

# Software on Internet Time ...and Time Again

## *Part II: Agile Model-Based Development in Practice*

*Marc J. Balcer*



ModelCompilers.com  
*code at a higher level*

# Project Characteristics

- Information exchange for intermodal (container) shipping industry
  - Potentially enormous problem space (3' x 10' Visio<sup>®</sup> diagram)
- Customer has a long-term vision and short-term objectives
  - Long-term vision does not mean long time to get something
  - Can't make money until something is deployed
  - Objectives refined as system deployed



# Customer milestones

- September 1 – start
- November 30 – pilot application
  - major industry trade show
  - six pilot customers
- April 1 – production rollout
  - general availability of pilot functionality++



# Build on a solid foundation

- We can achieve the objective without the long time and uncertainty of nothing delivered (the “internet time”)
- We can sustain this (the “time again”)



# Principles

- Go Incremental
- Start Developing Immediately
- Build for Testing
- Code at a Higher Level
- Automate Aggressively
- Release Often
- Collaborate and Communicate



# Go Incremental

- Develop the application as a series of incremental releases
- Each increment is short (time-boxed)
- Each increment delivers working software
- Each increment delivers real business value



# Start Simple, Increase Complexity

Increment 1	Increment 2	Increment 3	Increment 4
Single-item order	Multi-item order		
Each order is made up individually		Allow reordering from a "shopping cart"	Allow sharing of shopping carts (e.g. a book list for a course)
Books only		Movies, videos, and software	
All items assumed to be in stock		Items may not be in stock	
All items assumed to be stocked by and shipped from bookstore			Some items may be special-order from the publisher: we submit order to publisher and publisher ships directly
Ship when payment (credit card charge) approved		Ship only when all product in stock	Partial shipments as product is restocked
Order closed when shipped	Order closed when shipment received by customer		
Do not handle returns		Customer can produce a "return authorization" from a shipment	No return authorizations for items not yet shipped
Credit card payments only			House accounts for pre-approved customers; billing done by separate system
Credit card payment must be approved or order canceled	Allow charge resubmitting against declined charges		
Full payment when order checked out			
Can cancel only orders not yet checked out		Can cancel orders not yet shipped	Can cancel parts of orders not yet shipped
Cannot cancel paid orders		Canceling paid orders requires issuing a credit only when returned items received	Canceling parts requires issuing partial credit

Baseline versions of features

Cell = "user story"



A complete trace through the whole system

# Increment Types

- Production increments
  - longer-term (e.g. 2-3 months)
  - aligned to business objectives
  - business can't always absorb weekly changes
- Development increments
  - shorter term (e.g. 1-2 weeks)
  - provide visibility & allow customer to “drive”
  - how development measures its own progress



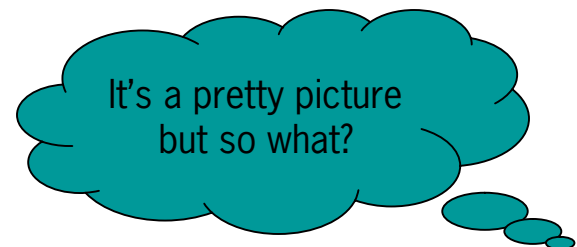
# Start Developing Immediately

- Traditional “overweight” approaches would have
  - done a big project charter document
  - spent months writing use cases, specification documents, and UML diagrams to define the app content
  - done a big technical spec document to design the architecture



# Start Developing Immediately

- But none of those things are verifiable
  - they cannot be checked for consistency
  - they cannot be executed
  - they cannot be tested



