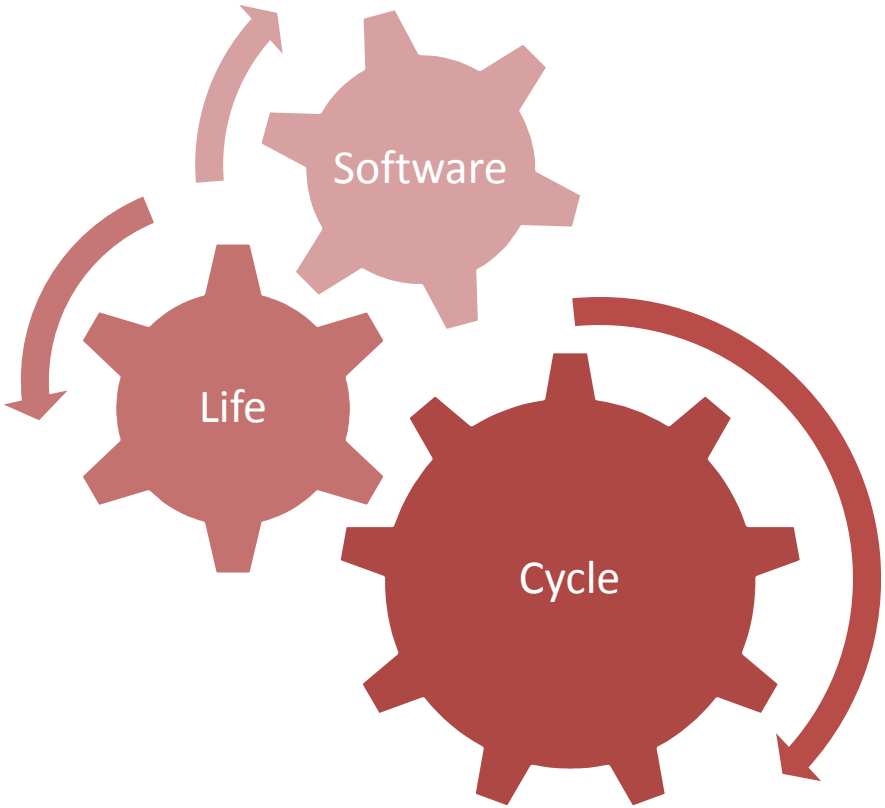
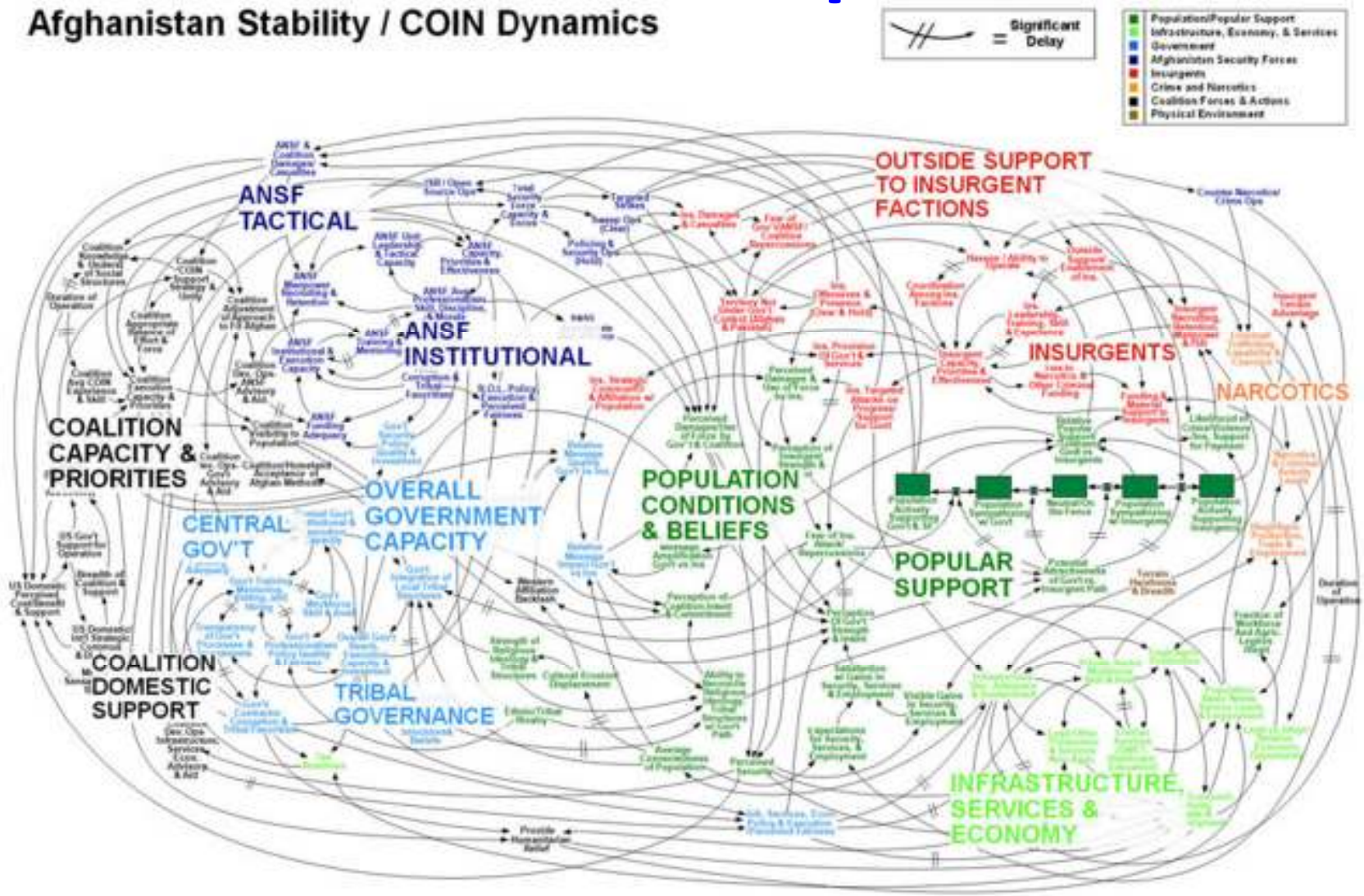


