User Design Phase

Users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. Typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models.

A continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
JAD Techniques

http://en.wikipedia.org/wiki/Jo\nint\_application\_design

CASE Tools

http://en.wikipedia.org/wiki/Co\nmputer-\naided\_software\_engineering
Construction Phase

Focuses on program and application development task similar to the SDLC. However, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration, and system testing.
Cutover Phase

Resembles the final tasks in the SDLC implementation phase. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner. Tasks are data conversion, full-scale testing, system changeover, user training.
RAD Strengths

Reduced cycle time and improved productivity with fewer people means lower costs

Time-box approach mitigates cost and schedule risk

Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs

Focus moves from documentation to code (WYSIWYG).

Uses modeling concepts to capture information about business, data, and processes.
RAD Weaknesses

Accelerated development process must give quick responses to the user.
Risk of never achieving closure.
Hard to use with legacy systems.
Requires a system that can be modularized.
Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.
When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized
Incremental SDLC Model

Construct a partial implementation of a total system
Then slowly add increased functionality
The incremental model prioritizes requirements of the system and then implements them in groups.
Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.
Incremental Model Strengths

Develop high-risk or major functions first

Each release delivers an operational product

Customer can respond to each build

Uses “divide and conquer” breakdown of tasks

Lowers initial delivery cost

Initial product delivery is faster

Customers get important functionality early

Risk of changing requirements is reduced
Incremental Model Weaknesses

Requires good planning and design
Requires early definition of a complete and fully functional system to allow for the definition of increments

Well-defined module interfaces are required (some will be developed long before others)

Total cost of the complete system is not lower
When to use the Incremental Model

Risk, funding, schedule, program complexity, or need for early realization of benefits.
Most of the requirements are known up-front but are expected to evolve over time
A need to get basic functionality to the market early
On projects which have lengthy development schedules
On a project with new technology
SCRUM
Scrum:

Sprint Backlog

Product Backlog
As prioritized by Product Owner

Daily Scrum Meeting

Backlog tasks expanded by team

24 hours

30 days

Demonstrable new functionality

Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.
Scrum in 13 seconds:
http://www.youtube.com/watch?v=9DKM9HcRnZ8&feature=related

Scrum in 10 minutes:
https://www.youtube.com/watch?v=XU0llRltyFM

More Scrum Slides

Scalability of scrum addressed on slides 33-35
Aside: User Stories

Informal, general explanation of a software feature
Written from perspective of the software user
Articulates how feature will provide value to the customer
Not software systems requirements

Thanks to: Max Rehkopf, “User Stories With Examples and Templates”
https://www.atlassian.com/agile/project-management/user-stories
Aside: User Stories

User story effectively puts end user at the center of the development conversation.
Non-technical language provides context for the development team.

The team learns why they are building a feature, what they are building, and the value it creates.
User Story Template

“As a [persona], I [want to], [so that].”

“As a [persona]”: Who are we building this for? We’re not just after a job title, we’re after the persona of the person. Max. Our team should have a shared understanding of who Max is. We’ve hopefully interviewed plenty of Max’s. We understand how that person works, how they think and what they feel. We have empathy for Max.
User Story Template

“Wants to”: Here we’re describing their intent – not the features they use. What is it they’re actually trying to achieve? This statement should be implementation free – if you’re describing any part of the UI and not what the user goal is you're missing the point.
User Story Template

“So that”: how does their immediate desire to do this fit into their bigger picture? What’s the overall benefit they’re trying to achieve? What is the big problem that needs solving?
User Story Examples

- As Max, I want to invite my friends, so we can enjoy this service together.
- As Sascha, I want to organize my work, so I can feel more in control.
- As a manager, I want to be able to understand my colleagues progress, so I can better report our success and failures.
When Team Decides to include story in a sprint...

- Discuss functionality and requirements the story requires
  This is technical
- Requirements added to the story
- Often story scored on complexity
- Story broken into smaller pieces, if necessary, to fit in sprint
- Determine what “done” means, time to completion, etc.
Scrum advantages

Agile scrum helps the company in saving time and money.
Scrum methodology enables projects where the business requirements documentation is hard to quantify to be successfully developed.
Fast moving, cutting edge developments can be quickly coded and tested using this method, as a mistake can be easily rectified.
Scrum advantages

It is a lightly controlled method which insists on frequent updating of the progress in work through regular meetings. Thus there is clear visibility of the project development.

Like any other agile methodology, this is also iterative in nature. It requires continuous feedback from the user.

Due to short sprints and constant feedback, it becomes easier to cope with the changes.
Scrum advantages

Daily meetings make it possible to measure individual productivity. This leads to the improvement in the productivity of each of the team members.

Issues are identified well in advance through the daily meetings and hence can be resolved speedily. It is easier to deliver a quality product in a scheduled time.
Scrum advantages

Agile Scrum can work with any technology/programming language but is particularly useful for fast moving web 2.0 or new media projects. The overhead cost in terms of process and management is minimal thus leading to a quicker, cheaper result.
Scrum disadvantages

Agile Scrum is one of the leading causes of scope creep because unless there is a definite end date, the project management stakeholders will be tempted to keep demanding new functionality. If a task is not well defined, estimating project costs and time will not be accurate. In such a case, the task can be spread over several sprints. If the team members are not committed, the project will either never complete or fail.
Scrum disadvantages

It is good for small, fast moving projects as it works well only with small team. This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed in time.

Scrum works well when the Scrum Master trusts the team they are managing. If they practice too strict control over the team members, it can be extremely frustrating for them, leading to demoralisation and the failure of the project.
Scrum disadvantages

If any of the team members leave during a development it can have a huge adverse effect on the project development. Project quality management is hard to implement and quantify unless the test team are able to conduct regression testing after each sprint.
Regression Testing

Actually should be called (and rarely is) NON-regression testing

Rerunning tests on previously developed and tested modules to ensure that they still perform after a change

If not, that is a regression