# Two weeks, two tracks

## ■ Prototype Architecture

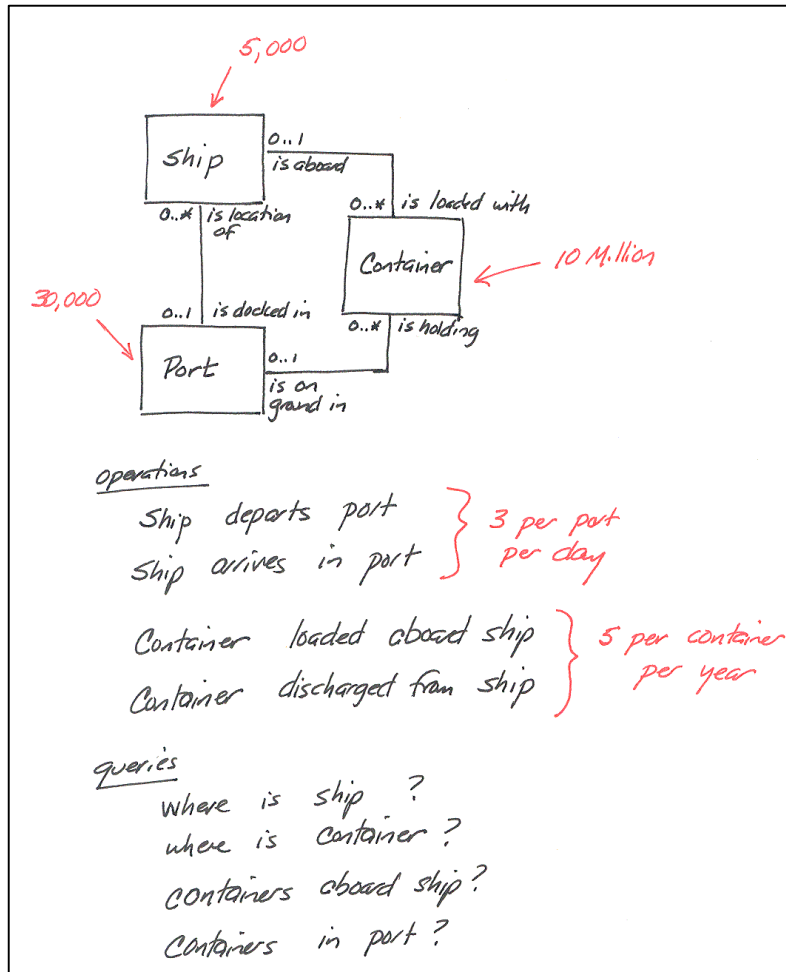
- establish development & deployment environments
- establish automated build, test, and deployment processes
- test key performance objectives
- code, test, deploy super-simple application

## ■ Application Modeling

- identify priorities
- set scope
- sketch solutions
- formalize knowledge
- define the vocabulary
- build models



# Prototype Architecture Immediately



- A super-simple guess—  
real app still horribly  
undefined
- Intended to get the  
architecture & practices  
in place
  - Test “-abilities”
  - Development foundation  
required for incremental  
development
  - “Working Software  
over ... Documents”



# Prototype Architecture Immediately

- Development infrastructure must be in place
- Team members must be trained in technologies
- Key performance issues need to be demonstrated
- Team needs to learn to work together
- Reinforce notion of “deliver value quickly”



# Application Models

- Develop collaboratively with the customer
  - Ensure understanding of the “big picture”
  - Time-box the process
  - Make simplifying assumptions
- Deliver understanding, not documents
  - Little thinking tools to get the big picture
  - Build what’s testable
  - Expose detail – formalize knowledge



# Build for Testing

- Tests = behavioral objectives
- Test-first design  
("if you dream it, you'd better know how to test it")
- Every build runs all tests
  - Several builds per day—Daily builds are for wimps!



# Build for Testing

- Unit tests
  - Each Java class has a Junit script
- Application tests
  - Application built with test hooks—automated XML tests
- UI tests
  - Design UI structure so tests can be automated—  
—not just capture-playback
- Model tests
  - Test the business rules



# Build for Testing

- “Close the loop”—  
modeled functionality is covered by the tests.
- Plan to audit the test coverage—  
“100% tests pass”—yes, but what tests?
- Do not rely completely upon UI tests—  
build a layer for scripted automated tests
- Related to “Automate Aggressively”



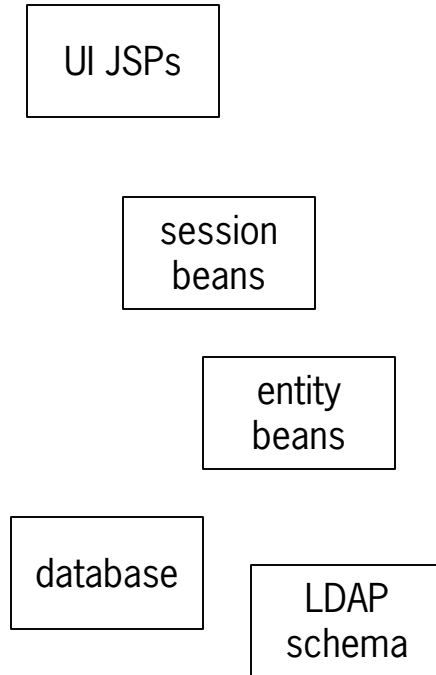
# Code at a higher level

- System code as the record of authority—  
yes, but what code?
  - “Code” is more than Java: it can include
    - configuration files
    - screen designs
    - pre-existing instances
- Factor out common services
  - Not merely code rearrangement refactoring
  - Discover and use new intermediate abstractions (domains)
  - More simpler, fewer complex things