Software development lifecycle

The power of process

How complex is software?

What is complex?



How complex is software?

- Measures of complexity:
  - lines of code
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors
  - … many more

How complex is software?

- Measures of complexity:
  - lines of code
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors
  - … many more

Windows Server 2003: 50 MSLoC
Debian 5.0: 324 MSLoC

How big is 324 MSLoC?

- 50 lines/page $\Rightarrow$ 6.5M pages
- 1K pages/ream $\Rightarrow$ 6.5K reams
- 2 inches/ream $\Rightarrow$ 13K inches
- 13K inches $\approx$ 13x the height of the Allen Center

- 5 words/LoC @ 50 wpm $\Rightarrow$ 32M min $\approx$ 61 years

Just to type!
No breaks and no thinking allowed!

Addressing software complexity

What are/is the …?

- Requirements
- Design
- Implementation
- Testing plan
- …

Who does the …?

- Requirements
- Design
- Implementation
- Testing
- …

- Two sides of the same coin
- Different approaches, representations, etc. are needed for the artifact-oriented components
- Different skill-sets, knowledge, etc. are needed for the human-oriented components

Outline

- What is a software development lifecycle?
- Why do we need a lifecycle process?
- Lifecycle models and their tradeoffs
  - "Code-and-fix"
  - Waterfall
  - Spiral
  - Evolutionary prototyping
  - Staged delivery
  - Agile (XP, scrum, …)

  … many others

Lifecycle stages

- Virtually all lifecycles share these steps/stages/phases:
  - Requirements
  - Design
  - Implementation
  - Testing
  - Maintenance
- Key question: how do you combine them, and in what order?

Ad-hoc development

- Ad-hoc development: creating software without any formal guidelines or process
- Advantage: easy to learn and use!
- Disadvantages?
  - may ignore some important tasks (testing, design)
  - not clear when to start or stop doing each task
  - scales poorly to multiple people
  - hard to review or evaluate one's work
  - code may not match user's needs (no requirements!)
  - code was not planned for modification, not flexible

The later a problem is found in software, the more costly it is to fix.

The software lifecycle

- Software lifecycle: series of steps / phases, through which software is produced
  - from conception to end-of-life
  - can take months or years to complete

- Goals of each phase:
  - mark out a clear set of steps to perform
  - produce a tangible item
  - allow for review of work
  - specify actions to perform in the next phase

Some lifecycle models

- **code-and-fix**: write some code, debug it, repeat (i.e., *ad-hoc*)
- **waterfall**: standard phases (req., design, code, test) in order
- **spiral**: assess risks at each step; do most critical action first
- **evolutionary prototyping**: build an initial small requirement spec, code it, then "evolve" the spec and code as needed
- **staged delivery**: build initial requirement specs for several releases, then design-and-code each in sequence

Benefits of using a lifecycle

- It provides us with a structure in which to work
- It forces us to think of the “big picture” and follow steps so that we reach it without glaring deficiencies
- Without it you may make decisions that are individually on target but collectively misdirected
- It is a management tool

Drawbacks?

