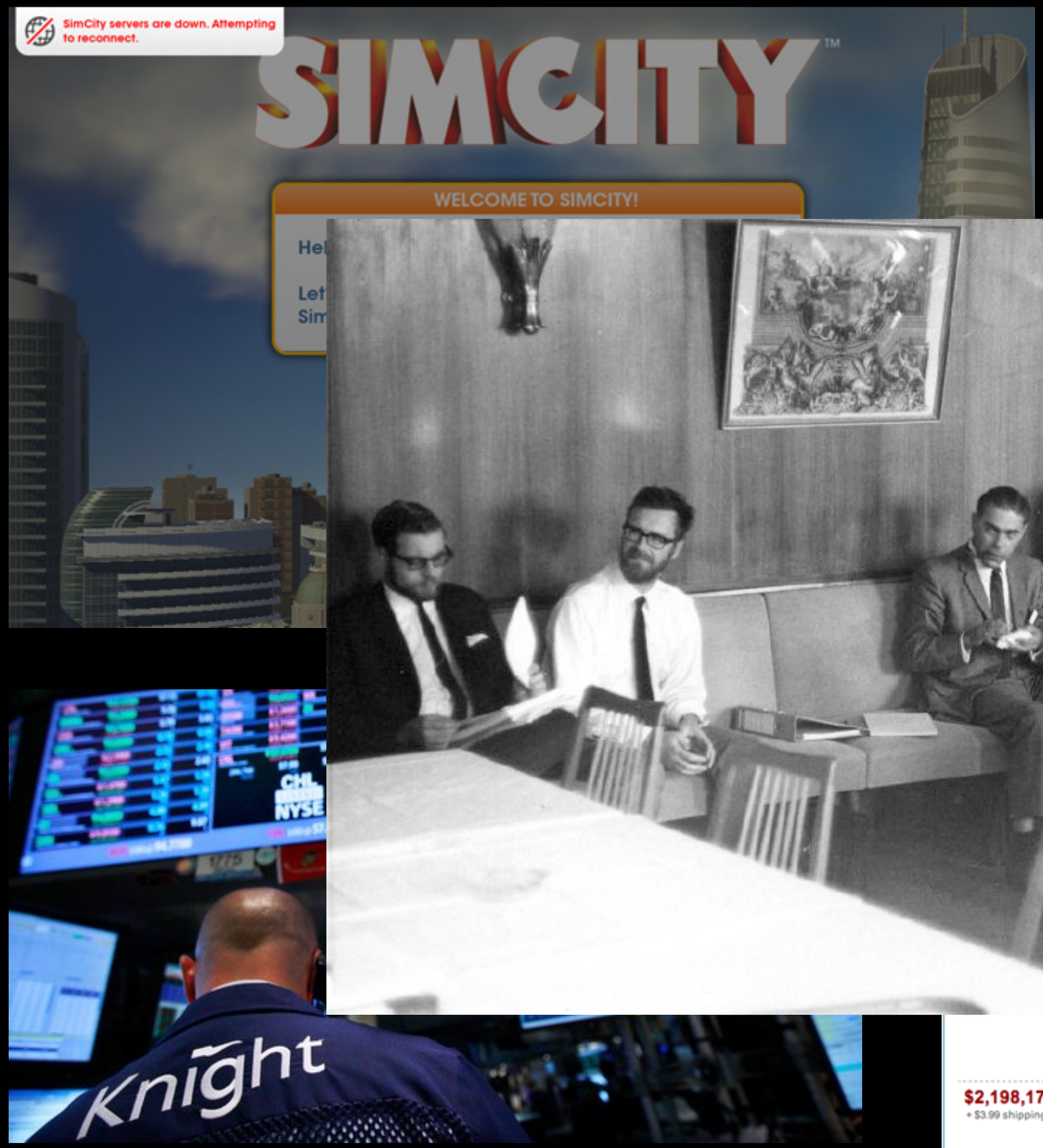
