NAU NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

CS386 - Software Engineering

Unit 4.3

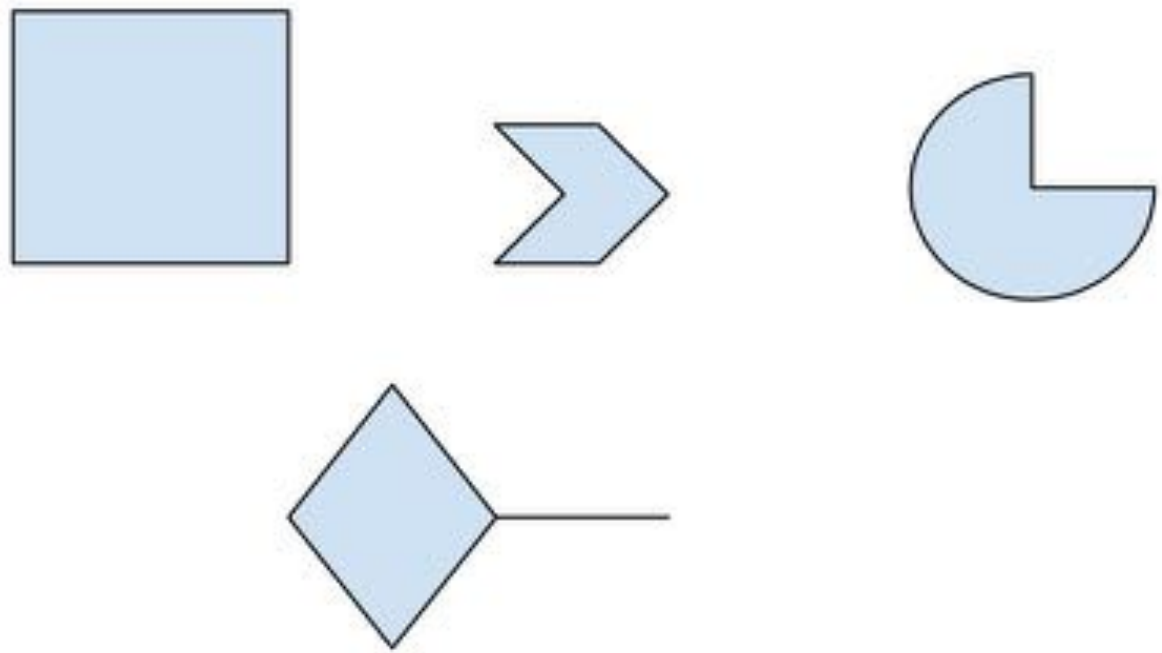
User Stories

Prof. Marco Aurélio Gerosa, PhD
<Marco.Gerosa@nau.edu>
http://nau.edu/siccs/faculty/marco-gerosa/

Group dynamics – Part I

1. Form groups of 4 people
   Choose 2 *developers* and 2 *requirement analysts*
2. *Developers* leave the room for 3 minutes
3. *Requirement analysts* write the textual specification of a figure. *Developers* will draw it after they return
4. *Developers* draw it within 3 minutes
5. *Analysts* must not talk

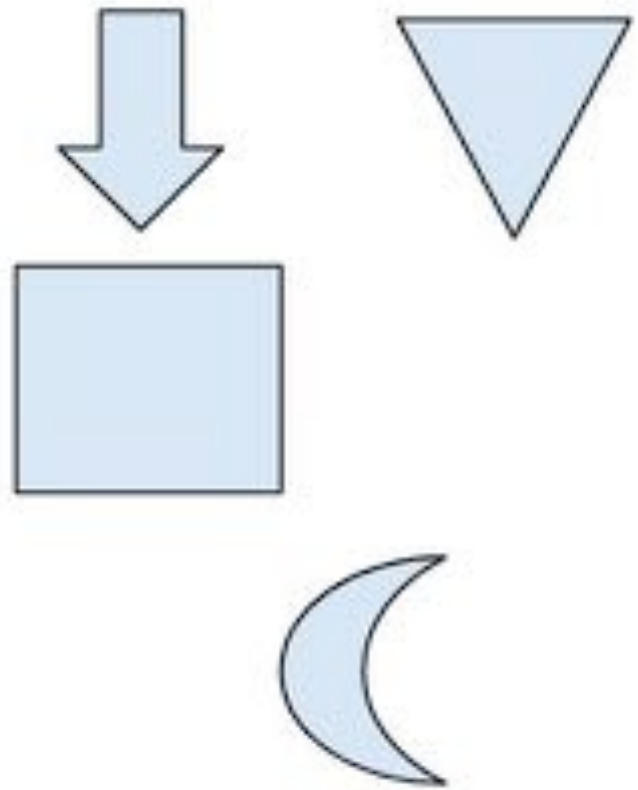
Acknowledgement: Special thanks for Dr. Maurício Aniche for preparing this exercise



Time’s up!

Group dynamics – Part II

We will repeat the dynamics, but **now the *requirement analysts* may give feedback when asked**



Time’s up!