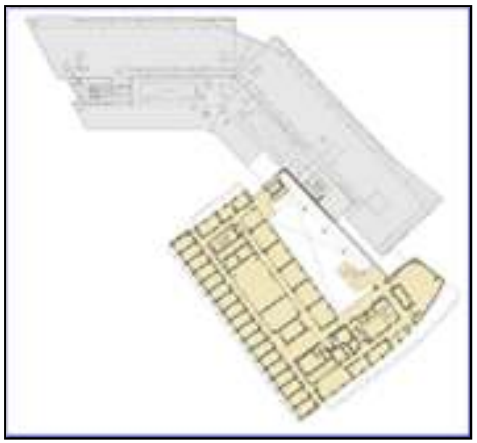
Limitations of lifecycle models

- Can lead to compromises and artificial constraints
- Risk of overemphasizing process (not the end in itself)

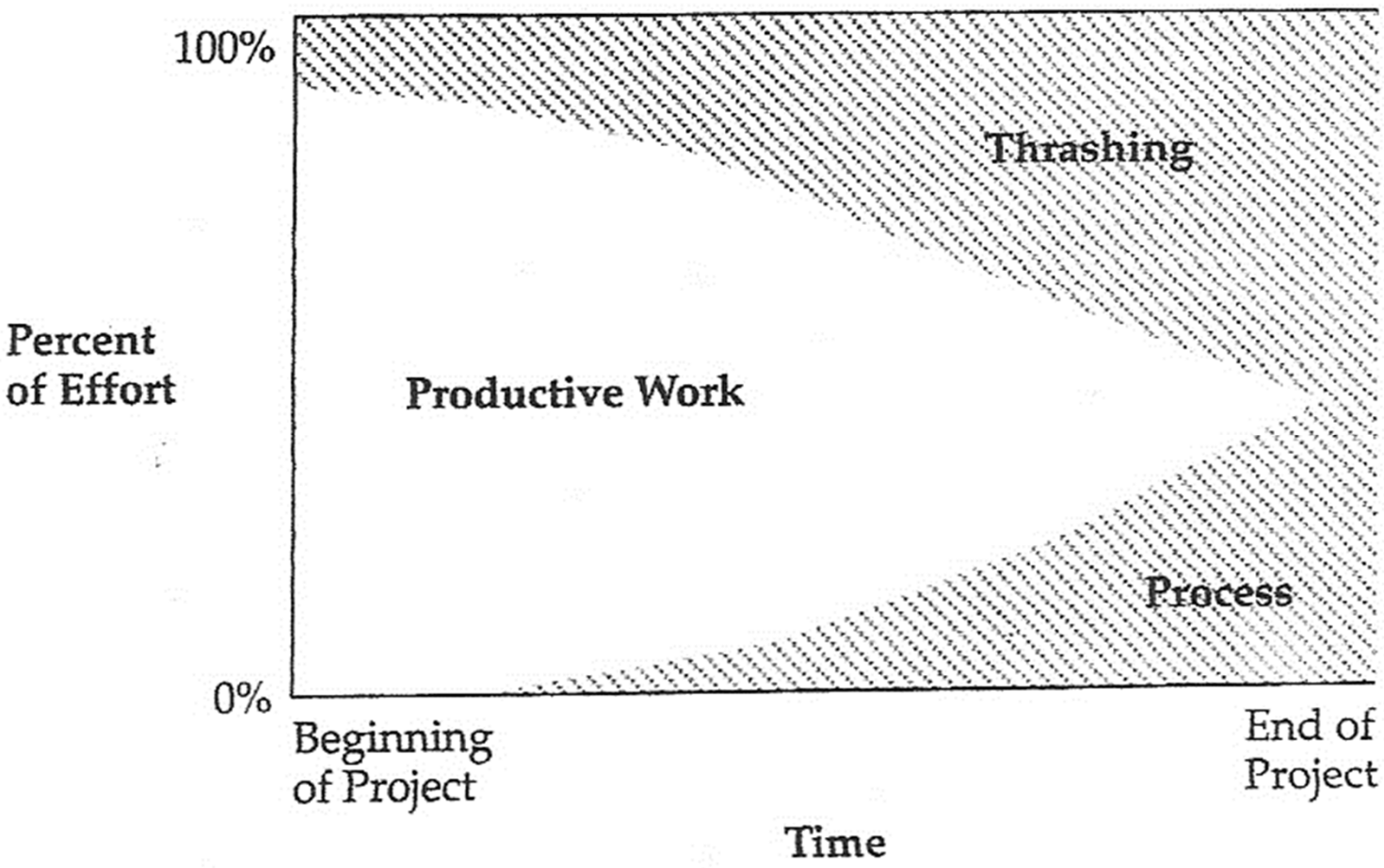
- Ways of evaluating models
  - risk management, quality/cost control, predictability, visibility of progress, customer involvement/feedback

Are there analogies outside of SE?