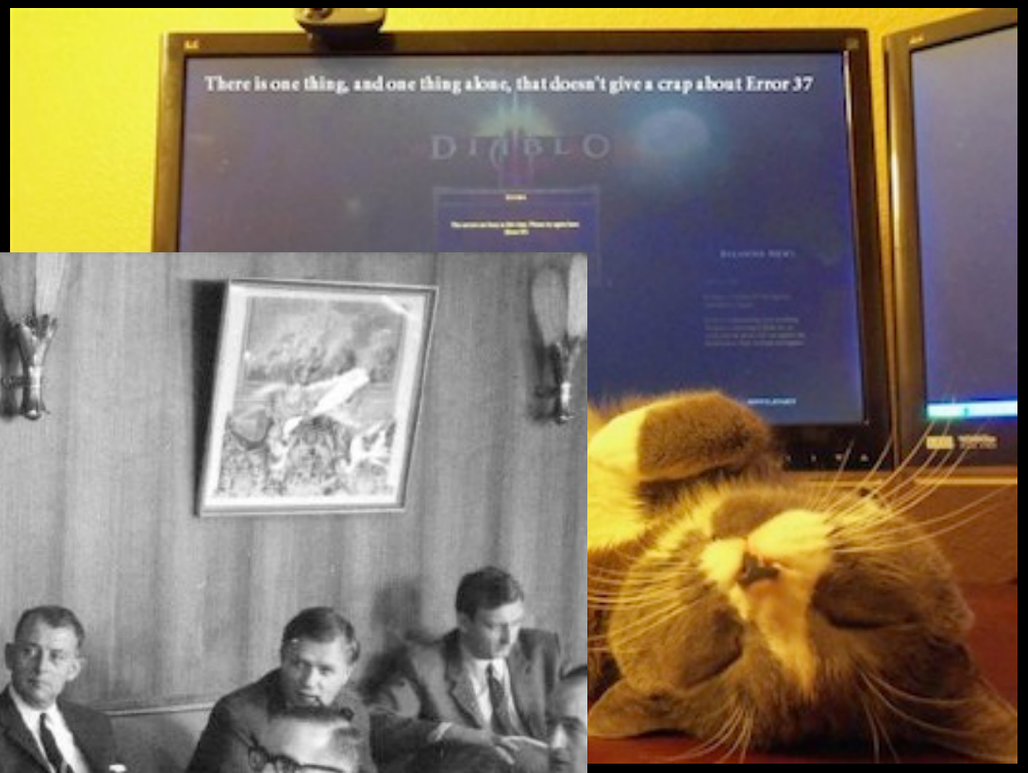
Software Engineering Project

