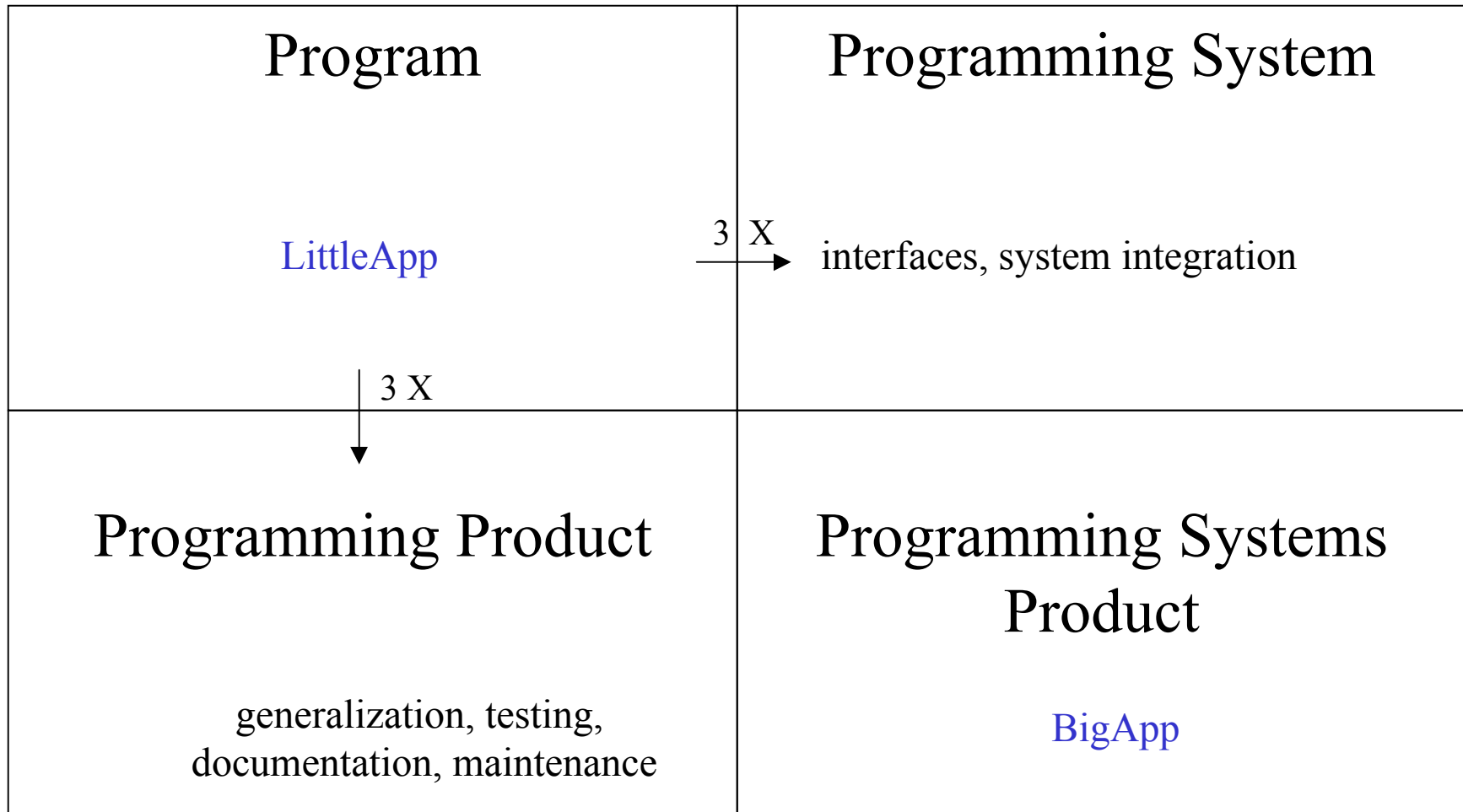

LittleApp to BigApp

CSE 403, Spring 2003
Software Engineering

<http://www.cs.washington.edu/education/courses/403/03sp/>

Readings and References

- Chapter 19, Designing for Change, *Rapid Development*, McConnell
- *Perfection and Simplicity*, and *Designing Distributed Systems*, from A Conversation with Ken Arnold, by Bill Venners
 - » <http://www.artima.com/intv/perfect.html>
 - » <http://www.artima.com/intv/distrib.html>



from Mythical Man-Month

From LittleApp to BigApp

- LittleApp prototypes can show that the basic concepts are workable
- Likely open issues
 - » Correctness - dummy data
 - » Completeness - inflexible sources, usability
 - » Robustness - frustrating response to errors
 - » Style - design, generalization, documentation