Consider the process of building the Paul Allen Center

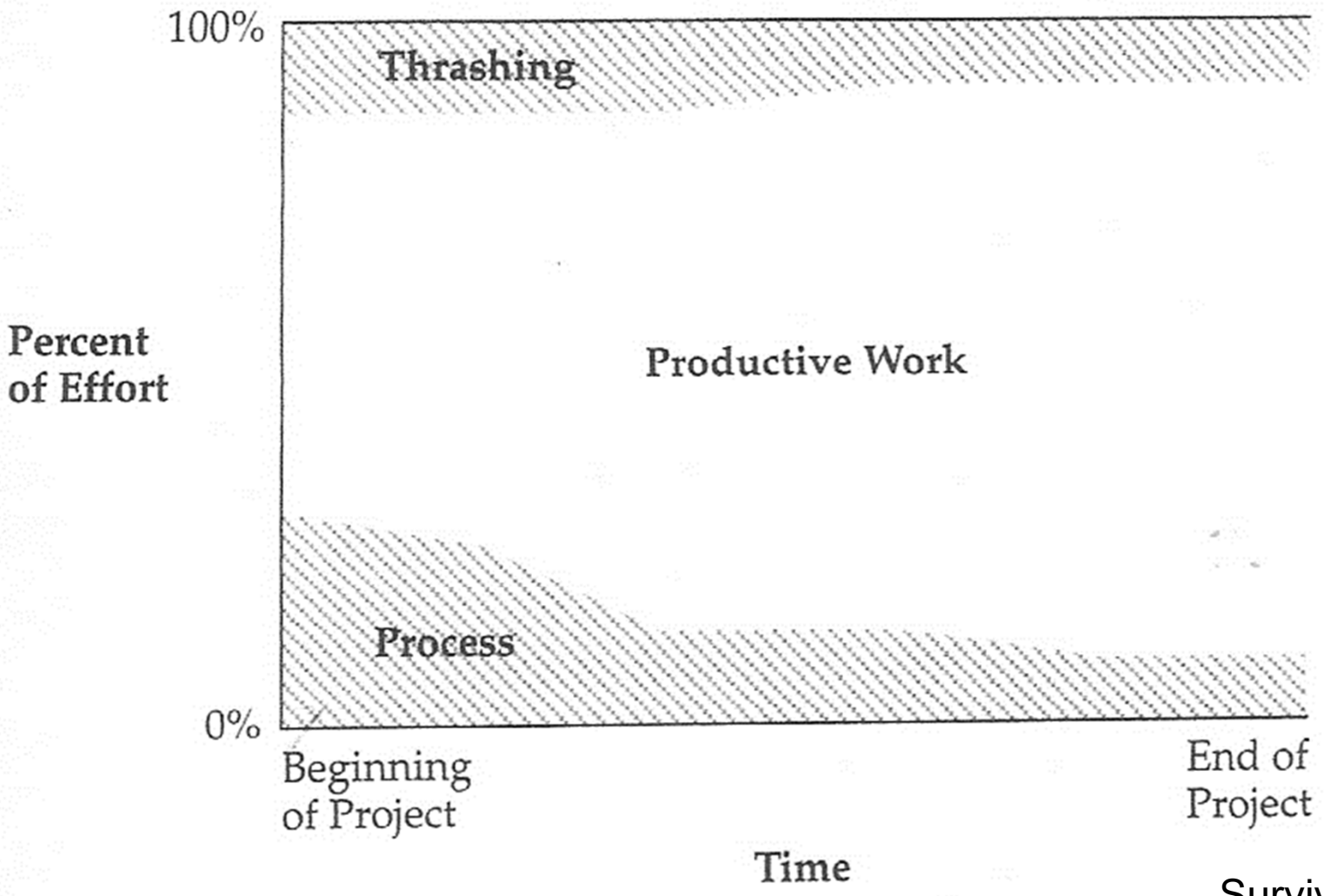


Project with little attention to process



Survival Guide:
McConnell p24

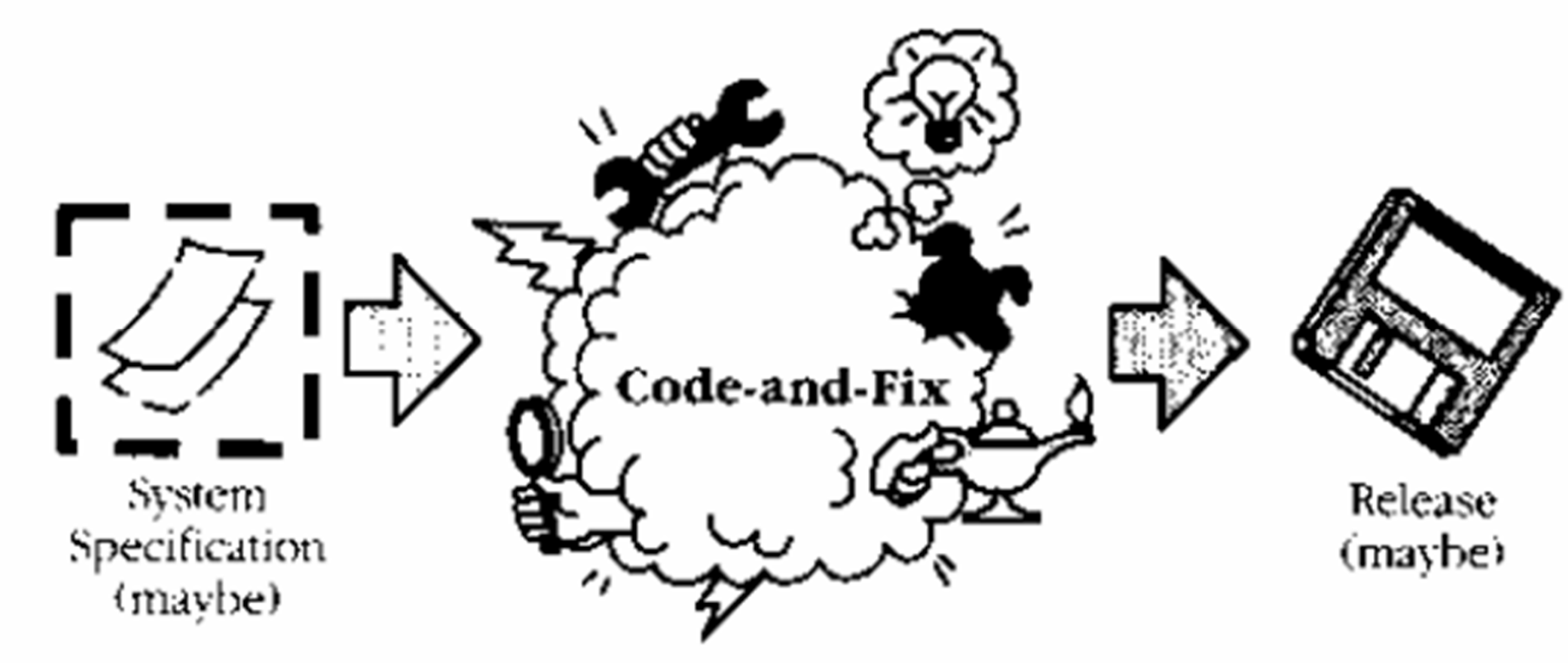
Project with early attention to process



Survival Guide:
McConnell p25