Morgan Ericsson

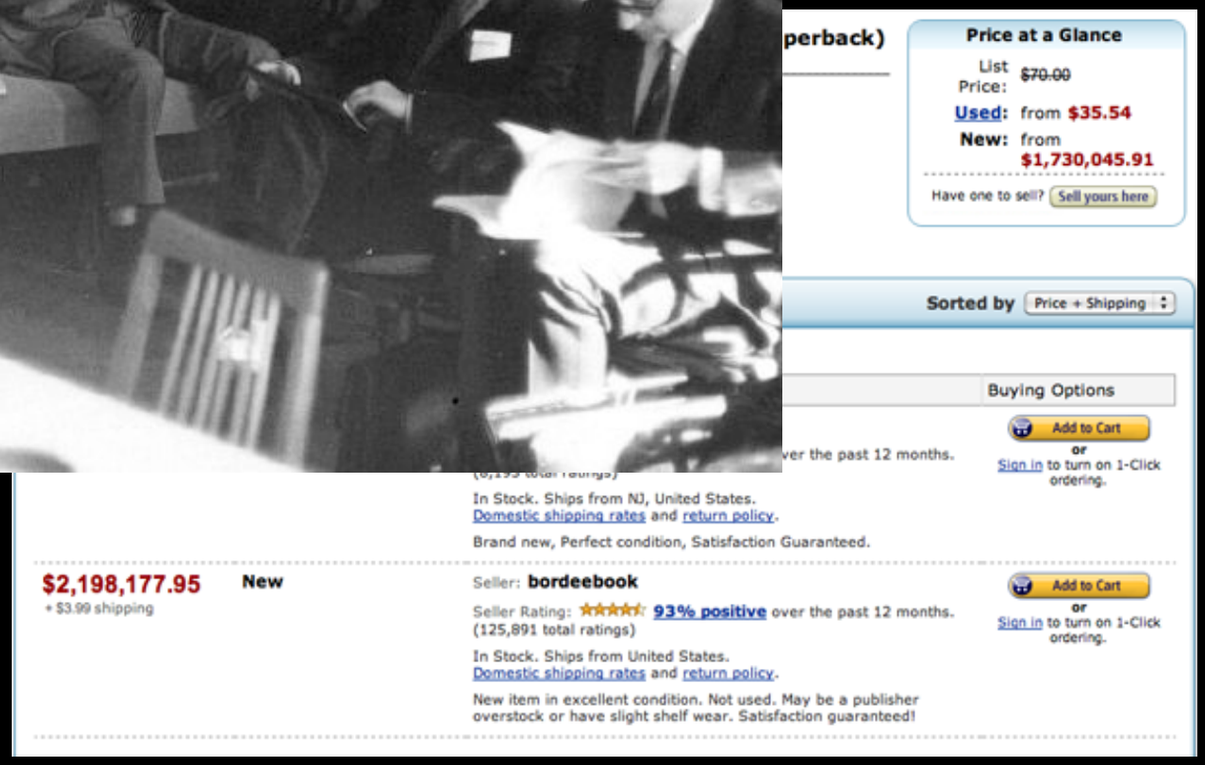
morgan@cse.gu.se
@morganericsson
morganericsson
morganericsson

UNIVERSITY OF GOTHENBURG
CHALMERS


SimCity servers are down. Attempting to reconnect.
SIMCITY™
WELCOME TO SIMCITY!


There is one thing, and one thing alone, that doesn't give a crap about Error 37
DIABLO

NYSE
Knight


perback)
Price at a Glance
List Price: $70.00
Used: from $35.54
New: from $1,730,045.91
Have one to sell? Sell yours here
Sorted by Price + Shipping
Buying Options
Add to Cart
or
Sign in to turn on 1-Click ordering.
In Stock. Ships from NJ, United States.
Domestic shipping rates and return policy.
Brand new, Perfect condition, Satisfaction Guaranteed.
$2,198,177.95
New
+ $3.99 shipping
Seller: bordeebook
Seller Rating: 93% positive over the past 12 months. (125,891 total ratings)
In Stock. Ships from United States.
Domestic shipping rates and return policy.
New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!
Add to Cart
or
Sign in to turn on 1-Click ordering.

Software Crisis and Engineering

- Established as a reaction to the "Software Crisis"
  - software was inefficient
  - software did not meet requirements
  - projects ran over time/budget
  - projects were unmanageable and software unmaintainable

"The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

Software Engineering

- The branch of computer science that creates practical, cost-effective solutions to computing and information processing problems
  - *"Application of engineering to software"*
    - *systematic*, *disciplined*, *quantifiable* approach to the *development*, *operation*, and *maintenance* of software
  - Assure the quality of the process and the product

“Engineering Seeks Quality”

- So, the goal of software engineering is the production of quality software
- However, software is inherently complex
  - the complexity of software systems often exceeds the human intellectual capacity
- The task of the software development team is to engineer the illusion of simplicity

Three Steps Revisited

Requirements

Implementation

Quality Assurance

1. What should I do?

2. Doing it!

3. Did I actually do it?

A sequential model of software development

Five Steps

Requirements

Design

Implementation

Quality Assurance

Evolution

A sequential model of software development

The Five Steps/Phases

- Requirements
  - understanding the problem domain
  - communication between stakeholders
- Design
  - engineer a solution that addresses the requirements
  - technology, algorithms, architecture, interfaces...

