Semester Wrap-Up

CSCI 5828: Foundations of Software Engineering
Lecture 03 — 05/03/2012

Goals

- Present a review of the topics covered in class this semester

Four Main Topics

- Software Engineering Fundamentals
- Software Life Cycles
  - Agile philosophy and techniques
- Software Testing
  - Behavior and Test Driven Development
- Software Concurrency
  - Design and Implementation of Concurrent Software Systems

SE Fundamentals (I)

- What is Software Engineering
  - Software engineering is that form of engineering that applies…
    - a systematic, disciplined, quantifiable approach,
    - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines…
  - to creating, developing, operating, and maintaining cost-effective, reliably correct, high-quality solutions to software problems. (Daniel M. Berry)
- Fred Brooks's No Silver Bullet
  - Progress will be made on software engineering in terms of our ability to be produce high quality software on time and under budget
    - But it will be hard work! No one technique is going to "save us"

SE Fundamentals (II)

- Overview of Software Life Cycles
  - In SE, Process is King!
    - Examined transition and differences between traditional waterfall methods and more recent agile approaches
- Overview of Software Testing
  - Errors, Faults, and Failures
  - Black box, gray box, and white box
  - Folding and Sampling
  - Test Driven Development

SE Fundamentals (III)

- Overview of Concurrency
  - Why the design of concurrent software systems is important
    - Chips are getting “wider” not faster
  - But concurrency is hard
    - race conditions, deadlock, etc.
  - It’s especially hard if you continue to do concurrency the way we’ve always done it
    - Shared mutability, low-level threading primitives, locks
  - We then examined higher-level abstractions that avoid these problems

Agile (I)

- A software development philosophy and a set of practices
  - that values
    - communication over process
    - communication over documents
    - communication over tools
    - <do you notice a pattern?>
    - and advocates a set of practices that help developers embrace the fact that change in software development is inevitable
      - Don’t hide from it!

Agile (II)

- Specific Techniques
  - Agile Inception Deck: Make sure the team and the customer are aligned
  - User Stories
  - Iteration Plan
  - Burn-Down Charts
  - Test Driven Development
  - Continuous Integration
  - Configuration Management

Software Testing

- Test Automation Frameworks
  - Cucumber as the example
    - How to get your customer to write tests
    - How to maintain separation between
      - test code and the system under test
  - Examined strategies for keeping test code abstract
    - while underneath via glue code the system under test could grow
      - from simple model code to full fledged system with UIs, web services, etc.

Concurrency

- Fairly broad coverage of concurrency techniques at the individual system level
  - Typical threading primitives and the problems associated with them
  - java.util.concurrent
    - notion of separation of thread and task
    - and the need for a thread allocation strategy
  - Styles of Concurrency: shared mutability, isolated mutability, ...
  - “New” concurrency models
    - Software Transactional Memory, Agent Model, Grand Central Dispatch

Two Aspects of Concurrency Not Explored

- A Multi-Process Approach to Concurrency
  - With this approach your “system” is a bunch of individual programs where
    - each individual program is single threaded and thus easier for developers to understand and maintain
    - concurrent operation comes from the fact that the operating system will run these processes at the same time on different cores
    - coordination occurs via inter-process communication or via the file system
- MapReduce
  - can be done in individual programs but also enables large scale distribution of computation across clusters of machines

MapReduce

MapReduce

- To understand MapReduce, we must first talk about functional programming
  - We encountered functional programming when we looked at Clojure in the context of STM and in our discussions of pure immutability
- Functional programming is an approach to programming language design in which functions are
  - first class values (with the same status as int or string)
    - you can pass functions as arguments, return them from functions and store them in variables (as we saw with blocks in GCD)
  - and have no side effects
    - they take input and produce output
    - this typically means that they operate on immutable values

Example (I)

- In python, strings are immutable
  - a = “Ken @@@”
  - b = a.replace(“@”, “!”)
  - b
    - 'Ken !!!'
  - a
    - 'Ken @@@'
- Replace is a function that takes an immutable value and produces a new immutable value with the desired transformation; it has no side effects

Example (II)

- Functions as values (in python)
  - def Foo(x, y):
    - return x + y
  - add = Foo
  - add(2, 2)
    - 4
- Here, we defined a function, stored it in a variable, and then used the “call syntax” with that variable to invoke the function that it pointed at

Example (III)

- continuing from previous example
  - def DoIt(fun, x, y): return fun(x,y)
  - DoIt(add, 2, 2)
    - 4
- Here, we defined a function that accepts three values, some other function and two arguments
  - We then invoked that function by passing our add function along with two arguments ;
  - DoIt() is an example of a **higher-order function**: functions that take functions as parameters
  - Higher-order functions are a common idiom in functional programming

Relationship to Concurrency?

- How does this relate to concurrency?
  - It leads naturally to the pure immutability style of concurrent design
    - Each thread operates on immutable data structures using functions with no side effects
    - A thread’s data structures are not shared with other threads
    - Work is performed by passing messages between threads
      - If one thread requires data from another, that data is copied and then sent
- As we’ve seen, such an approach allows each thread to act like a single-threaded program; no danger of interference

Map, Filter, Reduce

- Three common higher order functions are **map**, **filter**, **reduce**
- map(fun, list) -> list
  - Applies fun() to each element of list; returns results in new list
- filter(fun, list) -> list
  - Applies boolean fun() to each element of list; returns new list containing those members of list for which fun() returns True
- reduce(fun, list) -> value
  - Returns a value by applying fun() to successive members of list (total = fun(list[0], list[1]); total = fun(total, list[2]); …)

Examples

- list = [10, 20, 30, 40, 50]

- def double(x): return 2 * x
- def limit(x): return x > 30
- def add(x,y): return x + y

- map(double, list) returns [20, 40, 60, 80, 100]
- filter(limit, list) returns [40, 50]
- reduce(add, list) returns 150

Implications

- map is very powerful
  - especially when you consider that you can pass a list of functions to it and then pass a higher-order function as the function to be applied
    - for example
      - def DoIt(x): return x()
      - map(DoIt, [f(), g(), h(), i(), j(), k()])
- But the real power, with respect to concurrency is that map is simply an abstraction that can, in turn, be implemented in a number of ways

Single Threaded Map

- We could for instance implement map() like this:
    - def map(fun, list):
        - results = []
        - for item in list:
            - results.append(fun(item))
- This would implement map in a single threaded fashion

Multi-threaded Map

- We could also implement map like this (pseudocode):
  - def Mapper(Thread):
    - def __init__(… fun, list): …
    - def run():
      - self.results = map(fun, list)
  - def xmap(fun, list):
    - split list into N parts where N = number of cores
    - create N instances of Mapper(fn, list_i)
    - wait for each thread to end (in order) and grab results
    - append thread results to xmap results
    - return xmap results

Note: threads can complete in any order since each computation is independent

Super Powerful Map

- We could also implement map like this:
  - def supermap(fun, list):
    - divide list into N parts where N equals # of machines
    - send list_i to machine i which then invokes xmap
    - wait for results from each machine
    - combine into single list and return
- Given this implementation, you can apply a very complicated function to a very large list and have (potentially) thousands of machines leap into action to compute the answer

Google

- Indeed, this is what Google does when you submit a search query:
  - def aboveThreshold(x): return x > 0.5 <-- just making this up
  - def probabilityDocumentRelatedToSearchTerm(doc): …
  - searchResults =
    - filter(aboveThreshold,
      - map(probabilityDocumentRelatedToSearchTerm,
        - [<entire contents of the Internet]))

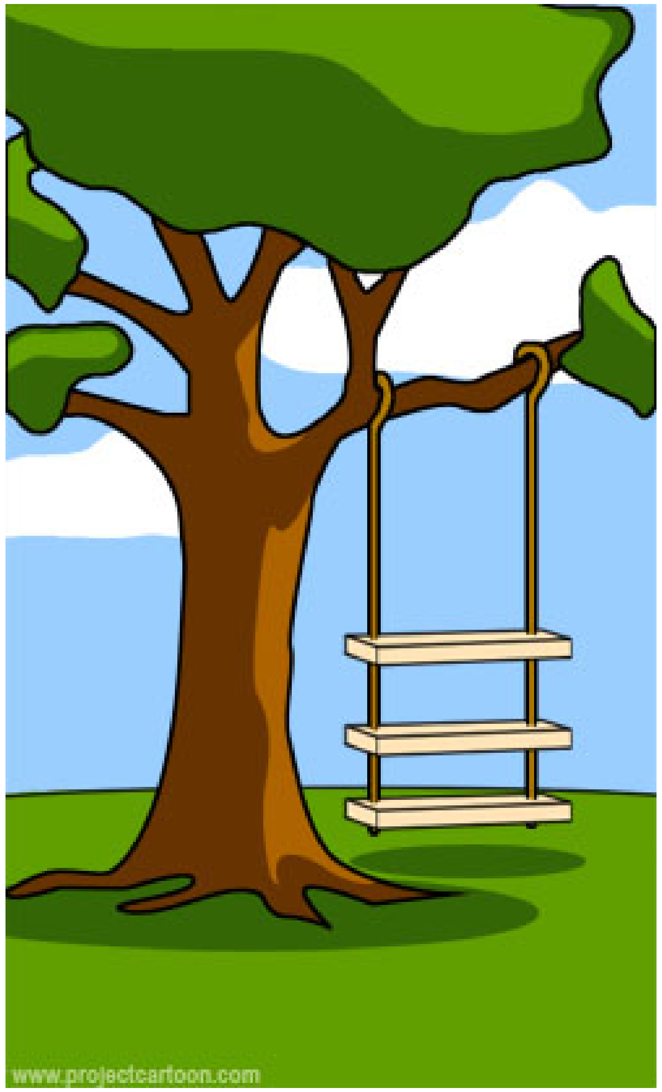
Difference between map and xmap?

- The team behind Erlang published results concerning the difference between map and xmap
  - They make a distinction between
    - CPU-bound computations with little message passing vs.
    - lightweight computations with lots of message passing
- With the former, xmap provides linear speed-up (10 CPUs provides a 10x speed-up, then declining) over map
  - the latter less so (10 CPUs provided 4x speed-up)
  - Indeed, xmap’s performance in the latter case tends to max out at 4x no matter how many CPUs were added

How Projects Really Work

- Our last lesson for the semester involves insight into how software development projects **REALLY** work
  - Taken from www.projectcartoon.com

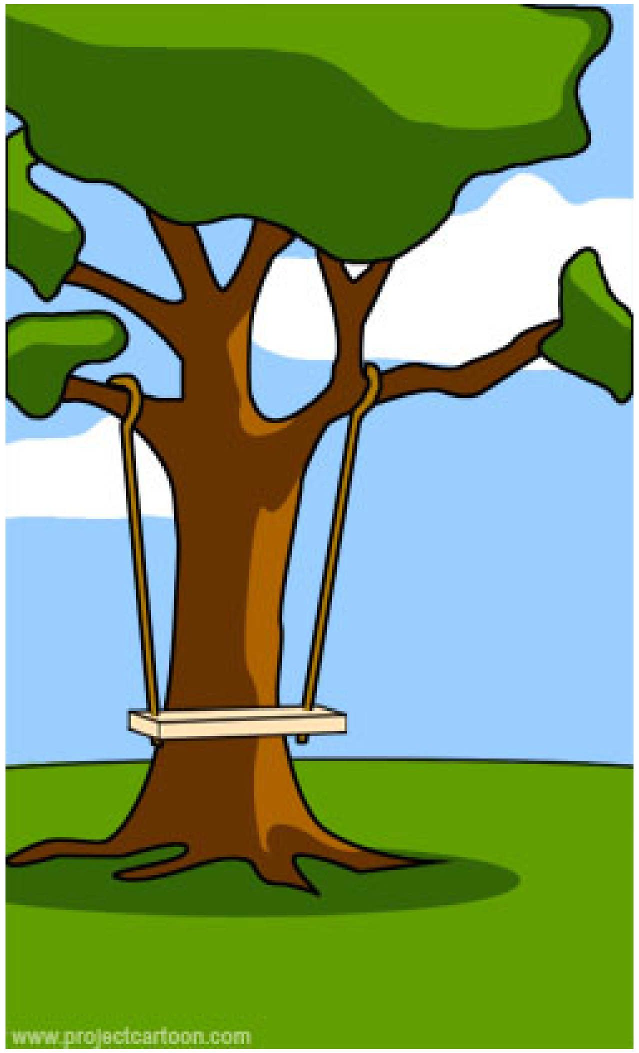
How Projects Really Work



How the customer explained it

www.project
cartoon.com

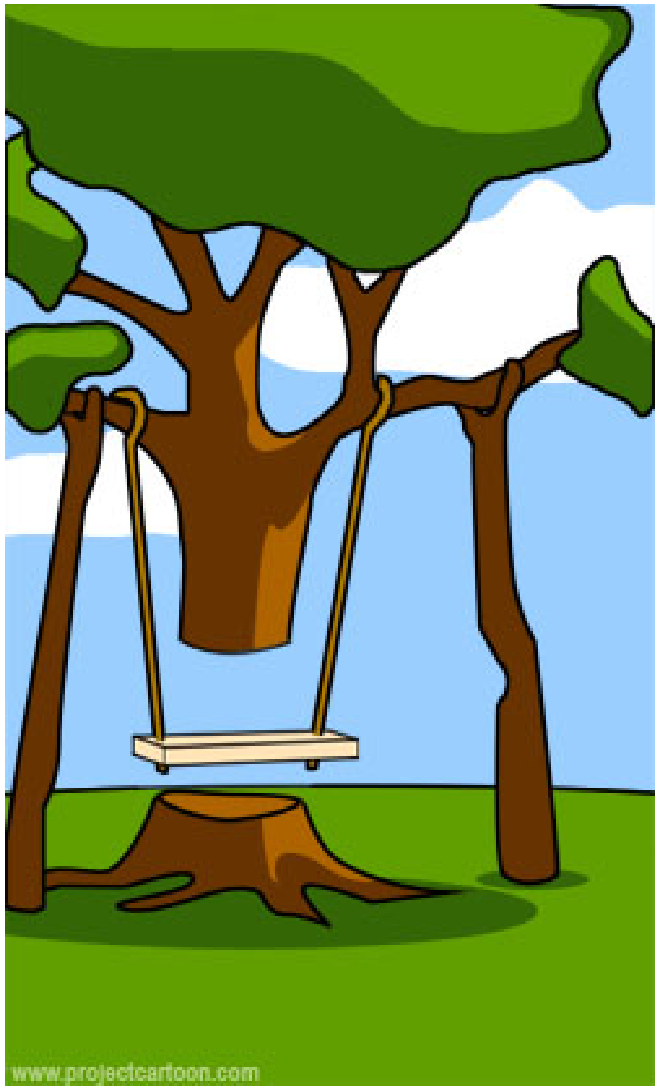
How Projects Really Work



How the project leader understood it

www.project
cartoon.com

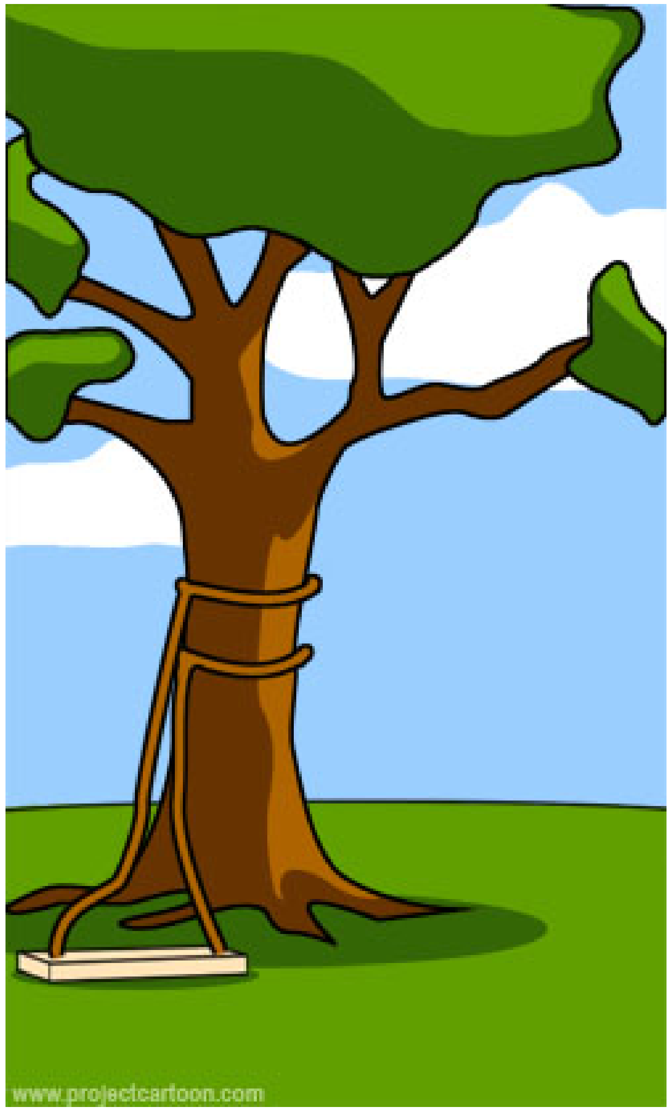
How Projects Really Work



How the analyst designed it

www.project cartoon.com

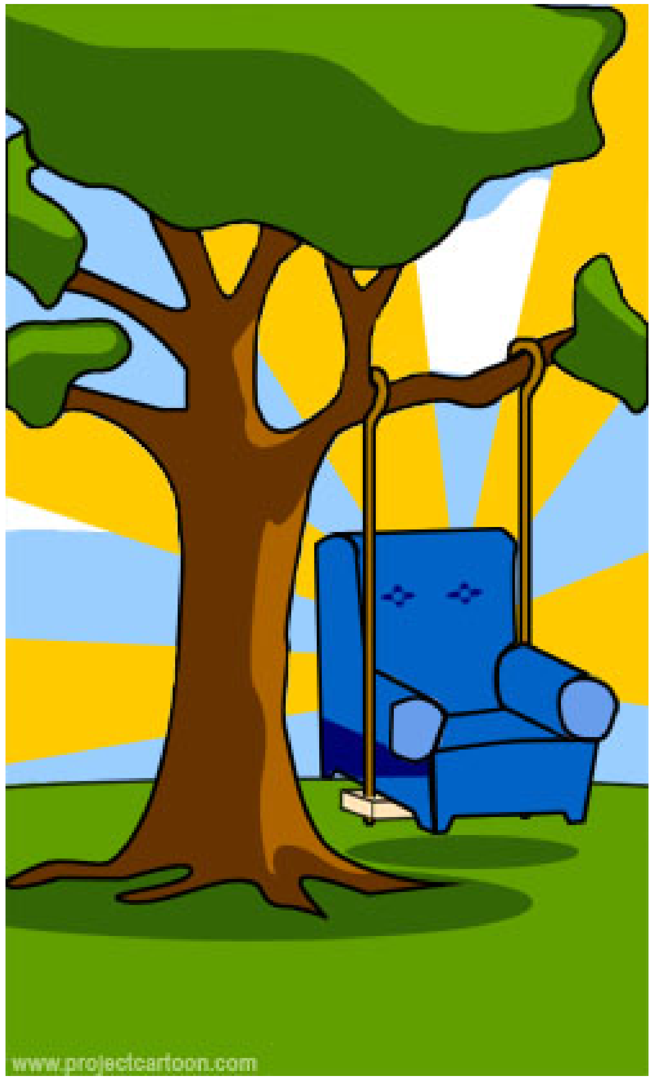
How Projects Really Work



How the programmer wrote it

www.project cartoon.com

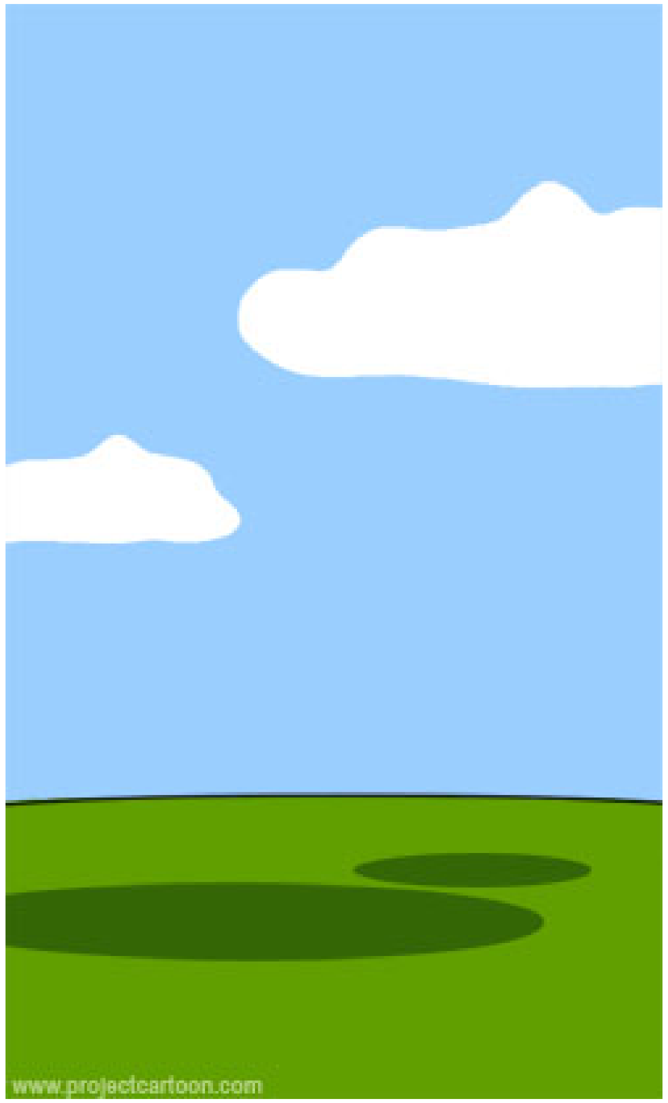
How Projects Really Work



How the business consultant described it

www.project
cartoon.com

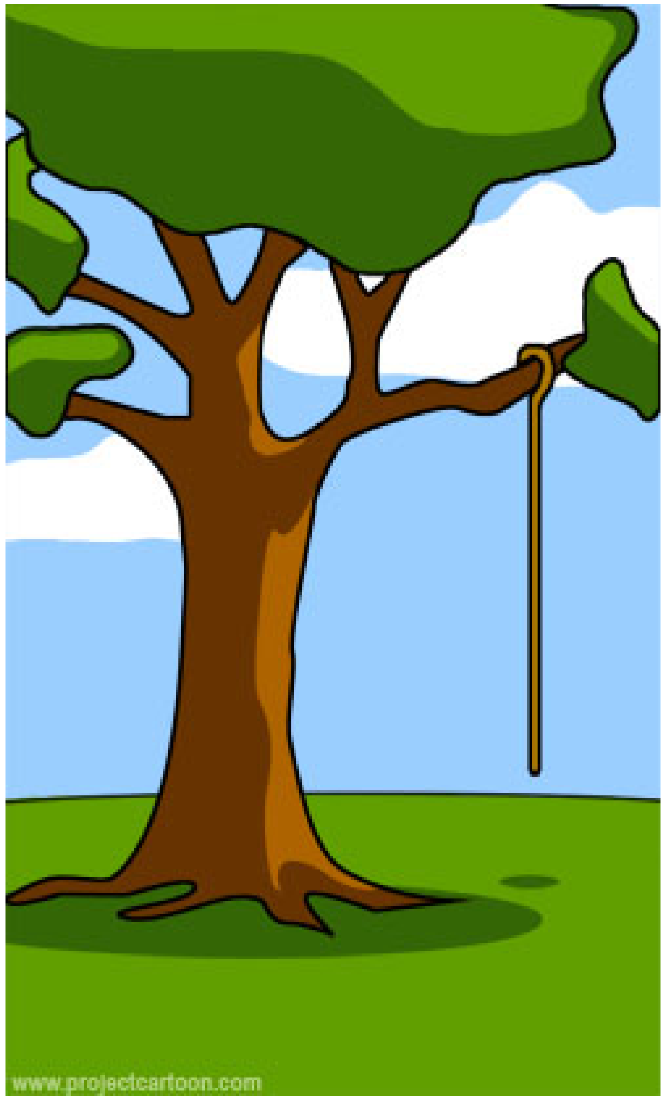
How Projects Really Work



How the project was documented

www.project
cartoon.com

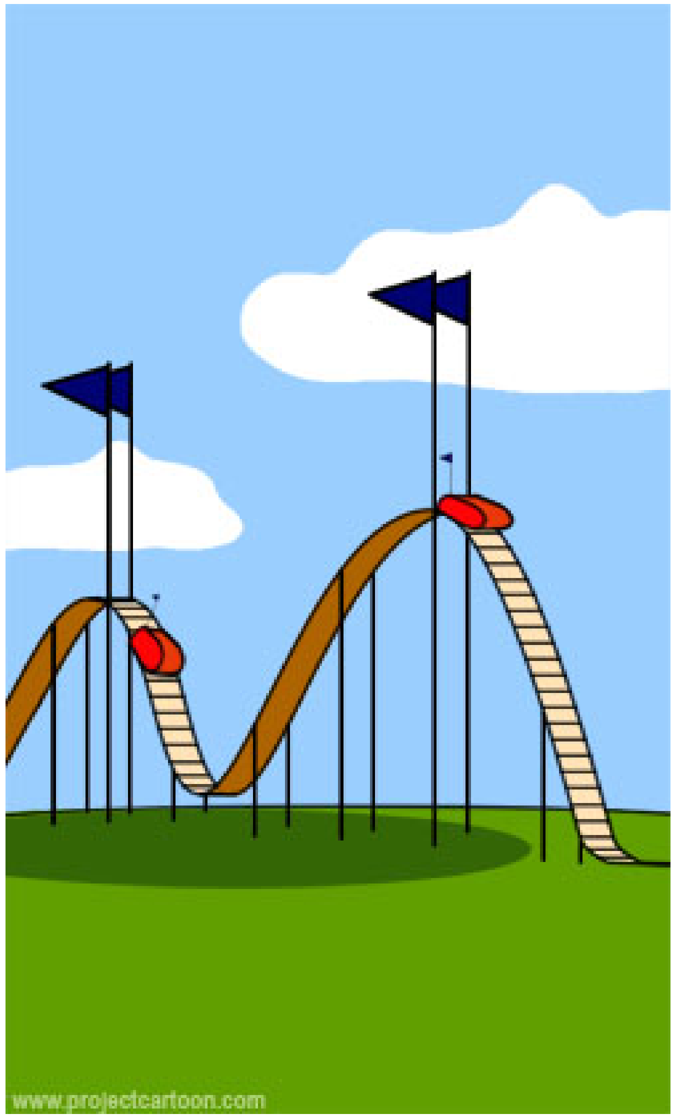
How Projects Really Work



What operations installed

www.project cartoon.com

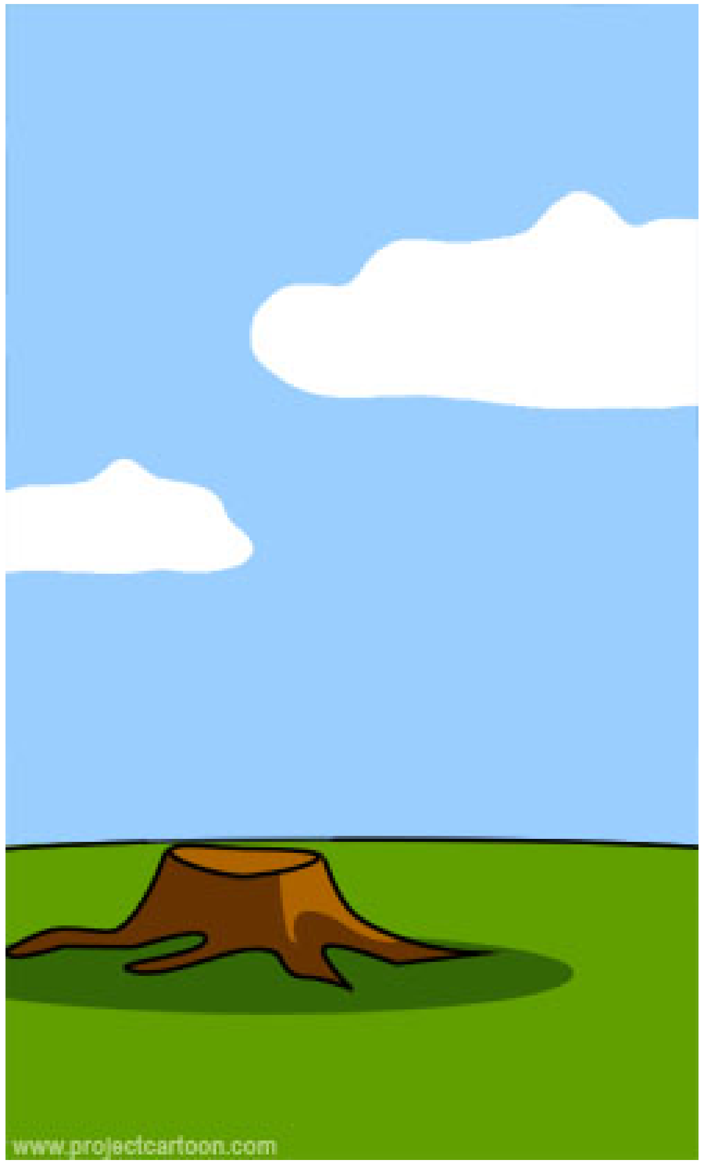
How Projects Really Work



How the customer was billed

www.project
cartoon.com

How Projects Really Work



How it was supported

www.project
cartoon.com

How Projects Really Work

What the customer really needed

www.project cartoon.com

Coming Up Next

- Summer! Have a good one!