Let’s talk about some lifecycle models

Code-and-fix model



Code-and-fix model

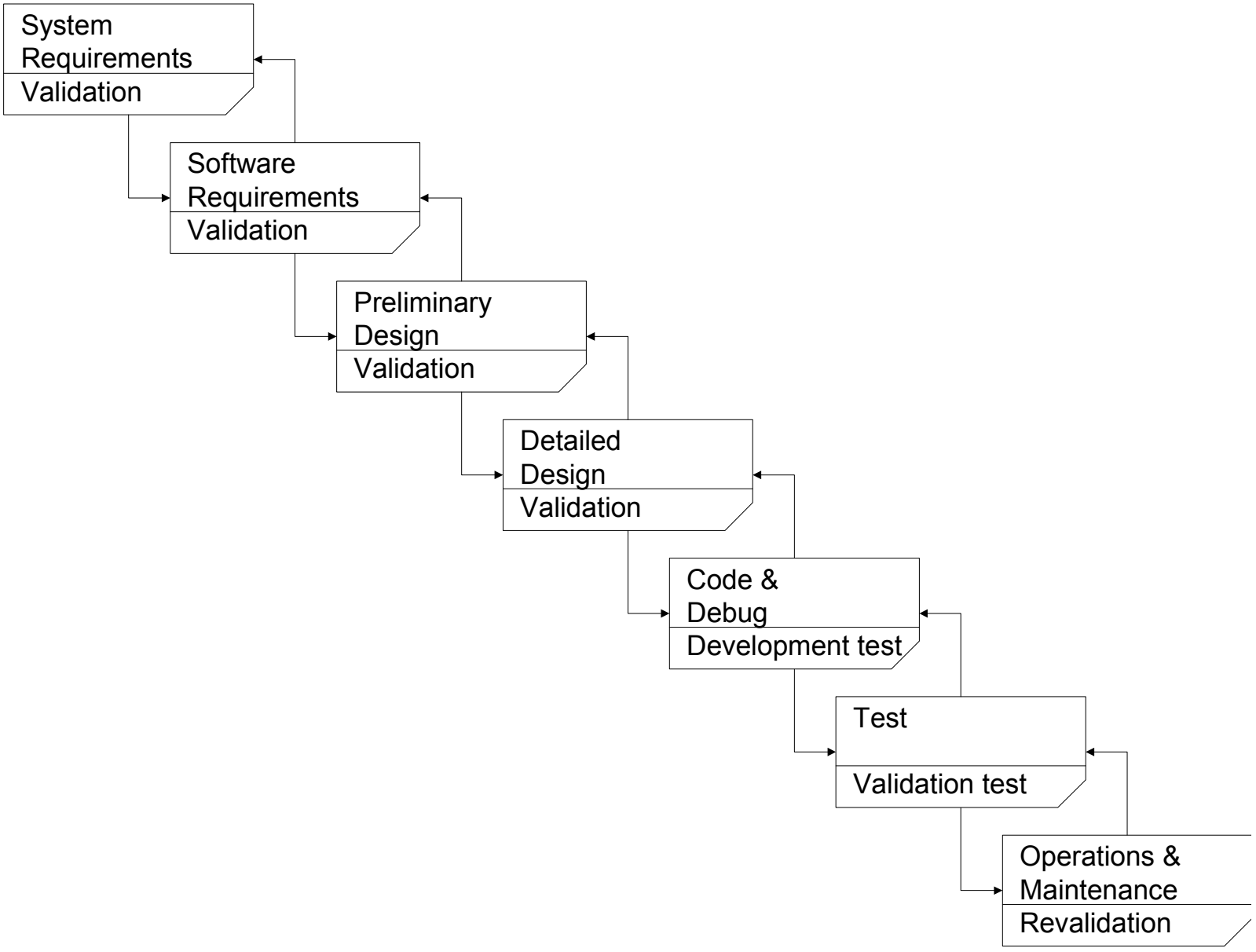
Advantages

- Little or no overhead
  - just dive in and develop, and see progress quickly
- Applicable *sometimes* for very small projects and short-lived prototypes

