

LEAN SOFTWARE DEVELOPMENT

Part 1: from continuous integration to continuous delivery

Part 2: continuous delivery: software projects & baby steps

Part 3: agile testing

TODAY'S AGENDA

- **Recap: continuous integration & delivery (10')**
- **Exercise: continuous delivery in software projects**
 - Intro (10')
 - phase 1 (20'), discussion (10')
 - phase 2 (20'), discussion (10')

Recap

INTEGRATION IN SOFTWARE

- In software projects, we always build create small **components** that we **integrate** in larger **units**, that we integrate in complete **systems**.
- All of these units must **work together**.
 - In theory, it is easy: we define “**contracts**” between the units (communication protocols, interfaces, APIs).
 - In practice, the “contracts” are often complex and contain elements that may be **interpreted** differently by the person **creating** the component and the person **using** the component.
 - The consequence is that doing the integration is not always trivial and raises issues that must be fixed. This involves **discussion** and **collaboration** between the parties.

CONTINUOUS INTEGRATION (I)

- **When planning a software project over 5 weeks, we could:**
 - **Split** the whole system into components (mobile app, web site, back-end); split the components into smaller units (UI, algorithms, security, database, etc.).
 - **Delegate** the implementation of every unit to a member of the development team and have them **work in parallel** for 4 weeks.
 - Keep 2 days in the **5th week** to collect all the results (source code), integrate everything, 2 days to test everything and 1 day to demo/deliver the system to the customer.
 - **Fail miserably.**

CONTINUOUS INTEGRATION (2)

- **Or, during the same period, we could:**
 - **Split** the whole system into components (mobile app, web site, back-end); split the components into smaller units (UI, algorithms, security, database, etc.). **Delegate** the implementation of every unit to a member of the development team and have them **work in parallel** for 4 weeks.
 - ask every team member to **save his code in a shared repository** (e.g. git) every single day and use a software (a CI server such as Jenkins or Travis CI) to **automatically build** every component (compile) and assemble them in larger units and complete systems.
 - the same **mis-interpretations and problems** will happen, but they will be discovered very **early** and will be **easier** to fix.
 - Keep the 5th week to do the final validation and to present the entire system to the customer. Who is likely to say “**Nice, it works technically, but it is not really what I had in mind...**”.

Build the product right

VS

Build the right product

CONTINUOUS DELIVERY

- **Continuous delivery and continuous deployment are extensions of continuous integration.**
- We do not only build software packages automatically on daily basis. We also **make them frequently available to users** (“eat your own dog food”, “friendly users”, “canary releases”).
- **At the beginning, users will have very little functionality available.** But they will already be able to get a more concrete idea of the product. They will be able to give feedback after realizing that they forgot to specify something (or that they changed their mind).
- We work in **short iterations** (**weekly**, if not **daily**) and at the end of iteration, we aim to **deliver new value to the users.**
- At the end of the five weeks, we are **more likely to have built the right product and to have built it right.** And it is very possible that it is a bit different from what we envisioned at the beginning of the project.

Plan your project as a series of
“**baby steps**”, where you add
more and more functionality around
the “**skeleton**” that deliver ASAP.

EXERCISE



Introduction

THE PRODUCT

**I don't want
to get wet
when
commuting...**

...therefore I would
like to know if I
need to take my
umbrella today.

*I would love a service that would send me a
notification 10 minutes before I leave home and that
lets me know if I need to take my umbrella or not.*



<http://www.geeky-gadgets.com/the-ambient-umbrella-30-12-2009/>

Step 1: What problems do we need to solve?

You are not developer.

Nevertheless, you should be able to identify some of the problems that must be solved when building this service.

What are the “big questions” that your development team will need to answer?

Take 5 minutes and write down your questions.

QUESTIONS

How do we **decide** when is the right time to send the SMS to a current user?

How do we **predict** if it is going to rain today or not?

How do we **send a notification to the user** so that he knows whether to take the umbrella or not?