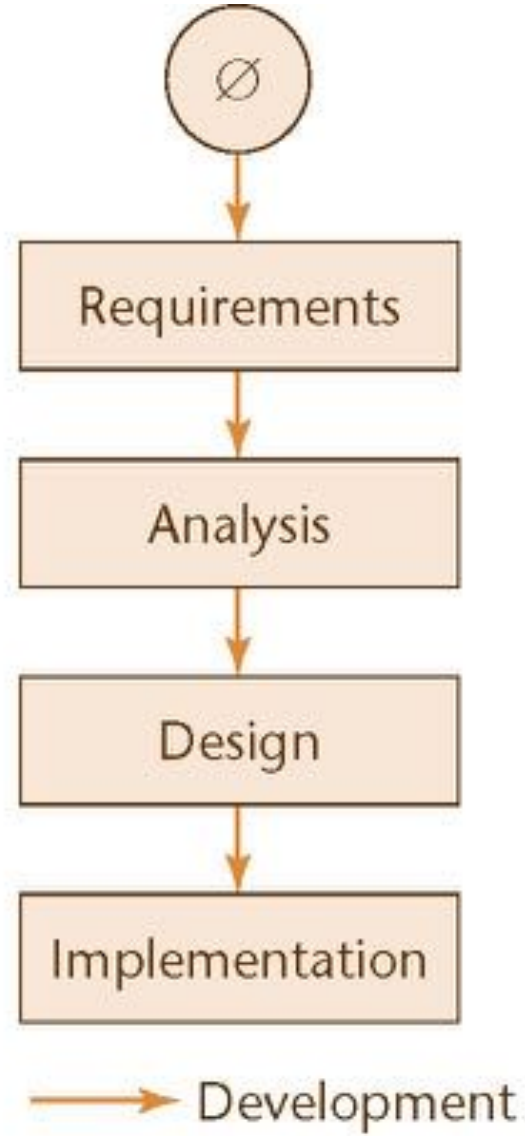
Reflecting about it

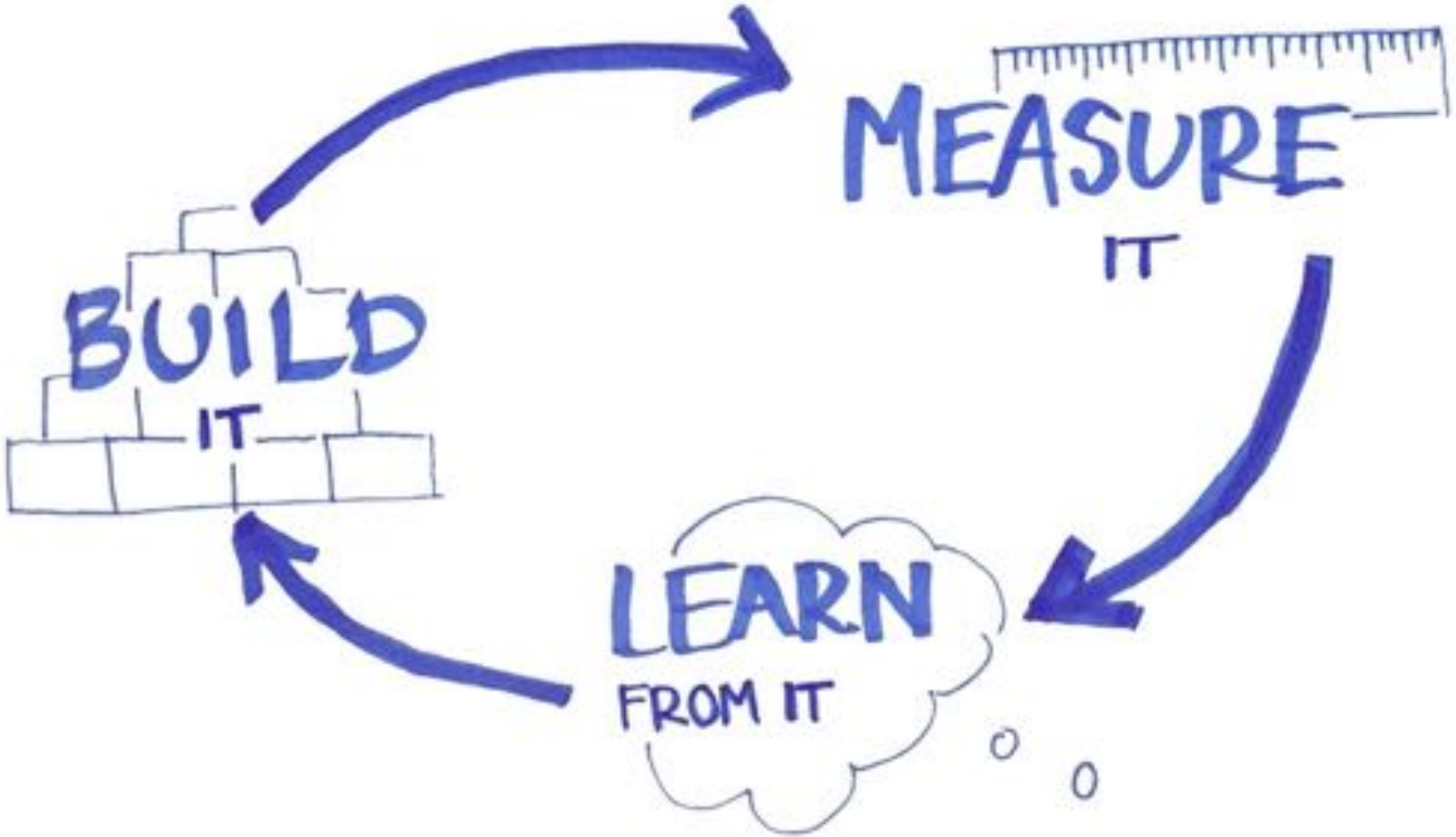
What are the benefits and drawbacks of each approach?


Code
Developer
Business Analyst
Client
_compiler


FEED
BACK

Waterfall vs. iterative cycle





Moving target problem

Requirements don’t stop changing

You can’t avoid the moving target problem, since the software operates in a changing world

- Companies merge, laws change, new business strategies emerge, etc.
- Underlining technologies also change

http://www.the3dstudio.com/product_details.aspx?id_product=113503

Feedback

Help analysts to adjust the requirements

Analysts can collect explicit feedback (verbal) and implicit feedback (observation)

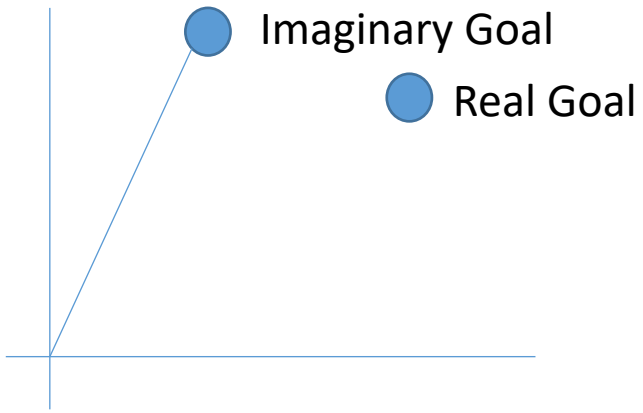
Use of prototypes

- Paper prototyping

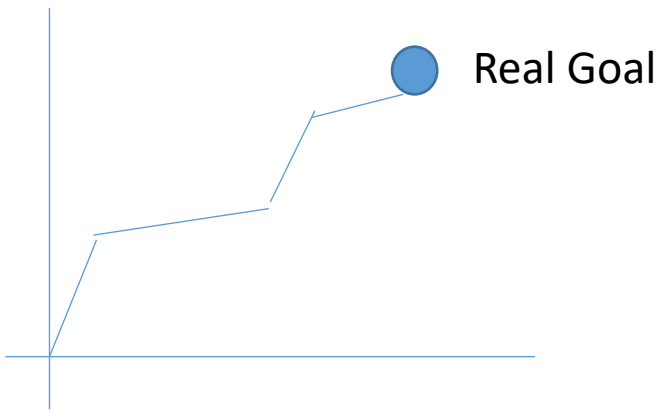
However, be careful with prototypes:

- They can make the impression of an almost ready product
- They can move the focus of the discussion to interface details
- The use of prototype simulates the use of the system, real needs appear when the system is actually used
- Do not directly incorporate the prototype code to the final product (avoid poorly written code) – it is cheaper to throw away than to convert a prototype (Brooks, 1975)

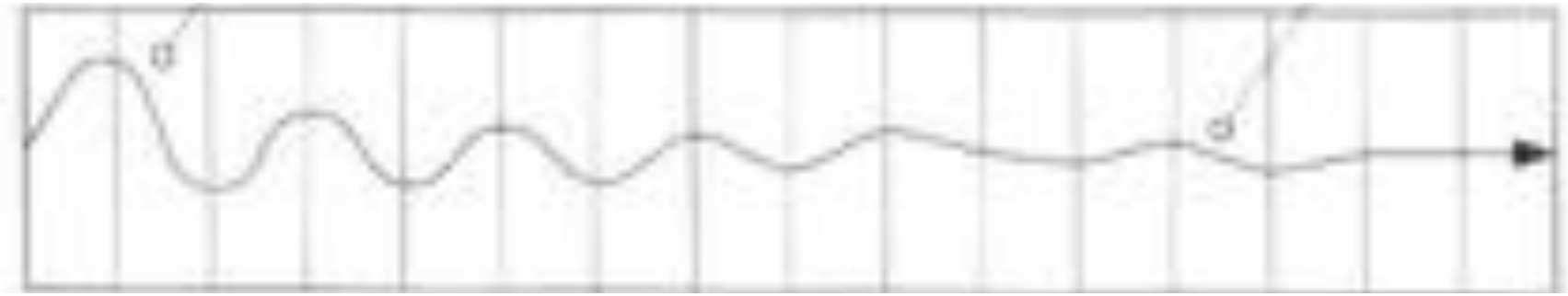
Feedback



Without feedback



With feedback



User stories

A written user story is a very short narrative—a sentence or two—describing some small piece of functionality that has business value.