The Five Steps/Phases

- Implementation
  - realizing the design
    - documentation, configuration, standards, tools, ...
- Quality Assurance
  - ensuring that the implementation meets quality standards
  - testing, analysis, reviews

The Five Steps/Phases

- Evolution
  - fixing problems
  - adding new functionality/address new requirements
  - while retaining existing functionality and code

Software Engineering

Change is Ubiquitous

- Manage
- change
- uncertainty
- quality

Change is Ubiquitous

Design

Requirements

Implementation

Quality Assurance

An iterative model of software development

Defined Process vs. Empirical Process

- Laying out a process that repeatedly will produce acceptable quality output is called defined process control
- When defined process control cannot be achieved because of the complexity of the intermediate activities, something called empirical process control has to be employed

Production vs. Creation

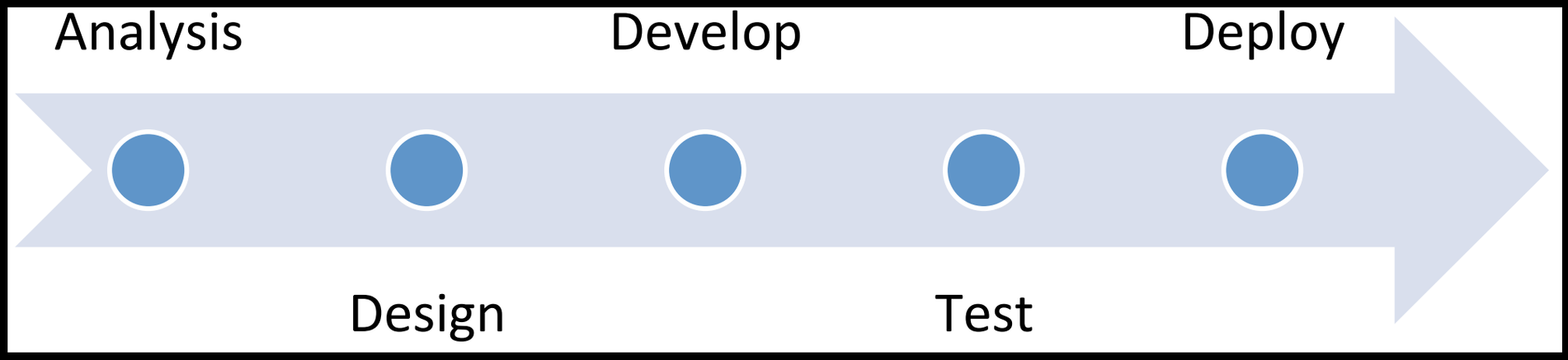
Defined Process control

- planning heavy
- assumes (more) static environment
- longer iterations
- change Management intensive
- typical pre-study heavy
- assumes good estimations
- control over actual work (seen seen as bureaucratic)

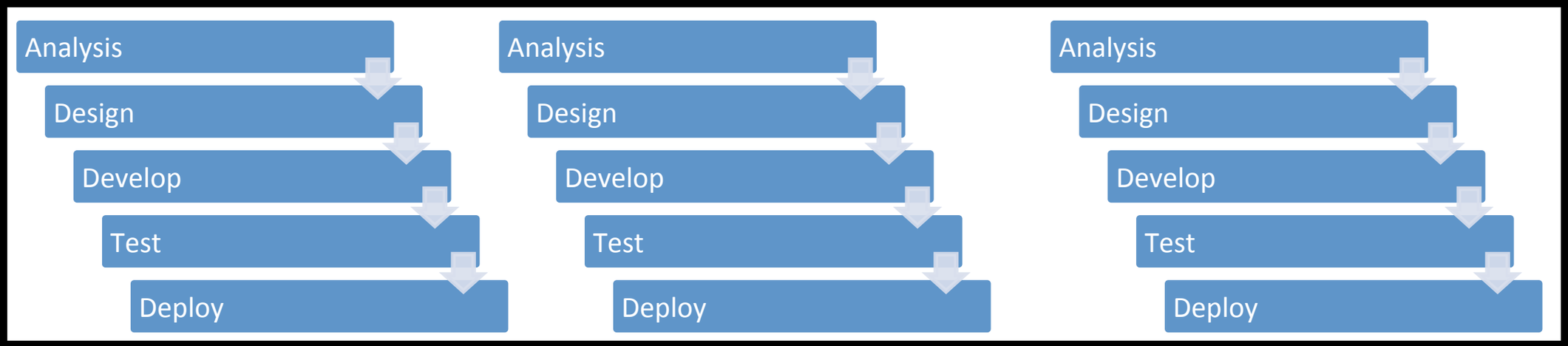
Empirical Process control

- change is reality
- shorter iterations
- problem vs. solution space (empowering the developers)
- just-enough (management, documentation, etc.)
- self organizing teams
- continuous “customer” interaction
- **NOT UNPLANNED, rather adaptive!!!**

Sequential / Waterfall / Non-agile



Iterative / Incremental / Agile



Several variants/names, e.g. Scrum, XP, FDD, etc.

To Be Continued...

- So far, discussion of need for process and what it can look like
- How can this be implemented?
- A story in two parts; development and project management
  - XP on Wednesday
  - Scrum next week

eXtreme Programming

eXtreme Programming

- XP is what it says, an extreme way of developing software
- if a practice is good, then do it all the time
- if a practice causes problems with project agility, then don't do it

eXtreme Programming

- Team, 3-10 programmers + 1 customer
- Iteration, tested and directly useful code
- Requirements, user story, written on index cards
- Estimate development time per story, prio on value
- Dev starts with discussion with expert user

eXtreme Programming

- Programmers work in pairs
- Unit tests passes at each check-in
- Stand-up meeting daily: Done? Planned? Hinders?
- Iteration review: Well? Improve? => Wall list

XP practices

- Whole Team (Customer Team Member, on-site customer)
- Small releases (Short cycles)
- Continuous Integration
- Test-Driven development (Testing)
- Customer tests (Acceptance Tests, Testing)
- Pair Programming

XP practices

- Collective Code Ownership
- Coding standards
- Sustainable Pace (40-hour week)
- The Planning Game
- Simple Design
- Design Improvement (Refactoring)
- Metaphor

Whole Team

- Everybody involved in the project works together as ONE team.
- Everybody on the team works in the same room (open workspace)
- One member of this team is the customer, or the customer representative.

Small Releases

- The software is frequently released and deployed to the customer
- The time for each release is planned ahead and are never allowed to slip. The functionality delivered with the release can however be changed right up to the end
- A typical XP project has a new release every 3 months
- Each release is then divided into 1-2 week iterations

Continuous Integration

- Daily build
  - A working new version of the complete software is released internally every night
- Continuous build
  - A new version of the complete software is built as soon as some functionality is added, removed or modified

Test-Driven Development

- No single line of code is ever written, without first writing a test that tests it
- All tests are written in a test framework like JUnit so they become fully automated

Customer Tests

- The customer (or the one representing the customer) writes tests that verifies that the program fulfills his/her needs

Pair Programming

- All program code is written by two programmers working together; a programming pair
- Working in this manner can have a number of positive effects:
  - better code quality
  - fun way of working
  - skills spreading
  - ...

Collective Code Ownership

- All programmers are responsible for all code
- You can change any code you like, and the minute you check in your code somebody else can change it
- You should not take pride in and responsibility for the quality of the code you written yourself but rather for the complete program

Coding standards

- In order to have a code base that is readable and understandable by everybody the team should use the same coding style

Sustainable Pace

- Work pace should be constant throughout the project and at such a level that people do not drain their energy reserves
- Overtime is not allowed two weeks in a row

Simple design

- Never have a more complex design than is needed for the current state of the implementation
- Make design decisions when you have to, not up front

Design Improvement

- Always try to find ways of improving the design
- Since design is not made up front it needs constant attention in order to not end up with a program looking like a snake pit
- Strive for minimal, simple, comprehensive code

Metaphor

- Try to find one or a few metaphors for your program
- The metaphors should aid in communicating design decisions and intends
- The most well known software metaphor is the desktop metaphor

User Stories

- One or more sentences in the everyday or business language
- Captures what a user does or needs to do as part of his or her job function
- Quick way of handling requirements without formalized requirement documents
- Respond faster to rapidly changing real-world requirements

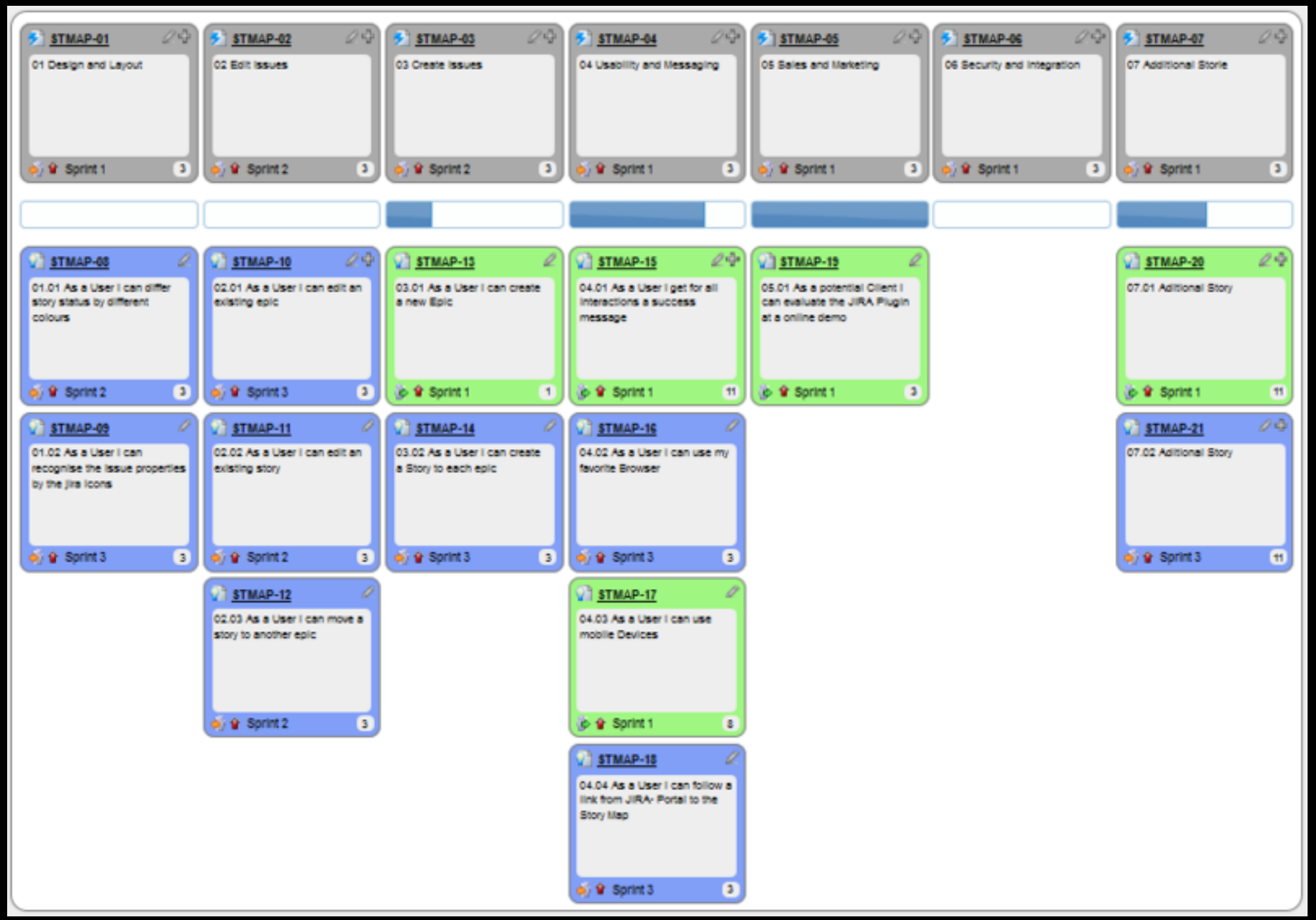
User Stories

- "As a <role>, I want <goal/desire> (so that <benefit>")
- "As <who> <when> <where>, I <what> because <why>."
- *"As a user, I want to search for my customers by their first and last names."*
- *"As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save."*

Benefits

- Represent small chunks of business value that can be implemented in a period of days to weeks
- Needing very little maintenance
- Allowing projects to be broken into small increments
- Being suited to projects where the requirements are volatile or poorly understood. Iterations of discovery drive the refinement process
- Making it easier to estimate development effort
- Require close customer contact throughout the project so that the most valued parts of the software get implemented

Story Maps


STMAP-01
01 Design and Layout
Sprint 1
3
STMAP-02
02 Edit Issues
Sprint 2
3
STMAP-03
03 Create Issues
Sprint 2
3
STMAP-04
04 Usability and Messaging
Sprint 1
3
STMAP-05
05 Sales and Marketing
Sprint 1
3
STMAP-06
06 Security and Integration
Sprint 1
3
STMAP-07
07 Additional Storie
Sprint 1
3
STMAP-08
01.01 As a User I can differ story status by different colours
Sprint 2
3
STMAP-10
02.01 As a User I can edit an existing epic
Sprint 3
3
STMAP-13
03.01 As a User I can create a new Epic
Sprint 1
1
STMAP-15
04.01 As a User I get for all interactions a success message
Sprint 1
11
STMAP-19
05.01 As a potential Client I can evaluate the JIRA Plugin at a online demo
Sprint 1
3
STMAP-20
07.01 Aditional Story
Sprint 1
11
STMAP-09
01.02 As a User I can recognise the issue properties by the jira icons
Sprint 3
3
STMAP-11
02.02 As a User I can edit an existing story
Sprint 2
3
STMAP-14
03.02 As a User I can create a Story to each epic
Sprint 3
3
STMAP-16
04.02 As a User I can use my favorite Browser
Sprint 3
3
STMAP-21
07.02 Aditional Story
Sprint 3
11
STMAP-12
02.03 As a User I can move a story to another epic
Sprint 2
3
STMAP-17
04.03 As a User I can use mobile Devices
Sprint 1
8
STMAP-18
04.04 As a User I can follow a link from JIRA- Portal to the Story Map
Sprint 3
3

Example

Roles

- **Instructor**: Expected to use the website frequently, often once a week. Through the company's telephone sales group, an Instructor frequently places similar orders (for example, 20 copies of the same book). Proficient with the website but usually somewhat nervous around computers. Interested in getting the best prices. Not interested in reviews or other "frills."
- **Experienced Sailor**: Proficient with computers. Expected to order once or twice per quarter, perhaps more often during the summer. Knowledgeable about sailing but usually only for local regions. Very interested in what other sailors say are the best products and the best places to sail.

Stories for Experienced Sailors

- A user can search for books by author, title or ISBN number
- A user can view detailed information on a book. For example, number of pages, publication date and a brief description
- A user can put books into a “shopping cart” and buy them when she is done shopping
- A user can remove books from her cart before completing an order

Stories for Experienced Sailors

- To buy a book the user enters her billing address, the shipping address and credit card information
- A user can rate and review books
- A user can establish an account that remembers shipping and billing information
- A user can edit her account information (credit card, shipping address, billing address and so on)
- A user can put books into a "wish list" that is visible to other site visitors

Stories for Instructors

- A user can view a history of all of his past orders.
- A user can easily re-purchase items when viewing past orders.
- The site always tells a shopper what the last 3 (?) items she viewed are and provides links back to them. (This works even between sessions.)

Assessment

- “A user can search for books by author, title or ISBN number”
- What does that mean? Either or, or any possible combination?
- Fix!
- “A user can do a basic simple search that searches for a word or phrase in both the author and title fields”
- “A user can search for books by entering values in any combination of author, title and ISBN”

Week 2

- **Monday**: SCRUM
- **Tuesday**: Pitch your app ideas
- **Wednesday**: Git and Android
- **Friday**: Submit app vision