How do we handle the **on-boarding** (registration) of users?

How can we **have confidence** in the estimation of our users departure times?
How do we deal with estimation errors?

How do we know if our users are **happy**?

How do users **interact** with the service (app? site?)

What does the system **architecture** look like?

Step 2: What does the architecture look like?

Once again, you are not a developer.

Nevertheless, based on your experience with existing software products, how would you describe the **high-level** architecture that needs to be built?

Take 5 minutes and write down your questions.

ARCHITECTURE

Web browser

“SmartUmbrella”
iOS app

“SmartUmbrella”
android app

Mobile phone

“SmartUmbrella”
web site

“SmartUmbrella”
Back-end server
(cloud)

predictivite algorithms

SMS
Broker

Apple
Push

Android
Push

Weather web site

Weather API
provider 1

Weather API
provider 2

Weather API
provider 3

Phase I

CONTEXT

- You have conducted a **user survey** and you have an initial understanding of what your customers want (or believe that they want).
- Your **development team** has done a preliminary analysis. They have identified the tasks that need to be done in order to build the software service. They have also estimated the time required to complete every task.
- Your responsibility is now to **plan and organize the project**. You have to decide in what order the team will need to do the tasks and when you will deliver the product to the customers.
- **Work in teams of 5-6 students. Grab one “developer” in every group.**

FIRST USER SURVEY

80% of users have a **regular daily commuting** schedule.

There are different ways to **notify** the user: e-mail, SMS, iOS/android push.

Users **slightly prefer** to receive an iOS/android push notification than an SMS.

60% of our target users have an **iOS** device. 35% have an **android** device.

Users **prefer** to receive an SMS than an email notification.

Users **strongly prefer** to receive a notification than having to visit a web site to get the information (push vs pull)

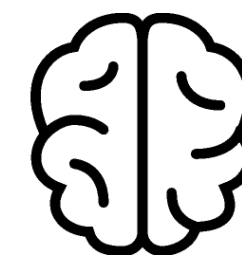
TEAM TASKS

Mechanism: our back-end is able to send an Android push notification



2 days

Implement a FORECAST algorithm, which computes the probability that it will rain today for a given registered user. We use the data provided by an API.