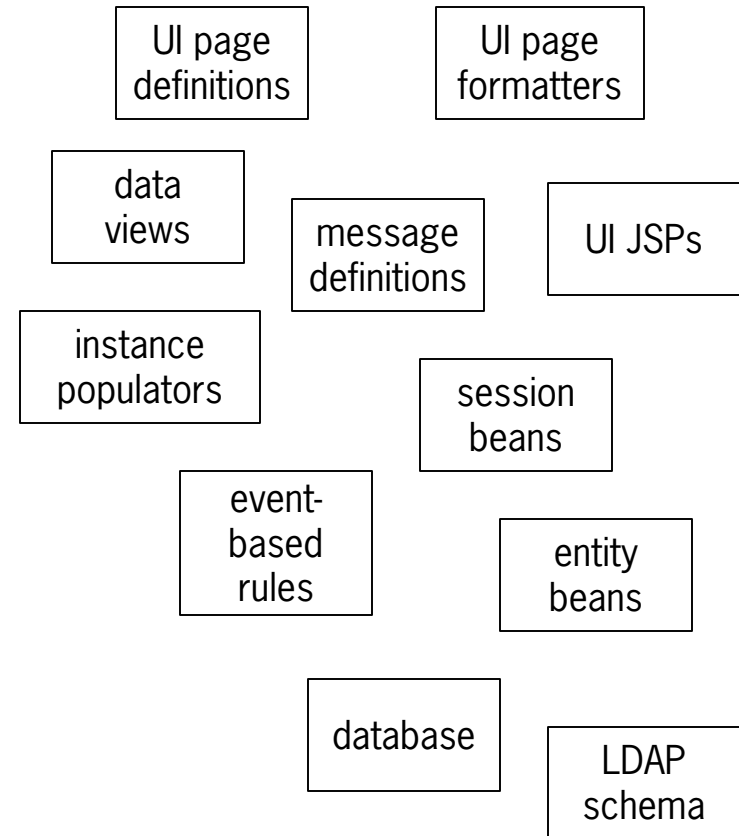


# New domains = New kinds of

*for the pilot,  
we created...*



*by increment 3  
we were creating...*



# Code at a Higher Level

- Everyone is a developer  
("How is this person contributing to delivering working software?")
- Replace documents and demoware with real working software
- Related to "Automate Aggressively"



# Automate Aggressively

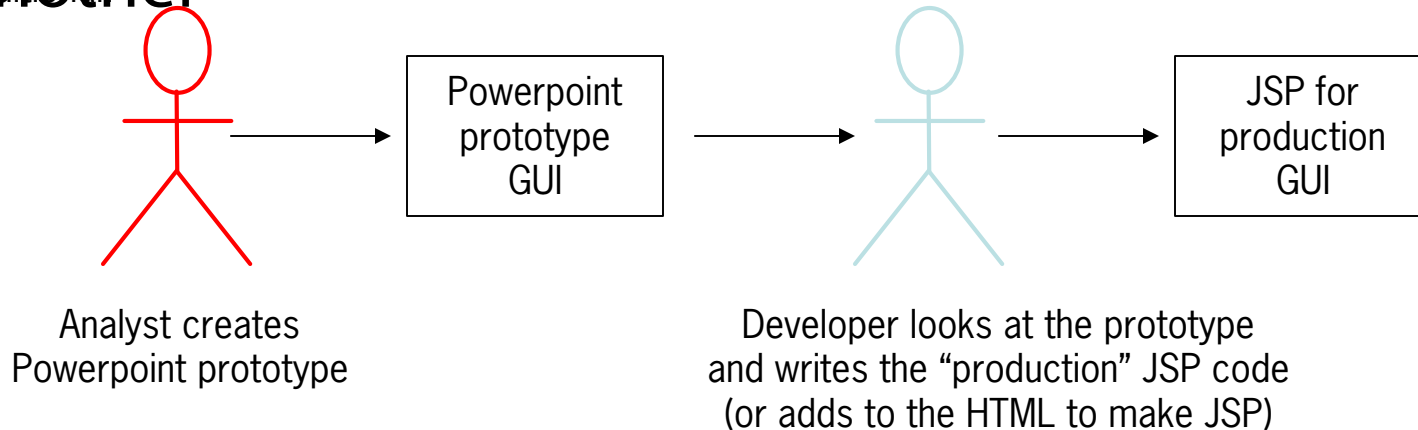
- Builds
  - obviously—everything built from source
- Tests
  - build the application to be automated-testable
- Translation & Generation
  - use the application and service models
  - build the engines to make the little languages executable
  - generate redundant code from the little languages