Design issues

- Interfaces
 - » What are the defined interfaces?
 - » Which fundamental decisions cannot be changed and still use the same architecture?
- Modules
 - » What are the major modules using those interfaces?
 - » Can fundamental design decisions in one module be changed without affecting the other modules?
- Documentation

Designing for Change

- Change happens
 - » underlying technology changes, a performance goal is not met, new requirements are levied
 - » perhaps the product is a success and lives for a decade or two!
- A successful design
 - » hides the implementation decisions
 - » can change locally without causing ripples throughout the entire structure

Not a single tool, but an approach

- Identify areas likely to change
- Use information hiding to conceal the design decisions
- Develop a change plan
- Define families of programs
- Use object-oriented design

from McConnell, Chap 19

What might change?

- Hardware for sure - many possible platforms
- File formats - how many graphics formats?
- Inputs and outputs, user's natural language
- Non-standard language features, libraries
- Features that are difficult to implement (AWT)
- Global variables
- Specific data structures and abstract data types
- Business rules, sequence of actions
- Requirements that were excluded, new features

Implementation is not just a detail

- What is important to keep in mind when you are designing a distributed system?
 - » A distributed system, in the sense in which I take any interest, means a system in which the failure of an unknown computer can screw you.
 - » Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure.

from Designing Distributed Systems, A Conversation with Ken Arnold, by Bill Venners

Develop a change plan

- Use abstract interfaces first, then classes
- Never use hardcoded literals
- Use late binding strategies
 - » dynamic allocation of data structures
 - » let the data structure tell you how big it is
- Use table driven strategies
 - » `getAppProperty(String key)` midlet jad file
 - » `getInitParameter(String name)` servlet web.xml file
 - » servlet name to class mapping in web.xml

More change plan

- Don't duplicate code or state
 - » put it in a single method and call it when needed
- Keep the methods and classes simple and cohesive
 - » easier to reuse or use in a new way
- Avoid coupling
- Keep the general purpose layers free of implementation leakage from below

Define families of programs


- What are the change vectors?
- If your product is a success, where will it go next?
 - » international? - language, currency, measurement
 - » system scale? - cell, PDA, desktop browser, server
 - » product distribution? - corporate, personal retail, educational, ad supported, free “lite”
- Think about the minimal subset of functions needed in all versions and how to present it

Perfection and Simplicity

- I once heard you say there is no such thing as a perfect design. Could you clarify what you meant by that?
- There is no such thing as a perfect design for a couple of reasons.
 - » All designs take place in context ... who will be using your design? ... if you try to create a perfect design you will expend a huge amount of effort ... then there's the problem of predicting the future.
- The best that people can reasonably hope for is to put forth an appropriate amount of effort and get a good design that is sufficient.

from Perfection and Simplicity, A Conversation with Ken Arnold, by Bill Venners

Now build it!

- Bad design leads you down the wrong road
- Bad construction takes you down a road full of potholes and bone-jarring problems
- Good construction techniques
 - » help build in quality the first time
 - » avoid having to back up and start over
 - » provide good visibility on how it's going without using made-up numbers
 - “we're 96% done” 

Some construction fundamentals

- Agreed-on coding standards
 - » naming, layout, documentation
- Data-related concepts
 - » scope, persistence, binding times
- Control-related
 - » complexity, control structures, exceptions
- Errors and exceptions
 - » assertions, defining and handling exceptions

More construction fundamentals

- Integration strategies
 - » Unit-testing and debugging
 - » Build and packaging practices
- Code tuning and performance measurement
- Programming tools
 - » editors, IDE, interoperability
 - » group work support tools (email, change visibility)
 - » source code revision management
 - » bug tracking

<http://java.sun.com/people/arnold/>

Someday this will
no doubt be a
vivacious page,
brimming with
fascinating
information,
penetrating
analysis, and the
most succulent links. Until then, this will have to do.