But DANGEROUS for most projects

- No way to assess progress, quality or risks
- Unlikely to accommodate changes without a major design overhaul
- Unclear delivery features (scope), timing, and support

Waterfall model



Waterfall model advantages

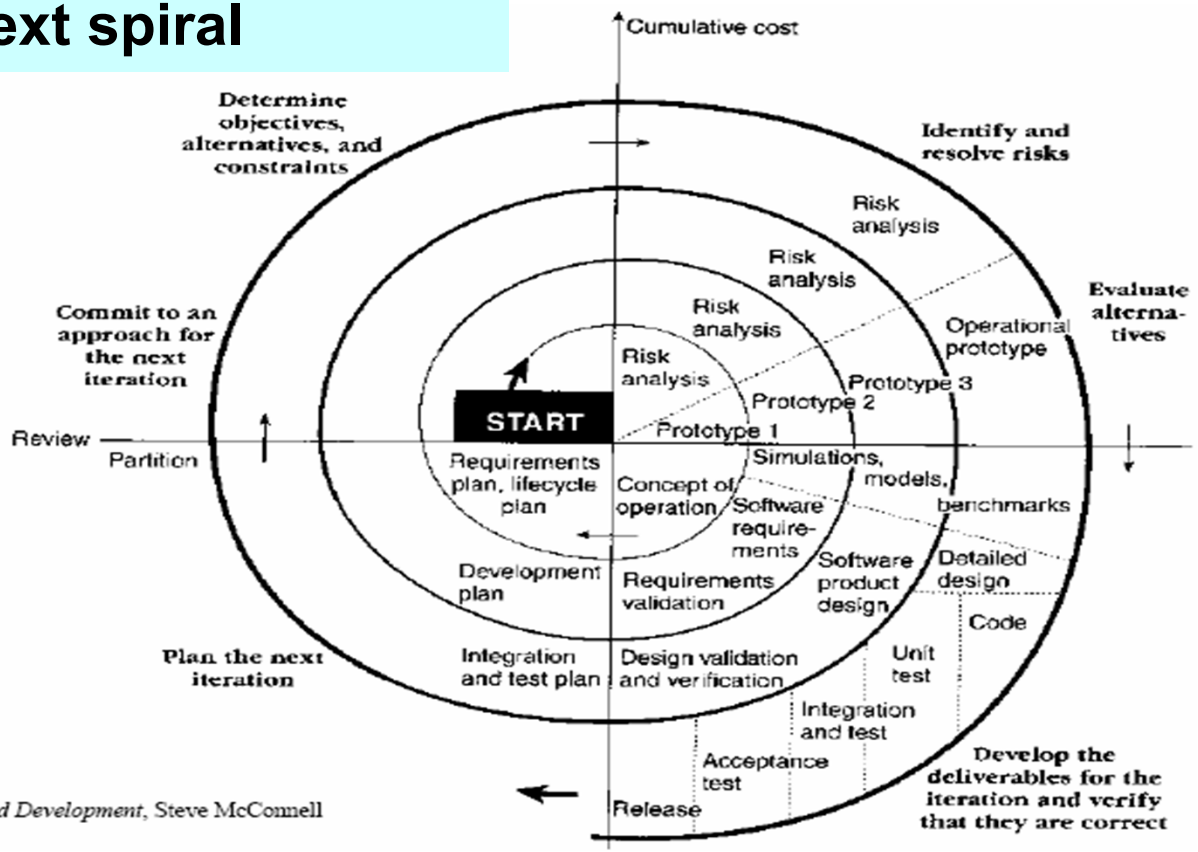
- Can work well for projects that are very well understood but complex
  - Tackles all planning upfront
  - The ideal of no midstream changes equates to an efficient software development process
- Supports inexperienced teams
  - Orderly, easy-to-follow sequential model
  - Reviews at each stage determine if the product is ready to advance

Waterfall model limitations

- Difficult to specify all reqs of a stage completely and correctly upfront
  - requires a lot of planning up front (not always easy)
  - assumes requirements will be clear and well-understood
- Rigid, linear; not adaptable to change in the product
  - costly to "swim upstream" back to a previous phase
- No sense of progress until the very end
  - nothing to show until almost done ("we're 90% done, I swear!")
- Integration occurs at the very end
  - Defies “integrate early and often” rule
  - Solutions are inflexible, no feedback until end
  - Delivered product may not match customer needs
- Phase reviews are massive affairs
  - Inertia means change is costly