5 days

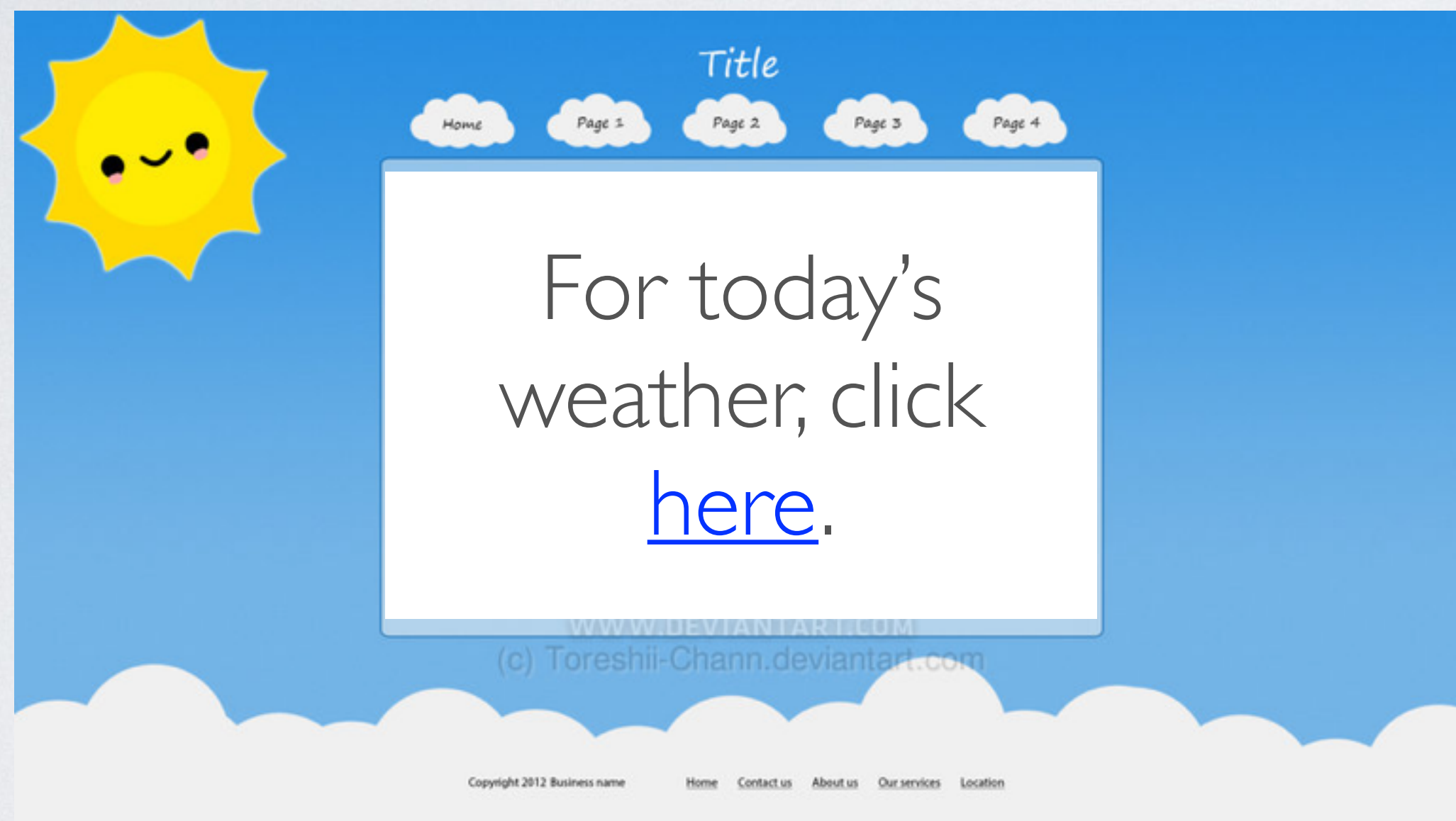
TEAM TASKS

- In the envelope, you have one card for every implementation task.
- To keep things simple, we assume that the “**whole team**” (dev, UX, QA) works jointly on the same task and for the same amount of time.
- The number on the card (e.g 5 days) is a value in “**team-day**” (vs man-day). In one week, you have 5 team-days available. If you decide to work on two “5 days” cards in parallel, it will take 2 weeks to complete both of them.

TEAM TASKS

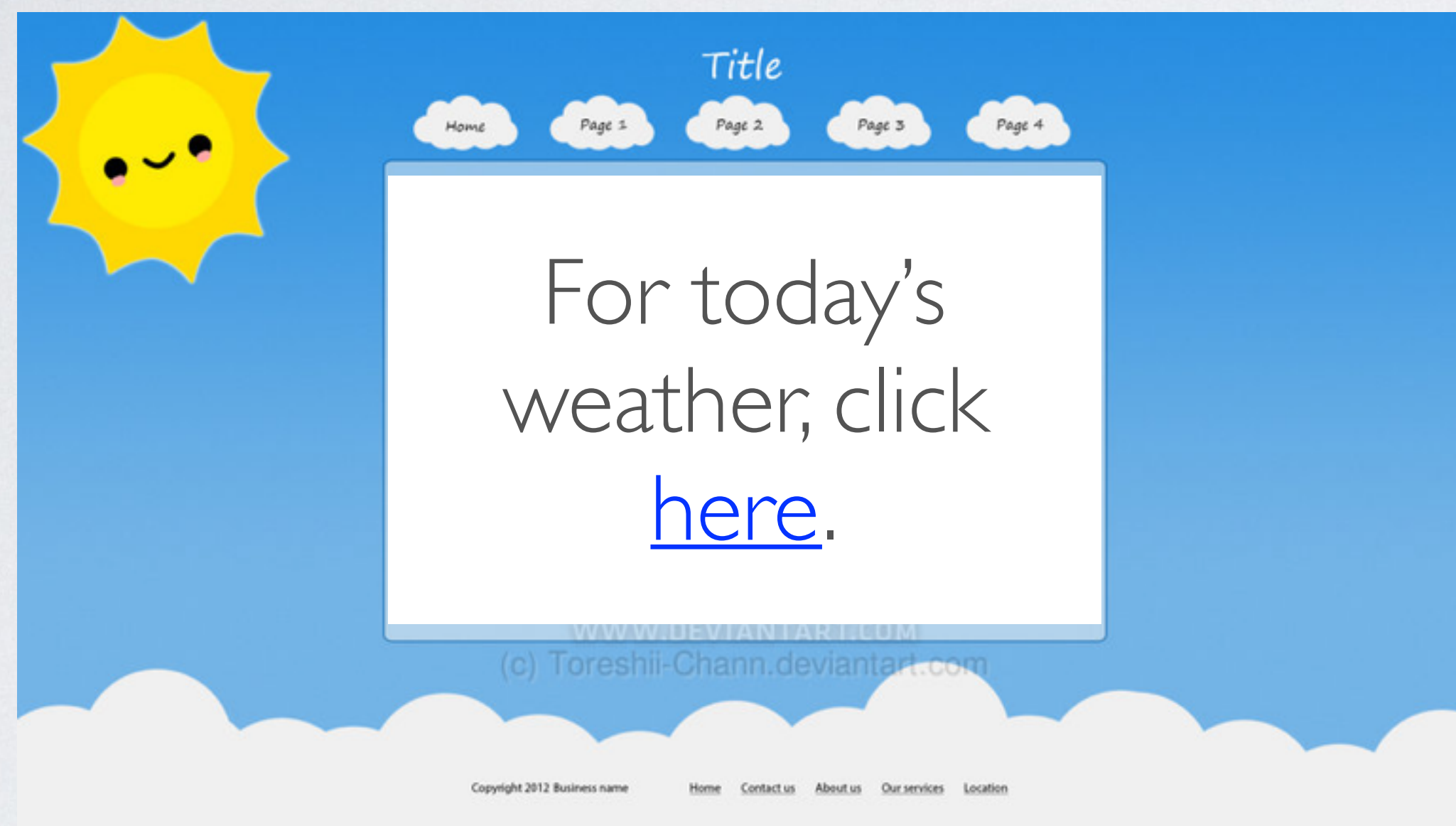
- Pay attention: there are *some* dependencies between the tasks (e.g. you need to have integrated an API before using the data provided by this API).
- When you plan the project, you do not have to do all the tasks. Some of them are redundant (alternatives). Some of them might be challenged/discarded (you need to explain why do decide not to do them).

“NICE” VS PLAIN VANILLA



For today's weather, click [here.](#)

LINK VS CONTENT



UP TO YOU



- Get together.
- Have a first look at the cards, ask questions if things are not clear. Discuss your perception about the **value** for the different cards.
- Try to find a way to organize them. How will you proceed to prepare the project plan?
- After 20', we get back together and let's discuss what strategy you have adopted. I will then share mine.

Phase 2

GOOD PRACTICES

- Deliver a simple system as quickly as possible and frequently deliver new value (functionality) to users.
- Value feedback from users.
- Minimize the work in progress items (lean).

A SIMPLE PLANNING TOOL

	What the team does (tasks)	Who are the users	What value do we deliver (done)?
Week 1			
Week 2			
...			

GUIDELINES

- Go back to your cards and organize them.
- Prepare a table, so that you know what are the milestones at which you deploy features (what and when).
- Write down the key decisions that you have made and the questions that you have discussed when doing the exercise.
- Let's get together at the end to do a wrap up.