# Translation & Generation

- Once something is in a machine representation, we shouldn't need to manually re-interpret it ("elaborate it") into

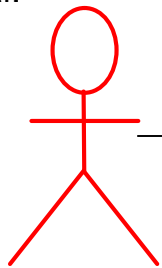
another



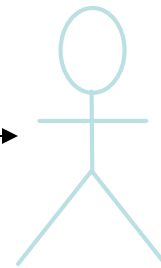
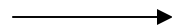
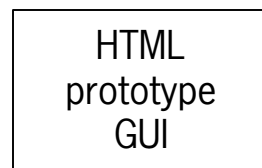
# Translation & Generation

- The rework introduces development errors and obscures analyst errors

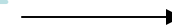
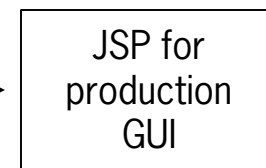
Traditional:



Analyst creates HTML prototype



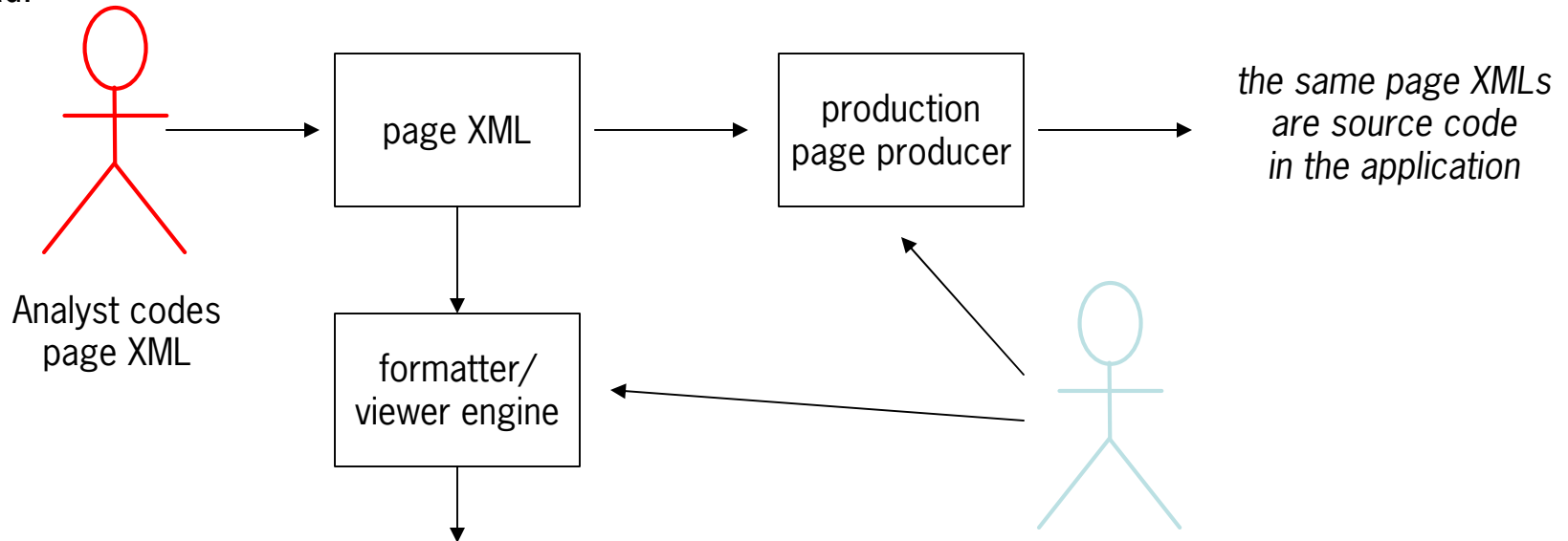
Developer looks at the prototype and writes the "production" JSP code (or adds to the HTML to make JSP)



# Translation & Generation

- The design is the code

Instead:



allows analyst to

- design pages with customer
- design (global) formatting with developers

Developer writes two engines:

- a quick formatter / viewer
- the actual production page producer



# Automate Aggressively

- Audit the build process—  
every released version + current one  
can be built from the source library alone
- Look out for elaborative processes
- Replace manual repetitive processes with  
automated testing

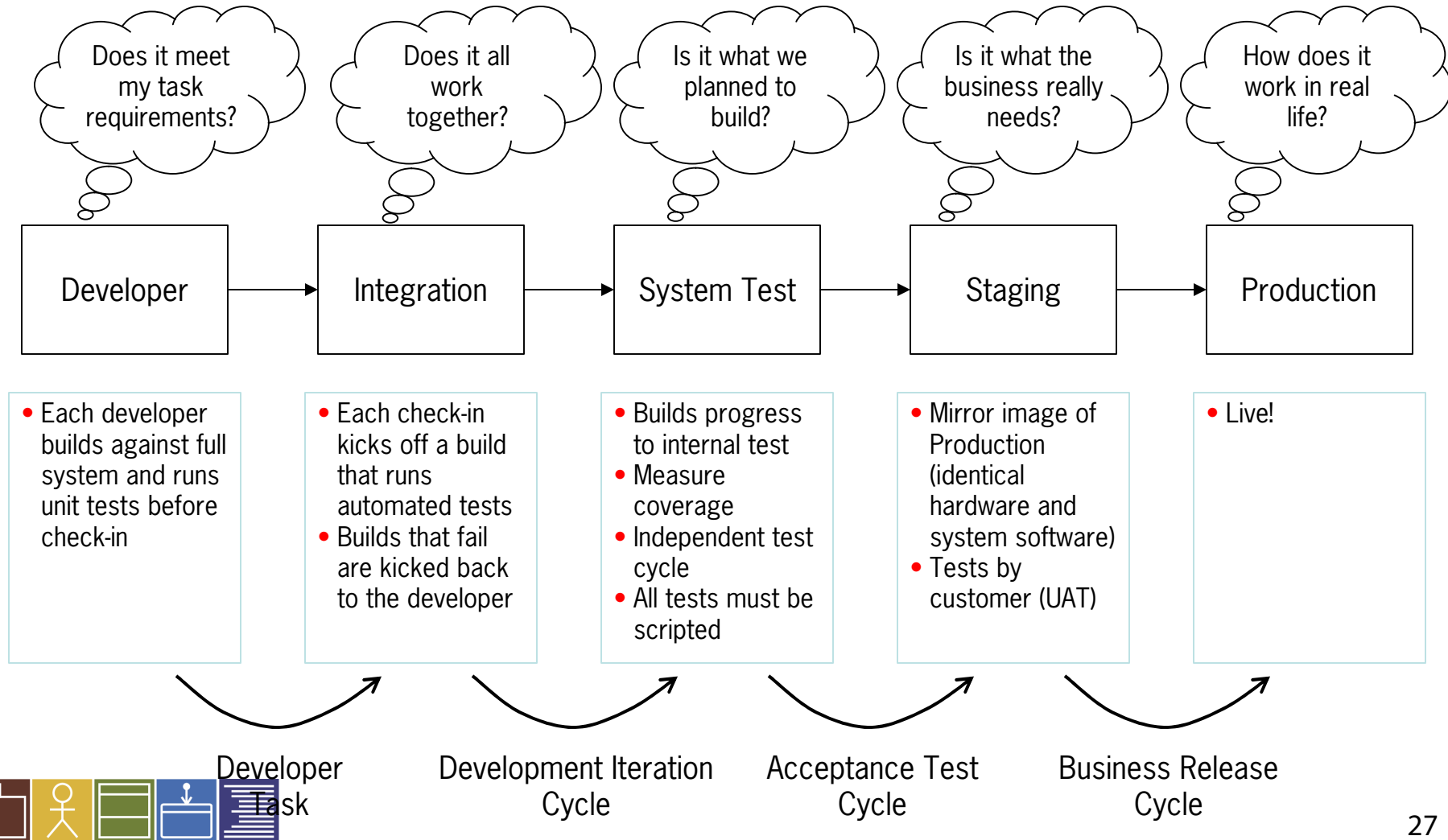


# Release Often

- Demonstrate real progress
- Real usage gives best feedback
- System in use changes business objectives
- Changed objectives change project direction



# Release Chain



# Release Often

- Customer must be tolerant of incompleteness (that's true for the whole team!)
- Automated test scripts are code
  - version-controlled
  - part of a build
- Can measure features/stories by where each is complete-to (where is it in the chain)



# Collaborate and Communicate

- “Customer Collaboration over contract negotiation”  
—Agile Manifesto
- Open Office Partnership
- Climate of trust and support
- Shared objective to deliver working software
- Change is expected



# Summary

- Go Incremental
- Start Developing Immediately
- Build for Testing
- Code at a Higher Level
- Automate Aggressively
- Release Often
- Collaborate and Communicate