Spiral model – risk oriented

- **Determine objectives and constraints**
- **Identify and resolve risks**
- **Evaluate options to resolve risks**
- **Develop and verify deliverables**
- **Plan next spiral**
- **Commit (or not) to next spiral**



Rapid Development, Steve McConnell

Spiral model

- Oriented towards phased reduction of risk
- Take on the big risks early, make decisions
  - are we building the right product?
  - do we have any customers for this product?
  - is it possible to implement the product with the technology that exists today? tomorrow?
- Progresses carefully to a result
  - tasks can be more clear each spiral

Spiral model

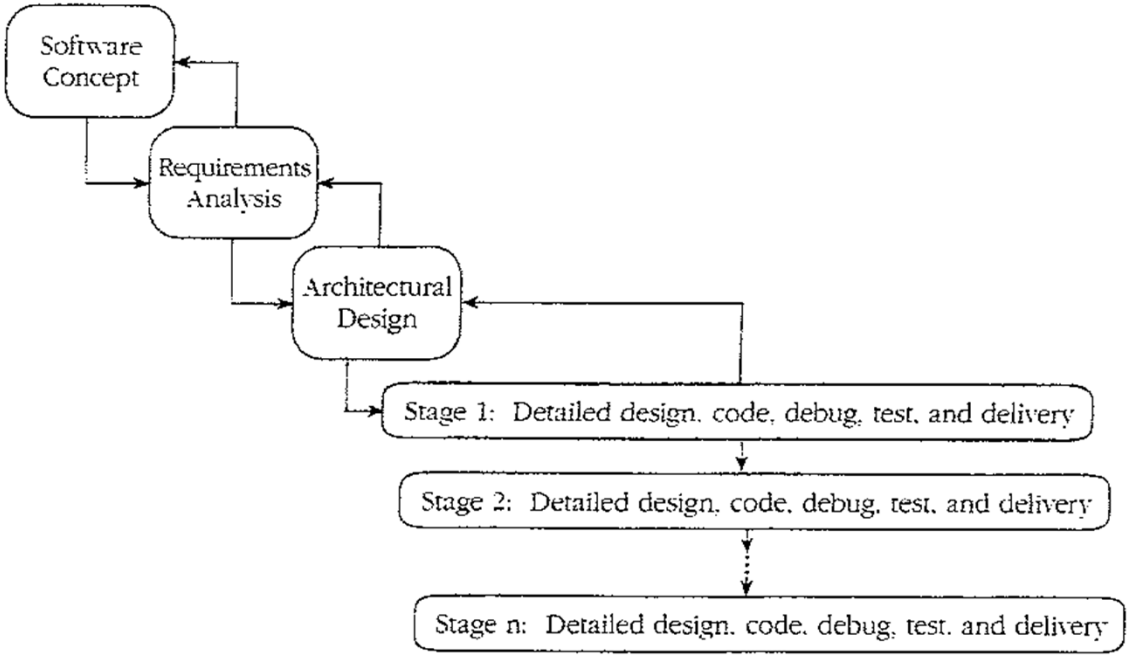
Advantages

- Especially appropriate at the beginning of the project, when the requirements are still fluid
- Provides early indication of unforeseen problems
- Accommodates change
- As costs increase, risks decrease!
  - Always addresses the biggest risk first

Spiral model disadvantages

- A lot of planning and management
- Frequent changes of task
  - But, get to stick with one product feature/goal
- Requires customer and contract flexibility
- Developers must be able to assess risk
  - Must address most important issues

Staged delivery model



Waterfall-like beginnings
Then, short release cycles:
plan, design, execute, test, release
with delivery possible at the end of any cycle

Staged delivery model advantages

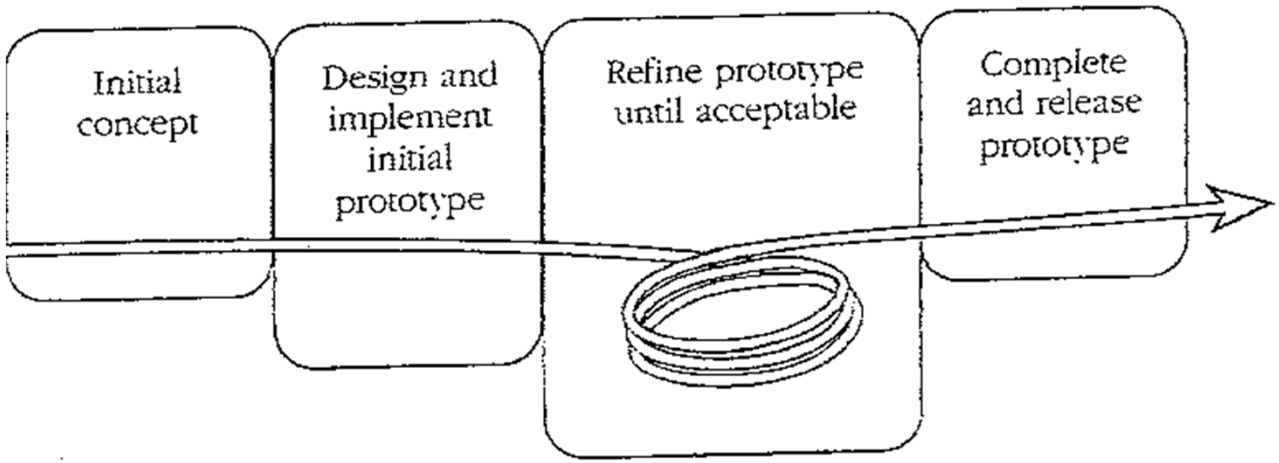
- Can ship at the end of any release cycle
  - Looks like success to customers, even if not original goal
- Intermediate deliveries show progress, satisfy customers, and lead to feedback
- Problems are visible early (e.g., integration)
- Facilitates shorter, more predictable release cycles