Instead of specifying the details of the requirements, the user stories are invitations for the dialogue.

Proposed in the Extreme Programming (1998) method and used in several projects that adopt agile methods

Use everyday or business language that captures what users do or need to do as part of their job

Informal examples

“Students can get parking passes”

“Parking passes can be paid by credit card.”

“Parking passes can be paid via PayPal.”

“Teachers can report students' grades.”

“Students can get their class schedule.”

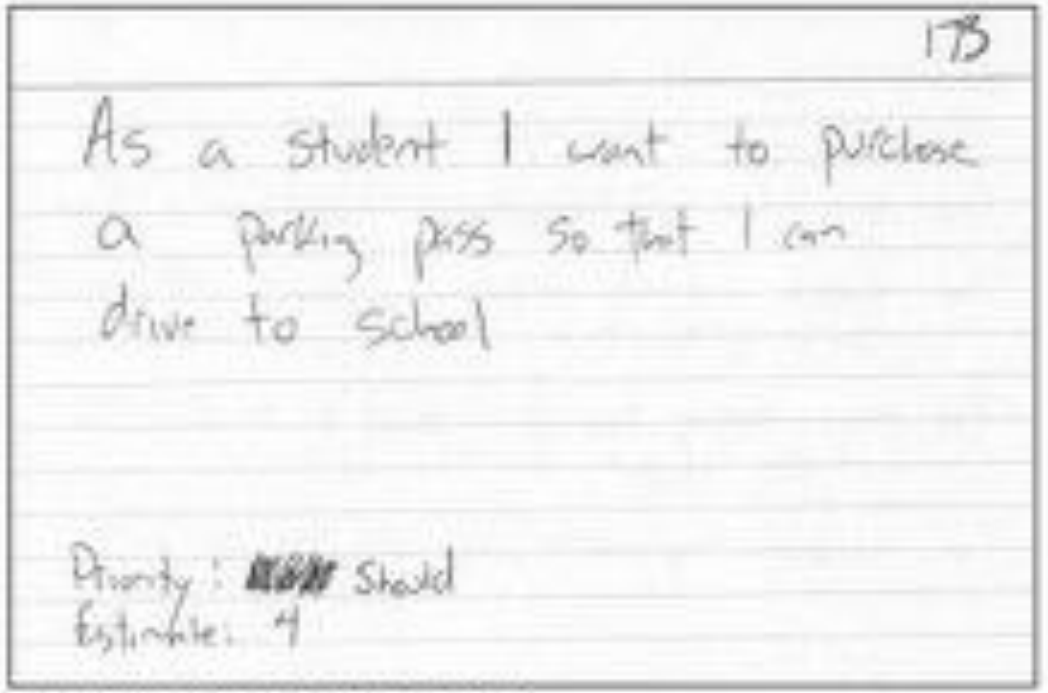
“Students can request their official transcripts.”

http://www.agilemodeling.com/artifacts/userStory.htm

Cards

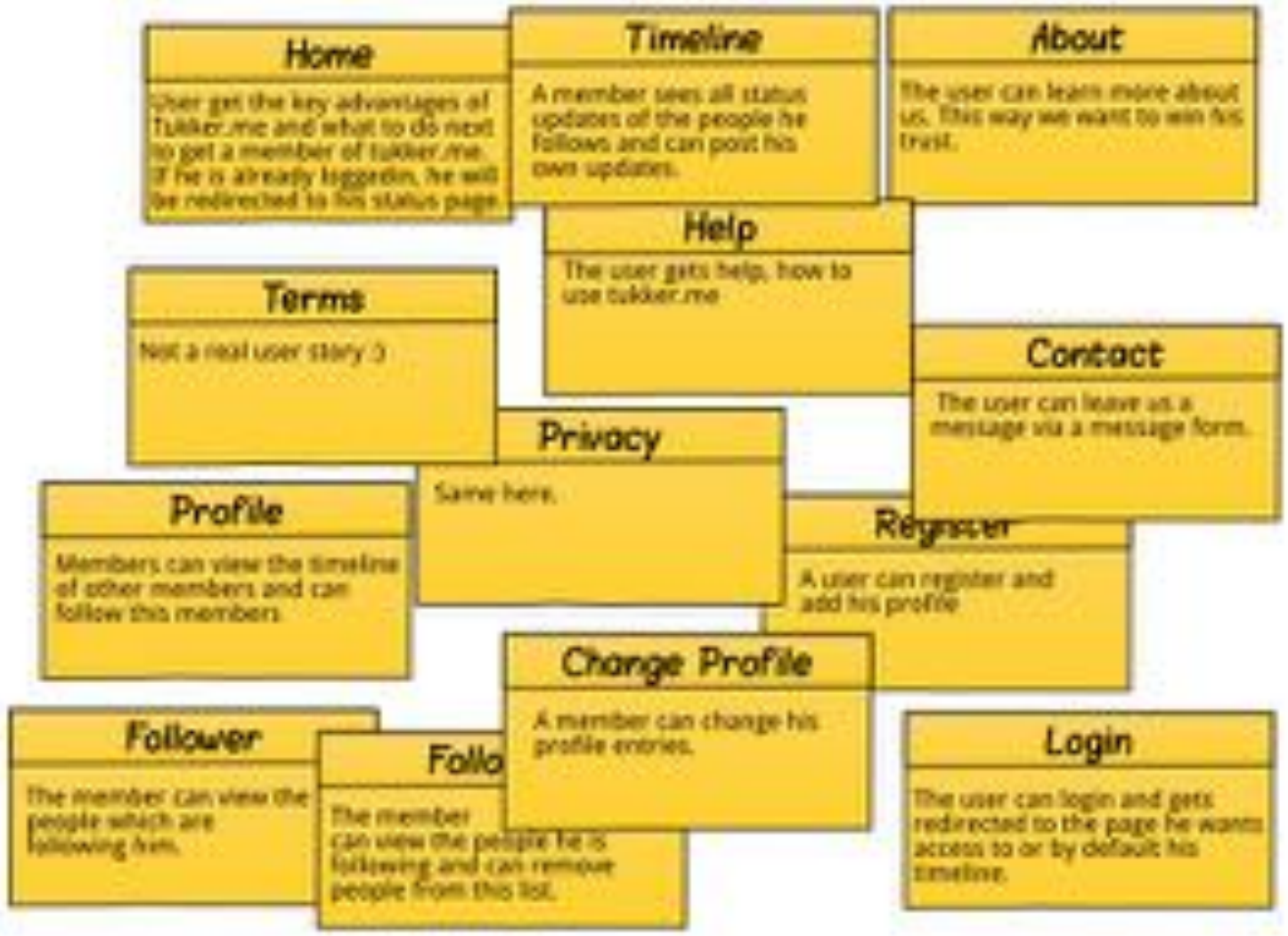
You can only write what fits in a handwritten card.

It contains only what is needed for the developers to produce a reasonable estimate of the effort to implement it.



http://www.agilemodeling.com/artifacts/userStory.htm

Post-its take place



http://killer-web-development.com/section/4/1

Exercise

Write a few user stories for recipe contributors in a social recipe web site.

Which one do you prefer?

1: “Users can search emails via the web interface to find relevant information.”

2: “As a User, I want to search emails via the web interface so that I can find relevant information.”

Structured user stories

As a

[user role]

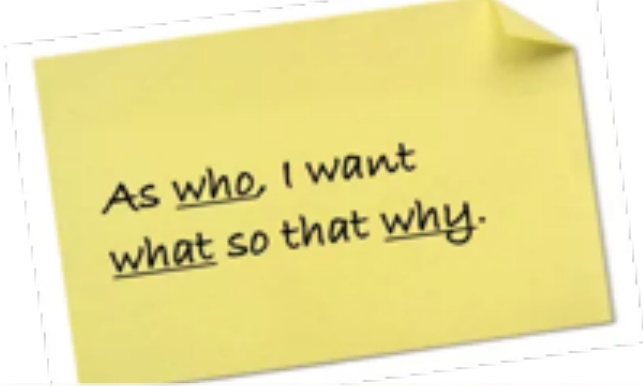
I want to

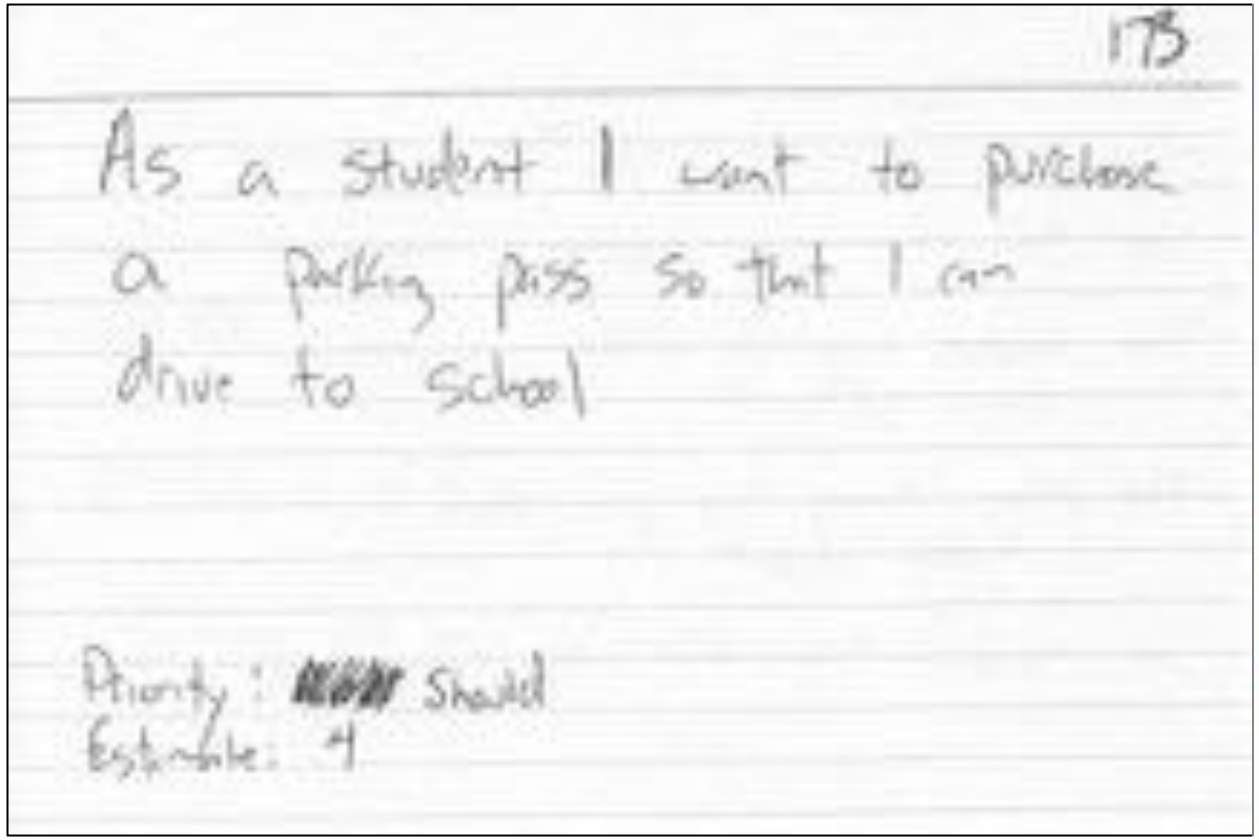
[perform an action]

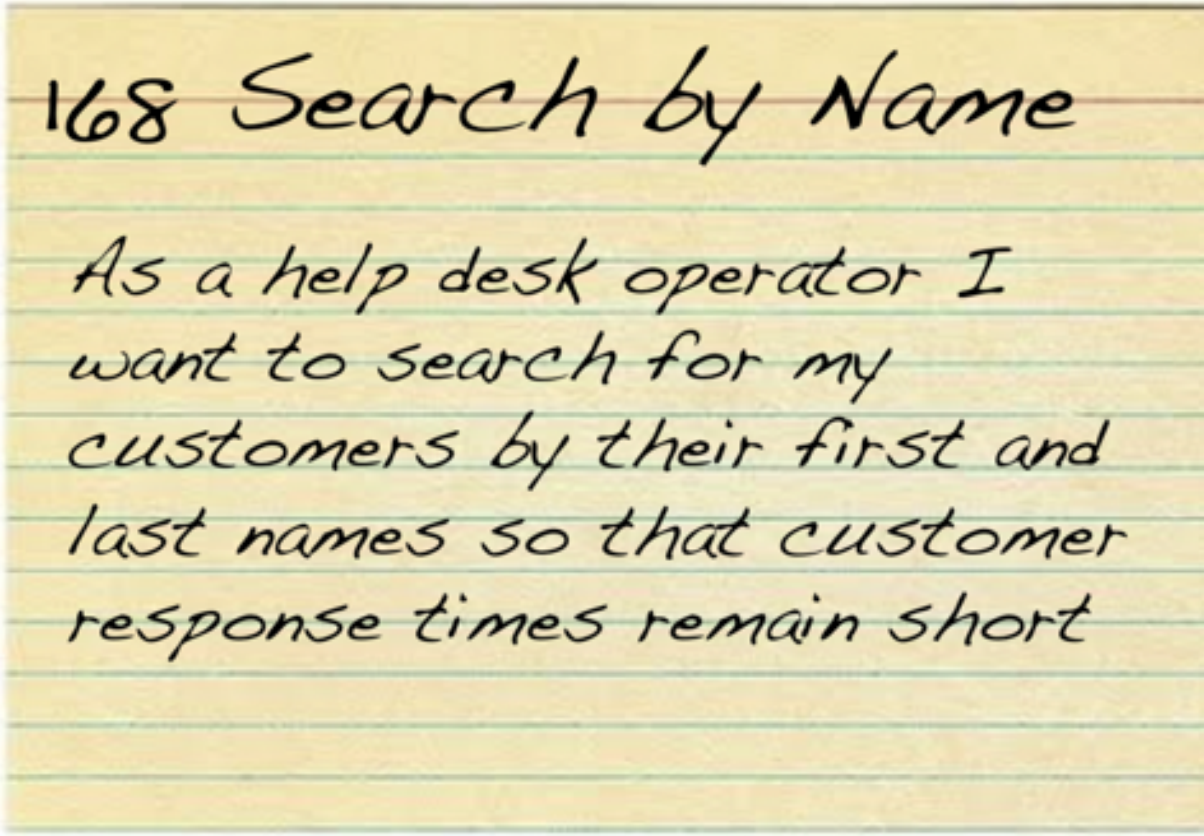
so that

[some value is realized]

Examples







http://www.agilemodeling.com/artifacts/userStory.htm

Exercise

Revisit your user stories rewriting them in the format **As a [user role], I want to [action] so that [goal]**.

Some important considerations

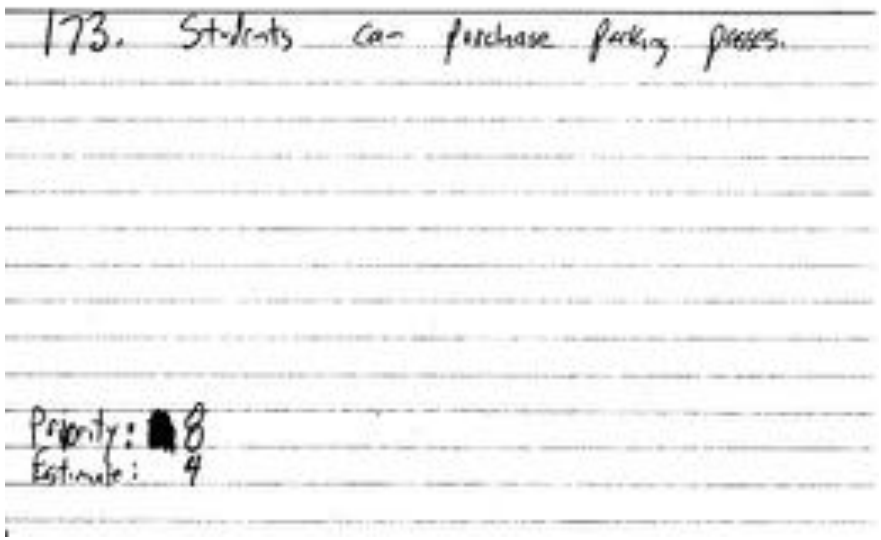
Stakeholders/product owners write stories

Remember the non-functional requirements

Leave room for estimation and priority

Include a unique identifier (for tracking between artifacts)

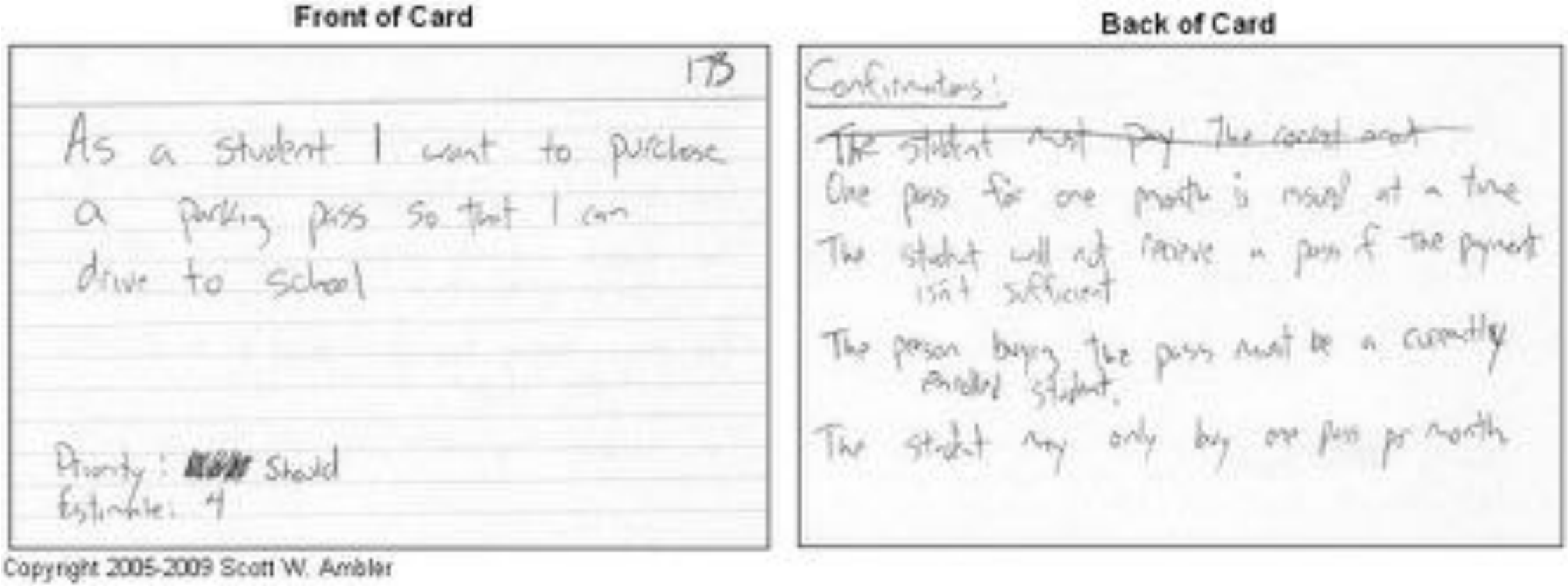
Story cards serve as reminders not contracts



http://www.agilemodeling.com/artifacts/userStory.htm

More about stories

The back of the card can be used to describe acceptance tests



http://www.agilemodeling.com/artifacts/userStory.htm

Guidelines for writing

1. Keep the UI out as long as possible
2. Include user roles in stories rather than saying “user”
3. Write for a single user (“a manager” not “managers”)
4. Use active voice

Technical stories

Sometimes the "as/I want/so that" structure sounds odd to write technical stories. In these cases, the structure can be skipped. E.g.:

> "*We need an API call to filter mail by a service provider so only mails by that provider shows up*."

However, **acceptance criteria** are important. E.g.:

- Verify that the request get through the service layers and receive a reply within 2 seconds
- Verify that the request has necessary OAuth permission to be exposed externally
- Verify that the request is generated in both XML and JSON formats
- Verify that records are returned from oldest to newest

http://rgalen.com/agile-training-news/2013/11/10/technical-user-stories-what-when-and-how
https://onproductmanagement.org/2016/09/20/writing-technical-user-stories/

Examples of technical stories

- **Product Infrastructure** – stories that directly support requested functional stories. This could include new and/or modified infrastructure. It might also identify refactoring opportunities, but driven from the functional need.
- **Team Infrastructure** – stories that support the team and their ability to deliver software. Often these surround tooling, testing, metrics, design, and planning. It could imply the team "creating" something or "buying and installing" something.
- **Refactoring** – these are stories that identify areas that are refactoring candidates. Not only code needs refactoring, but also it can often include designs, automation, tooling, and any process documentation.
- **Bug Fixing** – either clusters or packages of bugs that increase repair time or reduce aggregate of testing time. So this is more of an efficiency play.
- **Spikes** – research stories that will result in learning, architecture & design, prototypes, and ultimately a set of stories and execution strategy that will meet the functional goal of the spike. Spikes need to err on the side of prototype code over documentation as well, although I don't think you have to "demo" every spike.

http://rgalen.com/agile-training-news/2013/11/10/technical-user-stories-what-when-and-how

Good or bad?

1. “As a recipe contributor, I want to see my recipes, ratings, and comments so that I can find this information in one place.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Bad example

=> There are independent units of work inside a single story <=

1. “As a recipe contributor, I want to see my recipes, ratings, and comments so that I can find this information in one place.”

Better examples:

2. “As a recipe contributor, I want to see a list of the recipes that I have contributed so that I can select one for editing or reviewing.”
3. “As a recipe contributor, I want to see the comments for each of my recipes so that I can respond to any questions.”
4. “As a recipe contributor, I want to see the ratings for each of my recipes so that I can improve my contributions.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Good or bad?

1. “As a recipe contributor, I want to post recipes including title, description, date, instructions, pictures, and a list of attributes to choose from multi-select list boxes such as season, culture, time to prep, recipe size, estimated cost so that my recipes can be displayed in targeted search results.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Bad example

=> Too much detail <=

1. “As a recipe contributor, I want to post recipes including title, description, date, instructions, pictures, and a list of attributes to choose from multi-select list boxes such as season, culture, time to prep, recipe size, estimated cost so that my recipes can be displayed in targeted search results.”

Better example:

2. “As a recipe contributor, I want to post recipe and photo information with searchable fields so that users can find and view my recipe.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Good or bad?

1. “As a developer, I want to create a recipe database table, so that I can store recipe details.”
2. “As a developer, I want to call a recipe web service to retrieve recipe details.”
3. “As a developer, I want to create the recipe list UI.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Bad example

=> Leaking implementation details. Value to the user is not clear. <=

1. “As a developer, I want to create a recipe database table, so that I can store recipe details.”
2. “As a developer, I want to call a recipe web service to retrieve recipe details.”
3. “As a developer, I want to create the recipe list UI.”

Better example:

4. “As a recipe contributor, I want to see a list of the recipes that I have contributed so that I can select one for editing or reviewing.”

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Good or bad?

1. As a website owner, I want the website to be secure so that I don't have angry customers.
2. As a website owner, I want to the site to be useable so that I can enter recipes easily.

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Bad examples

=> Too vague. Not measurable. Not testable. <=

1. As a website owner, I want the website to be secure so that I don't have angry customers.
2. As a website owner, I want to the site to be useable so that I can enter recipes easily.

Better examples:

3. "As a recipe contributor, I want to be the only one who can edit my recipes so that I have confidence in the site."
4. As a recipe contributor, I want to post basic recipe and photo information so that users can search for and view my recipe.
   - Acceptance Test: Recipes should be entered within 30 seconds 80% of the time.

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Good or bad?

1. As a recipe contributor, I want to see detailed statistical analysis so that I can understand which of my recipes are the most effective
2. As a recipe contributor, I want the submitted date time for each recipe to be set automatically so that I can know the submission date without needing to type.

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Bad examples

=> Too big or too small. <=

1. As a recipe contributor, I want to see detailed statistical analysis so that I can understand which of my recipes are the most effective
2. As a recipe contributor, I want the submitted date time for each recipe to be set automatically so that I can know the submission date without needing to type.

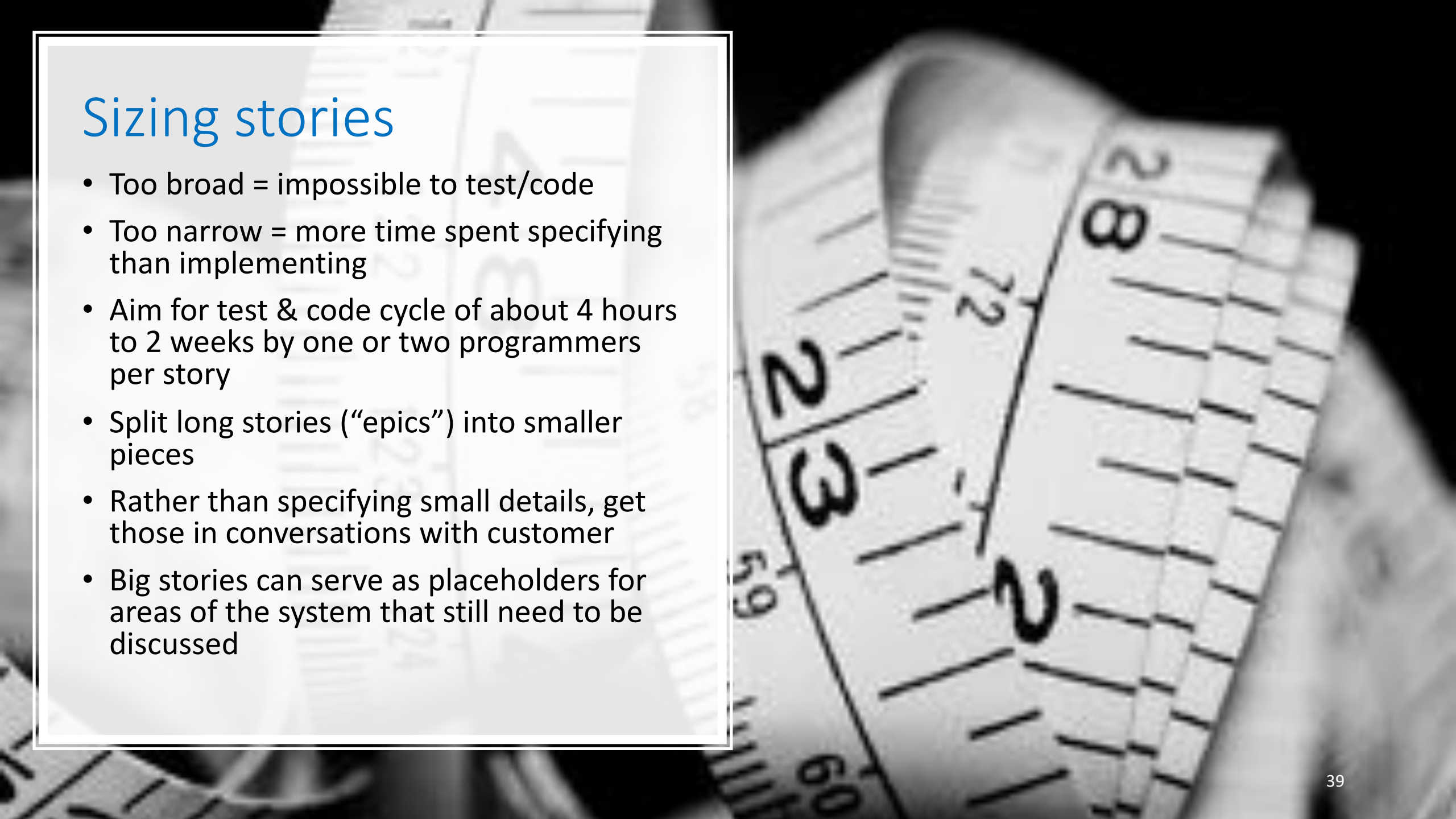
Better example:

3. As a recipe contributor, I want to see the number of views per recipe per day for each of my recipes so that I can understand which of my recipes are the most effective.

https://www.slideshare.net/SteveRogalsky/user-story-workshop-10505314

Sizing stories

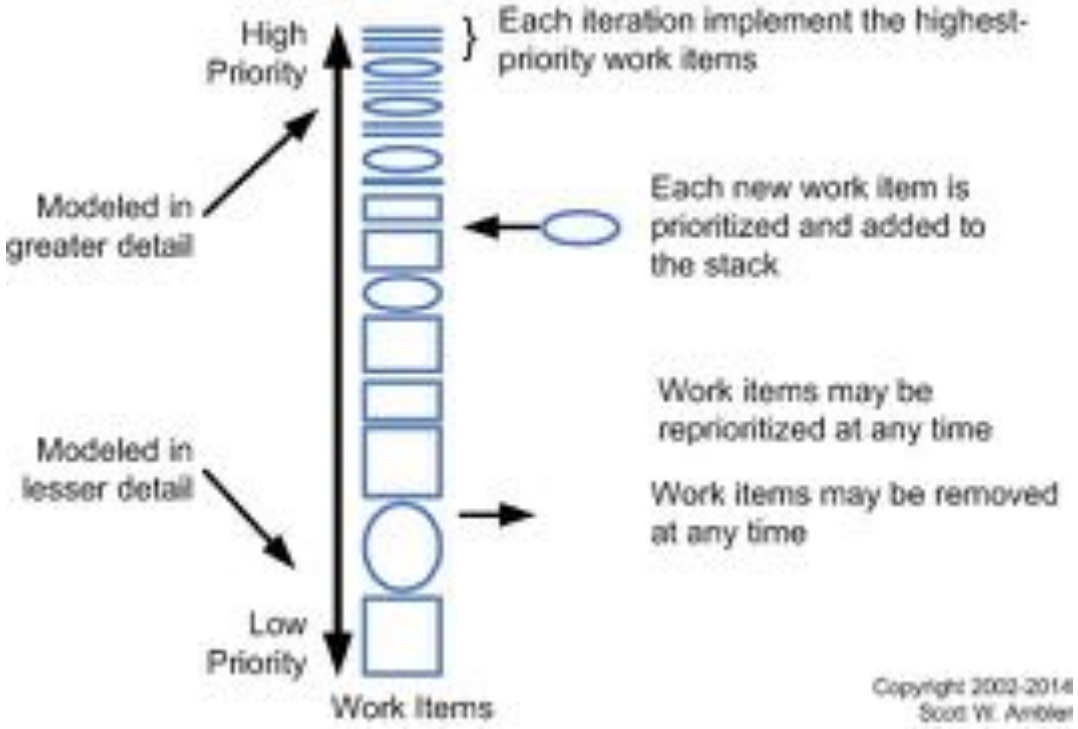
- Too broad = impossible to test/code
- Too narrow = more time spent specifying than implementing
- Aim for test & code cycle of about 4 hours to 2 weeks by one or two programmers per story
- Split long stories (“epics”) into smaller pieces
- Rather than specifying small details, get those in conversations with customer
- Big stories can serve as placeholders for areas of the system that still need to be discussed



Planning

Prioritization - customer prioritizes and stories are answered in order of priority

- Numerical scale
- Scale MoSCoW (Must, Should, Could, Will not)



http://www.agilemodeling.com/artifacts/userStory.htm

Planning

Estimate - Developers estimate

- A story has to be feasible to be implemented by a developer in the maximum time of an iteration
- If necessary, the story **should be divided**
- In the same way, it may be necessary to **merge stories**

Long stories are commonly called **EPICS**

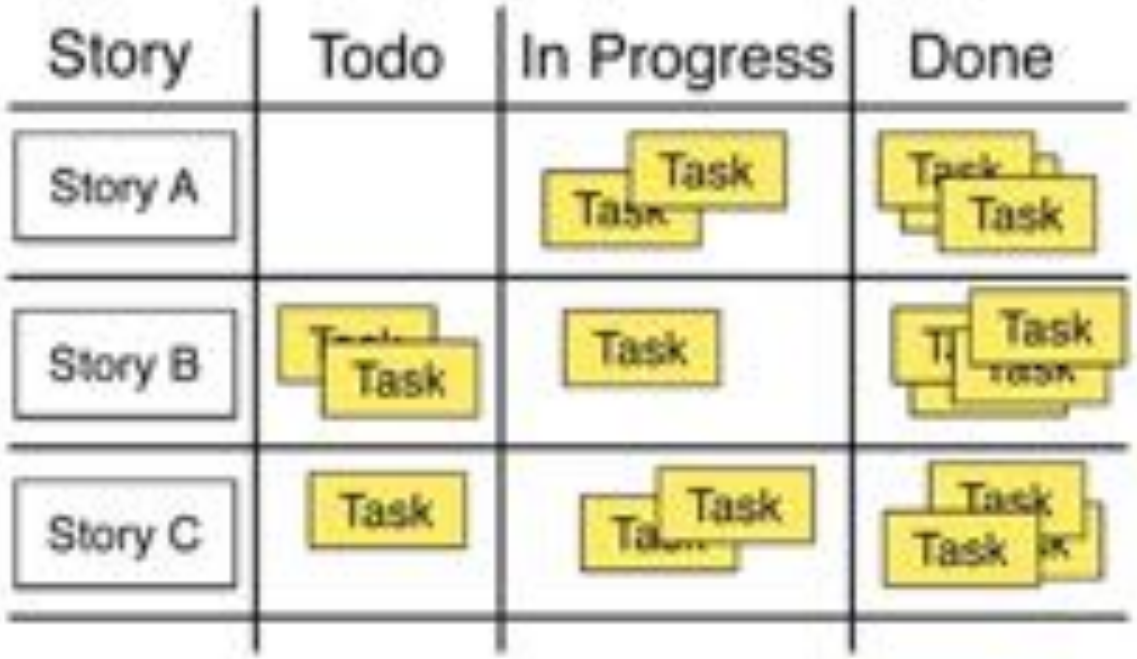
- Usually **too long** to be implemented in a **single iteration**
- Usually **start with lower priority** and go up in the priority stack over time
- Dismembered when necessary

Stories can be grouped into "**themes**"

Tasks

Stories can be (usually are) decomposed in development tasks

- Intermediate deliverables make it easier to follow-up, understand what to implement, and dividing the tasks (teamwork)



http://mrako.com/blog/2013/01/27/how-to-write-good-user-stories/

Kanban

http://www.agilebuddha.com/agile/story-mapping-andvs-process-maps/

http://www.agilebuddha.com/agile/story-mapping-andvs-process-maps/

User stories vs. Use cases

	User Stories	Use Cases
Similarities	• Generally formulated in users' everyday language. They should help the reader understand what the software should accomplish. • Must be accompanied by acceptance testing procedures (acceptance criteria) for clarification of behavior where ambiguous.	• Written in users' everyday business language, to facilitate stakeholder communications. • Must be accompanied and verifiable by test cases.
Differences	• XP stories (and similar things, often called features) break requirements into chunks for planning purposes. Stories are explicitly broken down until they can be estimated as part of XP's release planning process.[14] • Provide a small-scale and easy-to-use presentation of information, with little detail, thus remaining open to interpretation, through conversations with on-site customers. • Usually written on small note cards. • Stories are usually more fine-grained because they have to be entirely buildable within an iteration (one or two weeks for XP).[14]	• Use cases organize requirements to form a narrative of how users relate to and use a system. Hence they focus on user goals and how interacting with a system satisfies the goals.[14] • Use case flows describe sequences of interactions, and may be worded in terms of a formal model. A use case is intended to provide sufficient detail for it to be understood on its own. • Usually delivered in a stand-alone document, and visualized by UML diagrams. • A small use case may correspond entirely to a story; however a story might be one or more scenarios in a use case, or one or more steps in a use case.[14]

http://en.wikipedia.org/wiki/User_story

Why user stories?

Emphasize verbal communication

Comprehensible by everyone

Right size for planning

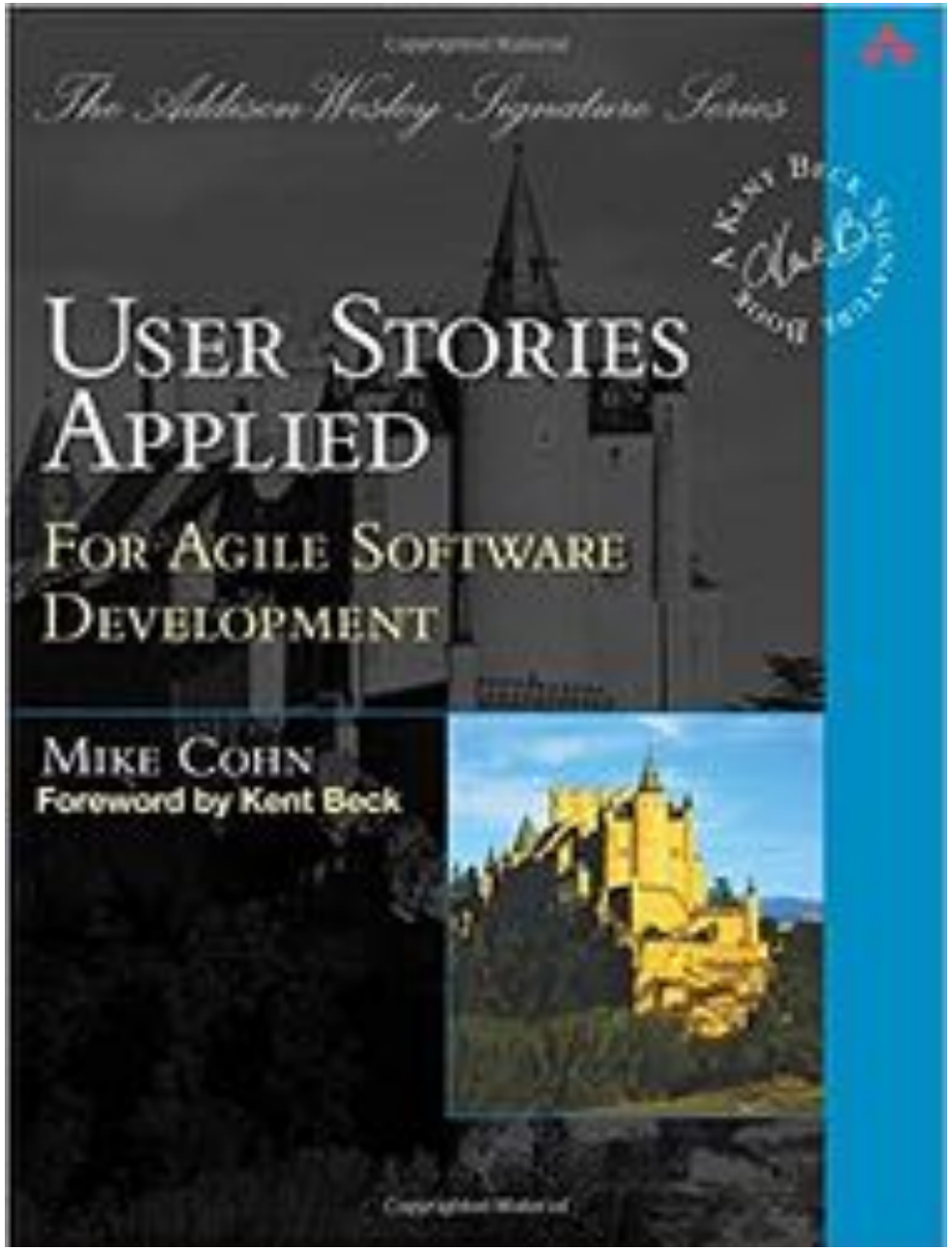
Work for iterative development

Encourage deferring detail

Encourage participatory design

http://slideplayer.com/slide/3119583/

To learn more



Exercise

Write user stories for:

- Bluetooth headphones.
- A small compact car.

Critique user stories for:

- A smartphone:
  - As an end user, I want my battery to last 72 hours so I don't have to charge it all the time.
  - As an end user, I want to be able to connect to other devices via Bluetooth so I can use Bluetooth enabled devices with my phone.
  - As an end user, I want to be able to track my location with the phone in order to aid in navigation.
- A TV:
  - As an end user, I want to be able to use my TV to surf the web so I can watch videos from any source.
  - As an end user, I want the ability to change picture and audio settings for the TV so I can dim the light and turn certain types of noises down when sleeping.
  - As an end user, I want to be able to watch Netflix directly through my TV through a built in application so I don't have to navigate there through the web.


AGILE - SCRUM
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
www.dilbert.com
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
© 2002 United Feature Syndicate, Inc.
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.