Very practical, widely used and successful

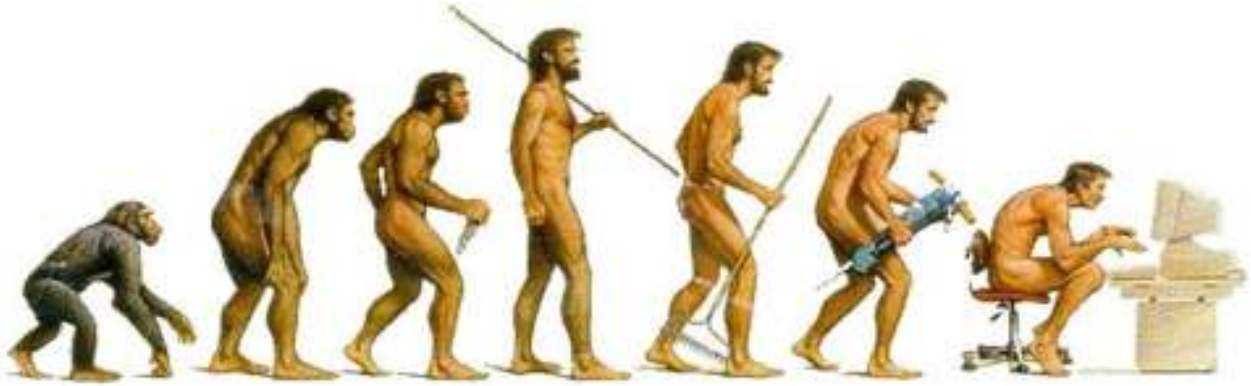
Staged delivery model disadvantages

- Requires tight coordination with documentation, management, marketing
- Product must be decomposable
- Extra releases cause overhead

Evolutionary prototyping model



Develop a skeleton system and evolve it for delivery



Evolutionary prototyping model

Another popular and successful model, especially for custom products

- Staged delivery ≠ evolutionary prototyping
  - Staged delivery: requirements are known ahead of time
  - Evalutionary: discovered by customer feedback on each release

Advantages

- Addresses risks early
- Produces steady signs of progress, builds customer confidence
- Useful when requirements are unknown or changing
- Customer involvement ("What do you think of this version?")

Evolutionary prototyping limitations

- Requires close customer involvement
- Assumes user's initial spec is flexible
- Problems with planning
  - Especially if the developers are inexperienced
  - Feature creep, major design decisions, use of time, etc.
  - Hard to estimate completion schedule or feature set
  - Unclear how many iterations will be needed to finish
- Integration problems
  - fails for separate pieces that must then be integrated
  - bridging; new software trying to gradually replace old
- Temporary fixes become permanent constraints

Design-to-schedule

Design-to-schedule

- useful when you absolutely need to ship by a certain date
- similar to the staged delivery model
  - but less flexible because of the fixed shipping date
- requires careful prioritization of features and risks to address

Design-to-tools

- a model where the project only incorporates features that are easy to implement by using or combining existing components
- reduces development time at cost of losing control of project

Why are there so many models?

Choices are good!

- The choice of a model depends on the project circumstances and requirements
- A good choice of a model can result in a vastly more productive environment than a bad choice
- A cocktail of models is frequently used in practice to get the best of all worlds. Models are often combined or tailored to environment

What's the best model?

Consider

- The task at hand
- Risk management
- Quality / cost control
- Predictability
- Visibility of progress
- Customer involvement and feedback

Aim for good, fast, and cheap.
But you can't have all three at the same time.

Model category matrix

- Rate each model 1-5 in each of the categories shown:

	Risk mgmt.	Quality/ cost ctrl.	Predict-ability	Visibility of progress	Customer involvement
code-and-fix	1	1	1	3	2
waterfall	2	4	3	1	2
spiral	5	5	3	3	3
evolutionary prototyping	3	3	2	5	5
staged delivery	3	5	3	3	4
design-to-schedule	4	3	5	3	2

What's the best SW dev model?

- A system to control anti-lock braking in a car
- A hospital accounting system that replaces an existing system
- An interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports