Developing International Software*

August 27, 2015

Contents

I Module 1 **2**

1 Why Globalize Software **2**
1.1 Video Transcript . . . 2
1.2 Why Is It Important to Globalize Software? . . . 3

2 Flower App **8**
2.1 Video Transcript . . . 8
2.2 Apps That Work Internationally, Apps That Don't . . . 9

3 Marketing and Public Relation (PR) **10**
3.1 Video Transcript . . . 10
3.2 Marketing and PR . . . 11

4 Kicking Off Your Software Project **13**

5 What Could Possibly Go Wrong? **15**

6 Assignments **18**
6.1 Assignment Part One: . . . 18
6.2 Assignment Part Two: . . . 19

II Module 2 **20**

7 World Ready Process **20**
7.1 Video Transcript . . . 20
7.2 The Process of Creating World-ready Software . . . 21

8 Terminology **22**
8.1 Video Transcript . . . 22
8.2 Terminology . . . 24

9 Course Name Change **28**

10 Basic UX Design Considerations **28**
10.1 Video Transcript . . . 28
10.2 Basic UX Design Considerations . . . 30

*Last updated on 20 Agustus 2015 by jibi-edx

11 Basic Engineering Considerations **32**
11.1 Video Transcript 32
11.2 Basic Engineering Considerations 32

12 UX and Engineering Errors **33**

13 World-Ready Marketing **33**

Part I
Module 1

1 Why Globalize Software

1.1 Video Transcript

[Bjorn] Let's talk about **why it's importance to make software world ready**. Nadine, why don't you look at this from a business point of view?

[Nadine] I'm happy to do that, Bjorn. Any software project or indeed any product development project at all starts with an idea and, be successful that idea has to fulfill some kind of need in the market. Whatever idea you come up with, your natural human instinct is going to be to see it through your own lens and in the context of your own life. This isn't necessarily a bad thing, but go **do an internet search on products that flopped overseas**. You'll see some really big company names in your results. Actually, you should go ahead and share some of the best stories you find in the course wiki, or if you have your own experiences you should share them. Bjorn and I have plenty of great stories on this topic. The lesson is, **you can't force fit a product into other markets**. When you're considering a market you have to ask yourself some key questions. For example **does my software fulfill a need in this market? Will anybody use it?** Just because somebody finds your app really useful in one place doesn't mean people someplace else are going to find it useful. Or that they'll want to use your app over something that was built in their own country. There are many examples of this. Okay so, perhaps your software does fulfill a need and people will want to use it. But the next question is **would they pay to use it?** Okay, let's say that they pay to use it. Now the question becomes **will they pay enough to use it to make it worth my while to send it to them?** The amount people would be willing to pay in the United States could be an entire month's wages in another economy. All right, let's say people will pay enough for your app to make it worth your while. The next question is **can it become popular enough for lots of people to pay for it to make it really worth your while?**. **So if you answered no to any of these questions it doesn't mean you should abandon your idea**, right Bjorn? **You may need to adjust it**, or to use a term you'll hear a lot in this course, **you may need to localize it**. That is, you have to account for market differences, so Bjorn, why don't you take us through what some differences there are?

[Bjorn] Sure. I mean, the obvious one is that people speak **different languages**. Languages use **different words**, but they also use **different scripts** and have **different grammar rules**. For example, the plural, which is in English just adding an s, in most other languages have way more complicated rules. Some languages use spaces, other languages don't. Languages can be read left to right, right to left, or at times, top to bottom.

Next, there are differences in country and regional standards, which include things like **currency**, how they **format an address**, how they **format dates, numbers**, whether they use 12 or 24 hour clocks, what **measurement systems** they use, and **whether they start their week on a Sunday or on a Monday**. Culture differences will influence how people respond to your software. Someone here may say wow that's useful. Someone there why would I ever need that? This comes down to how people live. Not just their taste, but their day-to-day reality. Where and how do they typically **shop**? What kind of **money** do they like to use? Cash, phones, credit cards, debit cards? How do they **socialize** and **network** with each other? How does **climate or economic factors** affect the way they do things. Also you have to consider **cultural history and customs**. What **colors** mean, what **gestures** mean, what **holidays** are important, whether communications use more **words** or more **pictures**. That's all super important. The same word or phrase can have completely different meanings in different cultures, even if it's the same language. It may be perfectly acceptable in one culture and titillating or horrifying in another. And,

here's always **slang**. Slang doesn't translate well and neither do cultural references. What does the American expression, motherhood and apple pie, mean to someone in China or rural India? What's a touchdown? What's a home run? These all mean nothing outside of your cultural reference. There are **technological differences**. For example, levels of infrastructure. **Some developing markets don't have access to high speed internet but have access to cell towers**. So they use phones instead of PCs or they use phones because PCs are too expensive. **In some countries the phone is the internet**. Some areas have regular power outage which can affect servers or even data centers which in turn can affect whether people can access your application. Very important are **legal differences**. European Union is very strict about **consumer privacy**. And there are **political differences**, which can be very sensitive. Just ask any software company that has displayed the map. One country thinks it's fine, but another nearby country considers a national affront.

We will spend a lot of time in the rest of the course explaining language and formatting differences and what kind of legal and other geopolitical issues you need to look out for. So Nadine, how about you cover the next areas?

[Nadine] The key point is this, Bjorn. **If you want your software to be successful worldwide, you need to accommodate the differences we've discussed in your design and in your code**. This involves processes called **globalization or internationalization and localization**. We'll talk about these concepts throughout the course. Suffice to say **if your software can't accommodate these differences, people in some regions won't wanna use it or they won't be able to use it, and they certainly won't pay for it**. And if they're required to use it, for example, if you're at a company and this becomes a standard tool, and **it doesn't work for everyone, it can affect your bottom line**. Perhaps the product you're working on, it isn't about the software itself, but the software plays a supporting role. For example, in a device or part of a manufacturing process.

Think about it, if your software doesn't work internationally, if it doesn't work for multiple languages and multiple accounts it can even become a safety issue. Or perhaps you're not creating a product for profit or business use. Maybe it's for social good. The more globalized your software is, the more world ready it is, the more good it can do, right Bjorn?

[Bjorn] Yeah, I mean it all comes back to being world ready. **If your software is world ready, it can be used in more countries, in more markets, and you can make a bigger difference**.

1.2 Why Is It Important to Globalize Software?

Going into the global market unprepared can turn the release of an otherwise exciting new product into a hellish slog that will make you wish you never began the process.

What a Hellish Slog Feels Like

Let's say you're embarking on your first ever trip overseas—a whirlwind tour of five countries on three continents. You've been looking forward to this for years and you know it's going to be amazing.

As your journey begins, your social media posts are full of enthusiasm and pics of you:

- Checking in at the airport.
- Buying Duty Free items you don't really need at only slightly-less-expensive prices.
- Relaxing in the waiting area, getting ready to board.

You take off and select a movie from the chock-full on-board entertainment system, but it's in the language of the flight's destination country—no subtitles. The same is true of the second and third movies you select. You've seen all of the movies available in your own language, English, so you turn to the passenger next to you to strike up a conversation.

She only speaks a few words and phrases of your language, so you stumble through in your high school version of her language about the adventure ahead of you. At first she looks befuddled, then amused, then horrified. She summons the flight attendant, and after she makes a passionate appeal, you find yourself reseated in the back of the plane, near the lavatory and in between two screaming infants.

Utterly drained upon arrival, you get stuck in the slowest line at customs, where they single you out for a search. The official spreads all of your personal items across a table, confiscating several items that are not allowed into his country. And after an hour of questioning, he releases you.

At this point, all you want is a coffee. You order one in the airport, handing a large bill to the vendor who promptly hands it back, raising his hand in a gesture suggesting something unpleasant. "They switched the currency last year," the tourist standing in line behind you comments smugly. He reaches forward with the correct currency and exchanges it for the coffee that should have been yours, ensuring that it passes right under your nose as he takes his first sip.

After another long wait, this time at the currency exchange window, you get in a taxi, fumble through a mispronounced explanation of your destination in the local language, to which your driver responds in perfect English. You give a forced smile for a selfie that you post to social media along with a story about your "fantastic flight" and "smooth traversal" of the airport, unaware that the data your phone is chugging down in the background will contribute to a $500 bill at the end of your month of travel.

Once you arrive at your hotel, all you want is a shower. It takes several minutes to figure out the hot and cold knobs are reversed, and then ... heaven. The ancient, discount hotel (which you reserved online, of course) does not supply hairdryers, but fortunately, you brought your own, except you forgot the adapter. A desperate call to the front desk scores you one. So throwing on some clothes, in your dripping wet hair, you grumpily head down to fetch it, head back up to your room, plug in your hairdryer with a sigh of relief and...

Sparks shoot out of the wall socket as a noxious, burning smell fills the air. Then the power goes out and you're in the dark. You open the window and then the room door to get a cross breeze going, at which point you realize the power is not just out in your room, it's out the hallway. In fact, there's no power in the entire hotel.

The hotel manager barrels toward your room, screaming and gesticulating at you. You try to explain you used what they gave you, until he barks, like you're an fool, that the voltages are different—you needed a converter in addition to an adapter, which he would have loaned you had you told him you were using a hairdryer. It is now officially all your fault.

Your next "cheerful" social media post explains that you've decided you can better immerse yourself in the local culture by staying a youth hostel and hanging out with the locals.

The Lesson

The point of this sad story is that trying to use software outside the market it's designed for can be just as hellish an experience. It can be annoying, frustrating, infuriating, or even—in today's technology-dependent world—dangerous.

Building software is a process of designing the user experience and then writing code to deliver that experience. It's natural to design an experience that you would find useful in the context of your own life and to write code with your experience in mind. Doing so, however, could limit your success.

Market Difference

Market Need

Any successful product starts with an idea that fulfills some kind of need. Software is no exception.

A product idea may strike you when you encounter a challenge in your everyday life:

- "I wish I had a way to..."
- "There must be an easier way to..."
- "I need a less expensive, more powerful, or more reliable alternative..."

You may want to take advantage of the new technology:

- "This dream scenario is now possible."
- "This task will become much less complicated and expensive to complete."
- "It will be much faster/more entertaining/safer to do the things I want or need to do."

Whatever idea you come up with, it will be your natural instinct to see it through your own lens and in the context of your own life and surroundings. Your product may be a hit with people who have similar tastes, lifestyles, or experiences as you, but that doesn't mean it will be a hit with everyone else. Some companies have learned this

painfully and publicly. Go do an Internet search on "products that flopped overseas," and you'll see some really big names in your results. Perhaps you have directly encountered a product that didn't work for you or people you know. Go ahead and swap stories with your fellow students in the course wiki.

You simply can't force fit a product into other markets. When you're considering a market for your product, you first have to ask yourself the following basic questions:

- Would people in this market even want or need your software?
- Would anyone use it? Just because people find your app useful in one market, doesn't mean people in other markets will.
- Would people in this market be able to use your software even if they wanted to? Does their country have the right infrastructure? Do they have compatible devices? Would any regulations prohibit its use?
- Is there any local competition? Would this market use your software over something built in their own country?
- If your software does fill a need and people can and will want to use it, the next question is, would they pay to use it?
- If they seem likely to pay for it, you now have to ask, "Will they pay enough for it to make shipping my software to their market worth the effort required?" The amount people willing to pay in the United States could be an entire month's wages in another economy. Or perhaps the potential audience is much smaller in this market than it is in yours.
- This leads to the final question: "Can my software become popular enough for lots of people to pay for it and make it really worth my while?"

Perhaps you're not building your software to sell in the mass market. Perhaps you're building an app or Line-of-Business application used inside your company. In this case, there's another set of questions to ask :

- Are there company employees who do not speak the language most employees speak?
- Is this software critical for those employees to do their jobs? Will having software that works for their market increase job satisfaction?
- Does the software need to account for marketplace or legal differences? For example, software related to human resources might need to accommodate different hiring laws or benefits requirements.

If you answer no to any of these questions, that doesn't mean you should abandon your idea. You may need to adjust it—or, to use a term you'll hear a lot in this course—you may need to "localize" it.

Software Localization: Accommodating Market Differences

"Localizing" your software goes beyond translating the User Interface. In fact, it isn't always necessary to translate your UI; for example, many markets will accept an English UI (at least initially). But properly localized software should accommodate the local language, as well as regional and national standards, and even local customs.

So when you design and write your software, you need to be aware of the following market differences:

Language

The obvious market difference (and the one most people think of first) is that people speak different languages. There are numerous aspects to this. Languages in the same family use the same basic set of alphabetic characters. We call that written form a "script." For example, Western European languages use the Latin script, Slavic languages use the Cyrillic script, and many Middle Eastern languages use the Arabic script. Some languages use a variation of a basic script; they may drop some characters, add characters, or use diacritic marks that change how a character is pronounced. Japanese uses multiple scripts: hiragana, katakana, romaji and kanji.

People need a way to enter their language into your software, whether by typing, inking (drawing the characters), or voice. Once it receives input, your software needs to display text properly. Depending on the language, you need to layout the text left-to-right, right-to-left, or top-to-bottom. Some languages use spaces. Others don't.

Grammar rules vary by language as well. That may sound obvious, but many software programmers have fallen into the trap of assuming that a sentence (or "string" in software parlance) will have the same grammatical structure when translated. This is often not the case. In Japanese, the verb usually comes at the very end of the sentence, and in both Japanese and French, possessives have a different form than they do in English. It's not "Björn's house," it's "house of Björn."

And then of course, numerous languages have "gendered" nouns which affect declensions as well other grammar rules.

Country and regional standards

The same language can also be spoken or written differently depending on the country. This is where country and regional standards come into play. You can't just say, "Translate that into French." Which French? French as spoken in France, Canada, Belgium, Switzerland, or Senegal? If you say, "make sure you can type in French," you need to make sure you can enter text using the various keyboard layouts that are standard in different countries. There are French, Belgian French, Canadian French, and Swiss French keyboards. Written and voice input aren't straightforward either. Writing styles, pronunciation, and idiomatic expressions (aka slang) vary by country or culture, even for the same language.

Sorting order

Not only does alphabetical order vary among languages, but conventions for sequencing items in dictionaries and phone books can also be quite different. The rules for sorting and comparing text are language-specific, even within languages based on the same script.

In Swedish, for example, some vowels with an accent sort after "Z," whereas in other European countries an accented vowel comes right after the plain version of the same vowel. Asian languages have several different sort orders based on phonetics, radical order, number of pen strokes, and so on.

Cultural Difference

> "If I'm selling to you, I speak your language. If I'm buying, dann müssen Sie Deutsch sprechen (then you must speak German)" —Willy Brandt

Cultural differences influence how people respond to software, from whether they like its appearance to whether they find it useful in their day-to-day life. Culture is a complex amalgamation of belief systems, habits, tastes, social norms, artistic expression, regional history, tradition, and so forth. It is influenced by external factors such as geography, climate, economic development, and politics. Culture is an entire field of study in and of itself, but for our purposes, we just need to know that differences can be substantial depending on where you are in the world, and making assumptions based on the culture you come from can get you into trouble.

Here's a partial list of things you should think about that can have a strong impact on how successful your software or software-driven product will be:

Religious beliefs An app that gives detailed information on types of beer might be huge in Germany, but it would be considered offensive in many Muslim countries.

Population density The United States is a very large country with wide open spaces and large houses that can accommodate lots of stuff, whereas Japan is a country the size of Montana where real estate is at a premium and many people live in very small apartments, making technology with a small footprint attractive.

Holidays and other seasons that drive purchases Many countries celebrate Valentine's Day, Mother's Day, Father's Day, and graduations. The United States has three major buying seasons that impact the purchase of technology: December holidays, Back-to-Business, and Back-to School. Asian countries celebrate Chinese New Year and Golden Week.

Standard of living The average salary in Switzerland is approximately $53,000 USD. If you create a subscription service that costs $10 USD a month, you're asking the average Bangladeshi to give you 10% of their annual income.

Banking systems Some cultures make heavy use of credit and debit cards. Developing countries tend to rely on cash, and, increasingly, mobile phones.

Literacy rates If people in your target market can't read, you will need to use pictures or icons instead of words wherever possible. There are many other examples. If you think of any, please share them with your colleagues using the course wiki.

One important trap you must watch out for is the phenomenon of "same word, different meaning" or "same meaning, different word." If you're not careful, you can confuse or even offend your customers.

Pictorial representations

Some cultures, such as Asian cultures, use pictures in their communications more than others. Pictures or icons can be one way to reduce the amount of text you have to translate. They are also useful for communicating to people in regions with low literacy rates, such as Africa. But you need to be careful when you use pictures.

A good example is pictures or drawings of gestures. They may be positive or benign in some cultures, but supremely insulting in others. Here are a couple of examples (note that we're describing them vs. showing them—our students are from all over the world!)

Maps

Showing the "wrong" map can embroil you in controversy. Just ask any software company that has displayed a map one country thinks is fine, but another nearby country considers a national affront.

For example, you have several choices when showing a map of Israel:

- The one with Judea and Samaria.
- The one with the Palestinian Authority.
- The one without the occupied territories.

China considers Taiwan and Tibet part of China. India has territory disputes with China, Pakistan, and Bangladesh.

Your best bet is to show the map that is expected by the target audience of your localized app. Or better yet, avoid showing maps at all unless they're directly relevant to your product.

Colors

In the United States, white flowers are appropriate for weddings. In China, they are appropriate for funerals; the Chinese adorn weddings with red flowers.

Images

Symbolic use of photos of people, animals, things, or places may be lost on some of your audience. Some people may instantly recognize certain celebrities, political leaders, musicians or athletes, but others may have never heard of them. This gap may be generational as well. Films and TV shows, even those with a worldwide audience, may be banned in some countries.

Be particularly careful of representations of women or animals, particularly as mascots, characters, or in advertising. Manga-like drawings of young girls in short skirts may be common in Asia, but considered pedophilic in the West, or downright taboo in the Middle East. People in India consider cows and monkeys sacred.

You also have to be careful about showing images that some cultures may consider racist. When Apple introduced emojis with different skin tones, they were trying to be inclusive. Asian Americans, however, who associate the color yellow with racial epithets, found the new yellow colored emojis shocking and insensitive.

Written or Verbal Expressions

The same word or phrase can have completely different meanings in different cultures, even if it's in the same language. It may be perfectly acceptable in one culture and not appropriate for polite company in another. This is particularly true of slang because it's culture-specific and doesn't translate well. And neither do cultural references. What does the American expression "motherhood and apple pie" mean to someone in China or rural India?

Technological Differences

Many technological differences stem from infrastructure, which may affect which markets are appropriate for your software, as well as how your software operates. Some developing markets don't have access to high speed internet, but do have access to cell towers. In these markets, many people use phones instead of PCs. In some countries, "The Phone is the Internet." Some countries, such as South Korea, have advanced infrastructure that can support technology experiences not yet accessible to other countries. They are rapidly rolling out affordable one-gigabit per second connections which are roughly 263 times faster than the world average and 100 times faster than the average speed in the United States. Other countries have unreliable infrastructures and even regular power outages.

Technological standards also vary, for example, voltage, AC frequency (50 Hz vs. 60 Hz), and DVD region code. Phones commonly used in one country may not work in other countries, as cell technology may use different standards.

Legal Differences

Shipping software that violates local laws can result in sales of your software being blocked and your company being sued. Adhering to legal standards is called "compliance." The European Union is very strict about consumer privacy, for example. Any software that deals with money and "Personally Identifiable Information" (PII) must adhere to strict security standards, often government mandated. In some European countries, this includes a requirement to store all data, even for games, inside in that country's physical boundaries.

If your software involves content, such as video or music, you may have to negotiate deals with content providers for each country, as licensing and distribution laws vary. You will also have to calculate and pay taxes differently on anything you sell.

As you build connections and form partnerships, you need to understand the difference between cultural expectations, such as giving gifts, and violating anti-bribery and corruption laws. It's always a good idea to have knowledgeable legal counsel.

There are pitfalls in more directions that you can easily keep in your head. So it behooves you to plan ahead for each and every market you hope to target, consulting with people who know each culture well.

2 Flower App

2.1 Video Transcript

[Bjorn] We promise that we give you a few **examples** of either applications that **didn't go quite so well**, or of **potential downfalls** that we could see. So this is a potential one as far as we know. But it could definitely happen.

[Daniel] Yeah.

[Bjorn] So, let's assume there is, well I come up with this great idea, that I'm creating a flower service that runs in Seattle. And, basically, what it does it automatically sends through subscription the appropriate flower to the customer that I pre-selected. So it would send like red roses on Valentine's, buy lilies for Easter, and like a nice flower, spring flower bouquet for Mother's Day and let's say some poinsettias for Christmas, and whatever is in between. And I could actually see that really this takes off, because it would be an easy way, it would be like I sign up, I pay once and it takes off, right? And takes off like wildfire. I make a lot of money here in the Seattle area. I go to San Francisco and L.A., and then on the east coast, and everywhere it works brilliantly. And then, a friend in Japan actually talks to me and says hey, how about we expand this to Tokyo?

[Daniel] Hey, Bjorn, I'm wondering about Easter. Is Easter as an holiday which is celebrated in Japan? And I'm wondering, the in the white color has different meaning actually in Japan, it's also symbolize death. And also,

I'm wondering about the date format, which is completely different in Japan than that what we're using here in the US and, clearly I'm wondering about the local vendor. Are you shipping the flower from California to Japan, or are you using a local vendor for the flowers? So please explain me.

[Bjorn] Oh, my! You know this is stuff I obviously haven't considered. Now I have sent hundreds of thousands of people in Tokyo Easter flowers, a holiday that almost nobody celebrates. And worst of all, I've sent them actually from the US, so they arrive like three weeks later and they were completely done by then. And, you're right, I didn't consider the date format because what I put in as 6, 12 did not ship on June 12th, but it shipped on December 6, that's Nicholas Day. And then, a local vendor, that would have been smart, man.

[Daniel] Yeah.

[Bjorn] Why didn't I think about this? Why didnt I take this class?

[Daniel] [LAUGH]

[Bjorn] Okay, obviously, this was a made up scenario and we're role playing a bit here. But you get the idea. **You need to watch out for these differences between markets**. Otherwise, you will end up with a scenario like this, and you might lose all the money that you made in your first rollout with the next **rollout** by going **into an international market**.

2.2 Apps That Work Internationally, Apps That Don't

Properly globalized software has a much better chance of being successful in markets outside your own. But as we like to say in the software business, if your feature works the way it's supposed to, no one will notice. If it breaks, everyone will.

Here's a **collection of true stories** from your instructors' collective experiences that illustrate what can go wrong, even when you're careful.

The Flowering of a Controversy

When Microsoft first released Windows, it included games like Reversi and Solitaire that would help people accustomed to working in MS-DOS learn how to use a graphical user interface. And when Microsoft introduced Minesweeper with Windows 3.1, it became a mindless distraction on an international scale. It was a silly, unsophisticated little game that Microsoft never imaged would cause any kind of controversy, until it did.

An American software engineer based in Redmond developed Minesweeper in his spare time. As his colleagues tried it out, it proved addictive and caught on. One could argue whether it was a good idea to ship such a productivity killer with Windows, but a little fun never hurt a product, right? This made perfect sense to people born and raised in North America, who may have heard or read about mines, but surely have never dealt with one in real life.

In some countries, however, landmines are a real problem, killing and maiming people, including many children. To people affected by mines, the game was appalling. Taking this feedback to heart, Microsoft offered an alternative with Windows Vista, called Flower Garden, a similar game with flowers instead of mines. Lawyers and geopolitical experts helped Microsoft come up with a list of countries where landmines were a sensitive issue and then, if you installed Windows Vista in one of these locations, you got the more country-appropriate game. Problem solved, right?

Except that it wasn't. As it turns out, even if users received Flower Garden by default, they would still see the name Minesweeper with an icon that looked like a large, prickly mine. Plus, if they looked at Flower Garden's help file, it still referred to mines, not flowers. (Note: In the second course in this series, we'll teach you how to change icons and file names based on what's called a LocaleID.)

So Microsoft addressed those issues too. End of story, right? Nope. Some groups didn't want either game on their computers, including businesses who wanted their productivity back. Today, IT administrators can control what apps get installed on PCs in their organization.

Pictures, or It Didn't Happen

Before human factors specialists and usability testing made software design more people-centric, technical people designed software. In fact, Microsoft called us "Software Design Engineers." This resulted in a lot of software that was "too technical" for the average person. When the Windows team embarked on Windows Vista, the leadership

prioritized the human experience. The team designed a number of "end-to-end experiences" that took people step-by-step through completing a task from beginning to end.

One of these experiences involved photos. When you plugged your camera or camera card into your PC, Windows automatically detected it, asked if you wanted to download your photos, and then helped you correct things like color, brightness, redeye, and cropping. As a final step, you could share your photos online, or you could print them. Back then, since few people had photo-quality color printers, the Windows business development team decided to broker deals with photo printing providers so consumers would be able to send their photos directly from Windows Photo Gallery, and the providers would make high quality prints and mail them back.

But as the team started making these deals with US providers, they realized people in other countries would never order their photo prints from a US provider. To make this feature real, they would have to make deals in every single country where Windows shipped. This was clearly unworkable, and they abandoned the idea.

Encouraging Efficiency, or Spying?

For businesses that charge by the hour, such as consulting or law, the ability to track how much time it takes to complete a document could be very useful, or so the Microsoft Office team thought. They created a feature that allowed users (or their managers) to track how long an individual spends typing in a document. Some individuals and businesses found this feature useful and convenient. Those who didn't need it simply ignored it.

In Germany, however, the feature caused an uproar. Workers felt Microsoft was helping management spy on them! Germany is heavily unionized, and the unions vehemently objected to this perceived conspiracy. It was particularly egregious, they felt, because the functionality could be hidden, meaning managers could track the time spent in a document without the worker's knowledge.

Choose Your Words Carefully

It is impossible to anticipate every small detail that could potentially trip you up and cause problems for your customers. But it is good to be as prepared and as thorough as possible, because even a single word or character can make your company a target. The following is a list of real incidents from Microsoft's experience:

- The time someone typed a German name into an English-language version of Microsoft Word and it auto-corrected to a very bad word.
- The time the Spanish version of Windows used the word Hembra, which means "woman" but "bitch" in Nicaragua.
- The time Microsoft Outlook acknowledged April 30 as the queen's birthday in Uruguay, a proud republic, offending its government.
- The time Windows 95's time zone map of the world did not display the disputed region as part of a certain country, which then banned the operating system.
- The time Microsoft had to withdraw the game Age of Empires 2 from Saudi Arabia because it showed Muslim armies converting churches into mosques, offending Saudi authorities.
- The time Microsoft accidentally reversed the Korean flag, upsetting the Korean government.

And that's just the tip of the iceberg. If you know of other great anecdotes about software products that inadvertently courted controversy because of cross-cultural snafus, please add them to the course Wiki.

3 Marketing and Public Relation (PR)

3.1 Video Transcript

[Nadine] Early in my Microsoft career I took some time off to get my MBA at Stanford. And I remember coming back, during the summer, to a meeting where there was a senior technical executive. And he said to me, oh, you're getting your MBA? Don't worry. We won't hold that against you.

So I'd like to take a moment to talk about marketing and PR. Yeah, this is a computer science course. You can say Nadine, I'm a technical program manager, I'm a coder, I'm not a marketing and PR person. Why are you bothering me with this? Well, this is a computer science course and a big theme of this course is planning ahead. And this includes designing and writing the kind of software that your marketing people can convince people to buy. Plus, your PR people will be the ones who put you in front of the press, analysts, your partners and audiences who are interested in your software. And you have to say positive, compelling things that make people believe in what you're doing. So marketing and PR is important to you as a technical person. And what does this have to do with internationalization? Well, everything. And let me explain why.

I'm gonna give you a scenario, based on something that actually happened. Now let's say you work for a gaming company and you're developing this awesome hand to hand combat game and it's in the tradition of great kung fu movies. Now you're making it for a partner. A very big and important partner. Which is a great opportunity for your fledgling company. Well this is an epic game, so the soundtrack of course has to be just epic. A friend plays you an awesome sound of people chanting in unison. Wow. It's rhythmic, it's melodious, it's harmonious. It's a perfect fit for this game's feel and the background of it just makes everything come so alive.

So you ask the team's composer to add this sound to the sound effects, to the soundtrack of the game. But you want to get a couple more opinions, so you take it by a few more of the people around the office, and one of your colleagues is a Muslim and he speaks Arabic and he says, this chant, it's from the Koran. And he starts getting really upset. He says, how can you do this? How can you include something so meaningful to Muslims in the soundtrack of a computer game? That's really offensive.

Well, you're faced with the situation, what would you do? Well on the one hand, you could immediately drop the idea of using this sound and find something else. And on the other, you could tell your colleague, hey, stop being so politically correct. And we're gonna use this sound because it's just awesome. Of course, that choice seems kind of obvious, doesn't it?

Well, let me tell you what the real company and the real scenario in question did with this situation. They went ahead and they shipped the soundtrack with the chant, but only in the United States, because they said nobody will notice, and that seemed to work fine, until the Saudi government called. That's right, the government of Saudi Arabia, they did notice, and they weren't very happy. So the company ended up pulling the game worldwide, and someone high up in the company had to go in front of the press, with the help of the PR people and explain this. And they had to apologize to everyone who was offended in order to save the company's reputation.

So let's check back a little bit. How did this happen? In the real story the game in question was made by a company in Japan. Now if you're familiar with Japan, you know that millions of people in Japan chant. And they're mostly Buddhists who recite sutras that derive from ancient languages. So the game developer who found it, he thought it was really cool, but he didn't go check to see where it came from. This particular chant. He didn't know it was from the Koran, how would he? He doesn't speak Arabic. For all he knew was Sanskrit like the Buddhist chants in Japan, and since he liked it, he said hey let's use it. Well the team developing the game, not the marketing or the peer people, made this decision to include this sound. And the game became something the marketing team could never sell, and the PR team had to step in and clean things up.

So the **moral of the story is, decisions that you make as product developers have a tremendous impact, way beyond writing code. So raise your awareness, and plan ahead**.

3.2 Marketing and PR

Why is there a unit on Marketing and PR in a computer science course? We've included this discussion because how you design and write your software for international markets can have consequences (sometimes very big consequences) for marketing and PR.

For many years, marketing efforts started toward the end of the software development cycle. Before agile methodology gained favor, the "**Waterfall**" approach to developing software was dominant. **Software teams designed software, developed it, tested it, and shipped it**. As the release-to-market (RTM) date approached and the software stabilized, **marketing teams constructed marketing campaigns and PR teams got ready to generate buzz for the product's launch**. Once the product launched, **sales teams got busy** and the **software team started working on the next version of the software, beginning once more at the design phase**. The next version might ship a year or more later. The **marketing, PR, and sales teams continued to work on the product just released, until the next release** drew near and they had to shift their focus.

In today's age of rapid, agile development, and continuous updates for cloud-based software, timelines are

compressed. Marketing and PR also have to be rapid, agile, and continuous. If you don't want the teams tasked with convincing people to buy your software running around like crazy people with their shorts on fire, you need to put yourself in their shoes on a consistent and constant basis.

There's an American saying, "**Never believe your own press**," advice apparently given to movie stars because studios paid reporters to write nice things about them. "**Don't think too much of yourself**," it implies. When you promote software, you have to **find the right balance between telling the world how great it is and not appearing arrogant**. Shipping software that breaks when people outside your home market try to use it, or that includes features or graphics insensitive to some cultures, is a sure way to earn the "arrogant" label. What's more arrogant than only caring about people who do things and view things the way you do? Read up on the missteps Microsoft made during its earliest forays into the Chinese market in the 1980's for a lesson on how cultural misunderstanding can lead to business friction.

These days, once an ugly incident, embarrassing quote, or bad review gets on the Internet, it pretty much never goes away. It doesn't matter if the reports are taken out of context or you plead ignorance. If you do, you just get labeled "out of touch" or "ignorant." So if your software is unimpressive to someone—or worse yet, offensive—be prepared for the possibility that they will tell everyone they can, and that those people will tell all their friends. **You will then have to explain yourself, fix the problem, and hope you can recover**.

Here's a great exercise for putting on your marketing and PR shoes: as part of your planning process, **write your own press release, highlighting your software's strongest selling points**. Below is a form to get you started. You can replace the generic words in bold with words that match your project.

1. Today Team Name announced the availability of Software Name, a type of software that gives the target customer a powerful, inexpensive way to accomplish a task.

2. "We are thrilled to release this software to market," said Your Name, your title, your company or team name. "We believe it addresses a gap in the market that will lead to big impact."

3. Team Name has beta tested the product with a number of customers in specific international markets.

4. "Initial customer feedback has been outstanding," said Your Name. Customers have told us positive things about their experiences using the software. Customers living in certain countries find Software Name particularly useful, because it addresses a need specific to their market that no one else has been able to address as well before now."

5. Software Name includes a number of features that help users do a list of awesome things that will make everyone want to use the software.

6. Software Name is available today in list of countries, in list of languages language editions via where people can get it. It sells for price.

The above is a very simple example of a product announcement. The key points are to identify what your software does, who it is for, and what they will be able to accomplish using it. Now think about the list of markets you put into your press release:

- Does the gap that your software addresses exist in all of your target markets?

- Will your approach to addressing that gap be understandable and appealing to people in all of those markets, who come from different cultures and have different tastes?

- Do people living in your target markets have access to the technology necessary to run your software? Have you thought carefully about what constitutes a "great experience?" The experience needs to remain as great for someone who lives in one of your target markets and tries to interact with the experience in their own language.

- Are you certain that people in your target market have access to the distribution channel you have selected?

- Are you certain that the price you wish to charge would be considered reasonable in all of your target markets?

As a further exercise, write your own product review, which can serve as "an inspiration to live up to" for design, development, quality assurance, and marketing. As you describe your cool and awesome features, ask yourself first if it will require additional effort to implement the feature in markets outside your own, and whether doing so is even feasible. You also need to ask whether people outside your home market will find the features just as cool and awesome as you do—and for the same reasons. Other cultures may find some features you think are great not only useless, but downright annoying or offensive. The real-life examples in the Flower App unit illustrate this possibility.

The Press Release no one wants to write, and the interview you never want to do, is the one in which you have to accept responsibility for a problem you could have prevented through appropriate awareness, design, and execution. It's not just about alienating customers. Imagine having to apologize to an entire country—or an entire religion—for something your software does or does not do properly.

4 Kicking Off Your Software Project

It's time to start planning the software project you will design and build as part of this course and the next two. The first step is to come up with an idea for what your software will do. Your idea will invariably evolve into a better one as the course progresses and you learn additional concepts, so give it room to grow. Some of you may already know what you want to build. The rest of you may be thinking, "How am I going to come up with a workable idea out of thin air?"

To make this a little less daunting, we'll give you some basic guidelines:

1. We suggest writing an app from scratch. Some of you may be taking the course so you can learn how to make existing software world-ready, but if it has a lot of code, this approach will be quite challenging. Better to learn by writing something new and applying what you learn to your other project later.

2. We suggest writing something simple that includes these basic features:

- It requires inputting text, not just clicking on or touching areas of the device screen.
- It displays the text the user inputs.
- It allows the user to do at least rudimentary text editing.
- It uses UI elements that translation will affect, such as menus, dialog boxes, or buttons. It makes adjustments based on country or language, like the formatting of numbers, dates or currencies. Refer back to Module One Unit Two for a list of market differences your feature set may need to accommodate.

3. If you want to write an app but aren't sure what kind, check out the categories in your favorite app store. Here are the categories in the Windows Store (Table 1) :

Books & Reference	Government & Politics	Music	Security
Business	Health & Fitness	Navigation & Maps	Shopping
Developer Tools	Kids & Family	News & Weather	Social
Education	Lifestyle	Personal Finance	Sports
Entertainment	Medical	Photo & Video	Travel
Food & Dining	Multimedia Design	Productivity	Utilities & Tools

Table 1. Categories in the Windows Store.

4. We want you to have fun, but to also keep in mind what may be offensive to someone else. At the same time, it's a free market, so we won't put restrictions on what you write. As part of Module Two of this course, you will need prepare a "PR Plan" to manage or appease anyone who might not appreciate what your software does, for whatever reason. The peers grading your homework will provide a rehearsal in dealing with the market.

5. Lastly, some of you may come up with an idea so awesome you want to write the software and actually sell it. We're not lawyers, but keep in mind that edX is a more or less public forum and does not offer any Intellectual Property protection. If you're concerned someone may steal your awesome idea, don't share it here.

Our App Idea: Flowers As A Service

Here's our idea: a shopping app that delivers flowers. It's a very simple example, designed to illustrate the concepts of this course.

Target markets

We are starting our business in Seattle, Washington in the United States. If we are successful, we will expand to San Francisco, Los Angeles, New York City, and beyond.

Type of software

AutoFlowr is an annual subscription service, managed via a mobile phone app, that automatically sends flowers to special people on holidays throughout the year:

- Red roses for Valentine's Day.
- White lilies for Easter.
- Spring bouquet for Mother's Day.
- Poinsettias for Christmas.

Target audience

Our app will appeal to all kinds of people:

- Thoughtful, kind consumers who enjoy spreading happiness to the people around them.
- Businesses who want to strengthen relationships with important clients.

Target platform(s)

- iPhone / iPad
- Android phones
- Windows phones

The customer experience

Getting started requires only a few steps.

- Customers download our mobile phone app and create an account.
- They create a list of recipients and the addresses where we will deliver flowers.
- They give us their credit card or PayPal information for billing purposes.

At this point, customers can let the service take care of the rest. If they want to personalize the flower deliveries, however, they can do the following via the mobile app:

- Add or subtract holidays from the delivery schedule.

- Add delivery dates: birthdays, anniversaries, annual remembrances.
- Add one-time deliveries to say "Thank You," "I messed up," "Congratulations," "I'm sorry for your loss," "Get well soon," or "I like you."
- Select which flowers get delivered to each person on each occasion.
- Choose from a variety of default messages to include on a card that gets sent with the flowers or write their own messages.

Technical

For our first release, we will keep the app relatively simple. It will:

- Use social media sign-in, supporting Facebook, Google+, Twitter, or Microsoft Account.
- Allow customers to enter their recipient list by hand, select individuals from their phone's contact list, or select contacts from their social media accounts.
- Download addresses for the recipient list from contact information if customers connect their AutoFlowr to their phone's contact list or a social media account. If a specific contact does not include an address, the app will ask for one.
- Select florists to deliver the flowers based on proximity to the delivery address. The app will put pushpins on a map so customers can look up the florists we assign.
- Give customers the option to add reminders for who is getting which flowers on which occasion to their calendar.
- Store all customer information in the cloud using Amazon Web Services or Microsoft Azure.

Distribution and marketing

We will distribute the app via the online app stores for each platform we're targeting.

Our two main marketing vehicles are as follows:

- Social media, e.g., a Facebook page and Twitter feeds.
- Partnerships with florists. We will ask them to promote our app in their stores and on their websites.

5 What Could Possibly Go Wrong?

If you watched Bjorn pitch our app in the Flower App unit, you know that our flower app proposal has some shortcomings. Did you notice any potential problems in the written description that the video did not cover? In this unit, we'll explore how we weren't thinking.

Let's say we launch our app in Seattle, where it is so successful that we're also able to launch our service in San Francisco, Los Angeles, and New York City. Now we want to go international. So we send our write-up to a Venture Capitalist friend who specializes in global startups in hopes that she will fund our expansion to Asia and Europe. She sends our proposal back with comments.

> Dear Bjorn,
>
> Congratulations on the success of AutoFlowr in the United States. I've reviewed your proposal and tested your app. While I see the potential, I'm not sure this is something I can get behind. Comments are below:

Target markets

We are starting our business in Seattle, Washington in the United States. If we are successful, we will expand to San Francisco, Los Angeles, New York City, and beyond.

> I get your logic. You wanted to see if your service would take off in an area you know before expanding. Unfortunately, because you chose that strategy, you now have some serious challenges ahead if you want to take your service into other countries. You'll have to make a lot of changes to your business model and app if you want to be successful in Japan, Israel, or Germany just for starters.

Type of software

AutoFlowr is an annual subscription service, managed via a mobile phone app that automatically sends flowers to special people on holidays throughout the year:

- Red roses for Valentine's Day
- White lilies for Easter
- Spring bouquet for Mother's Day
- Poinsettias for Christmas

> Here's the first issue. "AutoFlowr" is a name that clearly summarizes what the service does—in English. Based on your success, it seems to resonate in the United States. My guess is that it won't resonate in other countries, however. Let's start with Japan. Although the Japanese often incorporate English words when there is no Japanese equivalent, they do have a word for "flower," which is "hana." "Auto" pronounced by a Japanese person is "oto," which means "noise" or "man." "Flower" pronounced by a Japanese person is "fu-ra-wa" or "fu-ro-wa". "Furo" in Japanese means bath. So your app title could potentially be heard as "noisy bath" or "male bath," neither of which conjure up images of holiday flowers. Do you see where I'm going with this?
>
> Speaking of holidays, while Valentine's day is huge in Japan, fewer than 1% of Japanese identify as Christians. So Easter isn't a big flower-giving day there. Not only that, sending someone white flowers could be shocking, as white signifies death. It's better to send white flowers to funerals.

Target audience

Our app will appeal to all kinds of people:

- Thoughtful, kind consumers who enjoy spreading happiness to the people around them
- Businesses who want to strengthen relationships with important clients

> I see that you're targeting your audience by "psychographics" instead of "demographics." That's an interesting strategy, as there are thoughtful people all over the world. However, when people in some cultures want to send a thoughtful gift, they don't necessarily think of flowers first. The Japanese consider fruit, which is expensive in Japan, an impressive gift. The French might would feel the same about pastries or high-end chocolates. In France, patisseries outnumber florists. Have you considered expanding your offering beyond flowers?

Target platform(s)

- iPhone / iPad
- Android phones
- Windows phones

> Going cross-platform is a great idea. Just make sure you are familiar with what support each platform offers for globalized apps. They're not necessarily consistent. I hope you have some good coders who know how to manage this.

The customer experience

- Getting started requires only a few steps:
- Customers download our mobile phone app and create an account.
- They create a list of recipients and the addresses where we will deliver flowers.
- They give us their credit card or PayPal information for billing purposes.

  The first red flag I see here is that address formats vary by country. Your app, as far as I can tell, can only accept American addresses. You'll need to update that code.

  Also, whereas a lot of people in the United States use credit cards or PayPal, people in other countries primarily use debit cards or mobile phones to make purchases. Some countries still rely on cash, just something to keep in mind.

At this point, customers can let the service take care of the rest. If they want to personalize the flower deliveries, however, they can do the following via the mobile app:

- Add or subtract holidays from the delivery schedule.
- Add delivery dates: birthdays, anniversaries, annual remembrances.
- Add one-time deliveries to say "Thank You," "I messed up," "Congratulations," "I'm sorry for your loss," "Get well soon," or "I like you."
- Select which flowers get delivered to each person on each occasion.
- Choose from a variety of default messages to include on a card that is sent with the flowers or write their own messages.

  Personalization is always good. Customers like options. But you will need to change your default list for different countries as holidays vary widely. Japan celebrates Respect for the Aged Day, and a number of countries still have monarchs, whose birthdays are national holidays. Asia celebrates Chinese New Year and Golden Week. You should also think about which religious holidays to show in which countries. It probably won't make much sense to have Easter, Christmas, and Palm Sunday on the default list for Israel. Speaking of which, Jewish holidays begin at sunset, so you'll have to make it clear deliveries will occur the day BEFORE the celebration date listed in many calendars.

  As for default messages, don't just translate the ones you came up with for the United States. Have someone who was born and raised in your target market write new ones for you to make sure they're appropriate.

  Moreover, I'm concerned using default messages goes against the purpose of your app. The app is supposed to give the impression the giver is thoughtful, when in fact, using default messages tells the recipient the app was more thoughtful than the flower senders. In some cultures, finding out you are on an automated list it could be hurtful and damage a relationship rather than strengthen it.

Technical

For our first release, we will keep the app relatively simple. It will:

- Use social media sign-in, supporting Facebook, Google+, Twitter, or Microsoft Account.
- Allow customers to enter their recipient list by hand, select individuals from their phone's contact list, or select contacts from their social media accounts.
- Download addresses for the recipient list from contact information if customers connect their AutoFlowr to their phone's contact list or a social media account. If a specific contact does not include an address, the app will ask for one.

- Select florists to deliver the flowers based on proximity to the delivery address. The app will put pushpins on a map so customers can look up the florists we assign.
- Give customers the option to add reminders for who is getting which flowers on which occasion to their calendar.
- Store all customer information in the cloud using Amazon Web Services or Microsoft Azure.

A couple of items from your technical list concern me:

Have you looked into how flower distribution currently works in the countries you're planning to target? Do you even know if local florists would be willing to work with you? Are you sure they have the technology to receive your orders, or get electronic payments?

If you're going to use the cloud, keep in mind that some countries, like Germany, insist that any sensitive information on its citizens be physically stored within the boundaries of their country. There are a lot of legal considerations. Have you thought about tax laws?

Distribution and marketing

We will distribute the app via the online app stores for each platform we're targeting.

Our two main marketing vehicles are as follows:

- Social media, e.g., a Facebook page and Twitter feeds.
- Partnerships with florists. We will ask them to promote our app in their stores and on their websites.

Online distribution is great. It's relatively straightforward. It's also easy to do a search and discover that you already have a number of competitors in local markets. Perhaps customers would rather sign-up with a local company. You should do more research on this.

As for social media, don't forget that some of the largest social media sites in the world don't even exist in the United States, for example, China's Qzone and Russia's vkontakte.

Working with florists is an interesting idea, but you will have to carefully understand their customers to forge strong partnerships. Your app is best for people who don't to go to flower shops in person. The personalized attention people get at a florist is a radically different experience than setting up an automated list, and it's probably different country-by-country. The people who take the time to go to a florist may not be the same people who want to use your App. Moreover, there's nothing to stop a local florist from offering a similar program to people who visit their shop. It just takes a one-time visit to set everything up, particularly if the list of recipients is short, and people can actually see the flower choices.

Lastly, how can you distinguish yourself in the market from the stalwarts who are your competition? How else can you get word out about it so it takes off in the Zeitgeist like Uber or Groupon? Like AutoFlowr, both of those apps deal in IRL purchases from your app. You're an update of flower delivery services that have existed for at least a century. Uber allowed you to easily and quickly summon a car without having to hail one on the street. Groupon provided a searchable interface that was far superior to sorting through coupons. So, other than automating things so you can forget about the people you want to know you care about so deeply, what else do you bring to the table that will make you a household name?

6 Assignments

6.1 Assignment Part One:

- Write a one- to two-page summary description of the software you'd like to build. If you wish, you can use the summary description of the Flower App from Unit Eight as a framework, or the press release form in Unit Six.

- There is no need to write a full business proposal, product plan, or technical specification. Subsequent units will take you through those exercises. Our description is approximately 500 words long. Your maximum is 750 words.
- Publish the description in the discussion section.

6.2 Assignment Part Two:

- Review the summary descriptions submitted by two of your course-mates. Point out at least one issue their software may encounter in markets outside theirs. Please be specific and give examples.

Part II
Module 2

7 World Ready Process

7.1 Video Transcript

[Bjorn] Yeah, let's talk about a better way of making your software world ready. First of all get a commitment up front from everyone on the team, which might be only you, to do their part to globalize the software. That is, **ensure that everything, features, code, documentation, is designed from the ground up to work around the world**.

[Daniel] Yeah. I would like to add to this. Everyone in the team, I mean, if it's a one team, if you are the only person in the team, that's okay. But many times **you have other people on the team**. Can be **designer**, **architect**, **software engineer**, **writers**, **finance**, whatever, everyone needs to understand that this product can be globalized. And be sure that everyone is aligned with you in globalizing the product itself.

Make sure your **design in your code are locale aware**, that they account for **differences in language rules, country or region, and culture**. Locale-based information includes **keyboard layout**, and formatting for **date**, **time**, **numbers**, **calendars**, and **currency**.

[Bjorn] Yes, Daniel, locale awareness is something that gets often ignored in software. For example in Switzerland you're not only using a different keyboard layout to enter **German** language, but you also have different spelling rules. Like the very common word for street, [FOREIGN] in Germany is spelled with a extended character, the sharp s, and in **Switzerland** it's spelled with two s's. And then language rules in general, like, even if you looks outside of, like, differences between one language to the other, there are like rules somehow to create plural forms of words. In **English** you just add an S to the end, that doesn't work in any of the other languages.

So in addition to **being locale aware**, you also need to design your features and your code to be localizable. Meaning **you can translate the user interface and customize features for specific locales without having to change your source code**. Most developers just **put their UI in the code**. **That doesn't work**. You need to **move it into resource files**.

[Daniel] Yeah, I would like to add to that. We spoke a little about **one binary for the product**, and this is include also this resource externalization. So we need to take all this stream and externalize them to resource files, but it's all not on this stream, can be other other things. We can think about images, we can think about calls, we can read font, and other resources that might be different from one market to another market, from one language to another language. Now, you need also to enable, you code to handle input and display and processing by direction on languages, like Arabic and Hebrew, complex script like Hindi and ideographic languages like Chinese and Japanese. This is really important, really, to have the possibility first to display and second also to have the input method or what we call IME to handle those inputs.

[Bjorn] Yeah. The **input method editor, IME**, is something that people who don't speak **Japanese or Chinese** or any of the other languages that use IMEs, they're not aware of that **you can not enter the thousands and thousands of characters or ideographs that are used in these languages just using a simple keyboard**. So what you're doing is actually **use a kind of a pronunciation** of it. And you **type it in** and you **get different candidates and then you select one**. And this is something that obviously the operating system handles for you but you need to be aware of it.

And then for the last, this is really an important point is that **localization can be done as localize as you go**. For the languages and locales you plan to support, translate none, some or all of the user interface. You need to **resize the dialogue boxes and you need to customize features for the local market if necessary, and test to make sure that everything still works.** But you don't have to do all languages at the same time.

[Daniel] Yeah, I would like to add to this. So first, we need to think about localization as a separate stage that will come after the software development. It actually can be integrated as part of the software development. Especially if it's an agile development, can be part of it. And as you said, **you don't need to translate all to localize for all languages at once. You can actually have several tiers of languages**. You start with one languages and then in tiers of five or ten important languages, and then other tiers. And you can translate, you know, through, you know, the life of the product itself.

Now you need also to **build everything in SQL and worldwide binary**, and this is what we discussed earlier. **You need really to have one binary**. This is **important** first **for the maintenance of a program**, but also it's good for the development itself. You know that for and et cetera, you have only one binary that you have, and you can release this binary and later add more and more languages whenever you have them ready for those markets.

[Bjorn] Yeah. And the cool thing about having this **single binary approach** is, like, well, it's kind of the negative, or the positive of what we were talking before, right? **You now can have one service pack. You can have one patching strategy. And all of the features are already built into the signal binary**. And that then allows you, I mean, also to look at stuff like **sim shipping or simultaneously shipping your software**, which is really an amazing thing once you get to a certain size that you can go out to like 20 markets or even **like Windows does it to over one hundred markets in different languages within a very short release delta** because everything is so **streamlined**.

[Daniel] Yeah, I think the **short release delta** is something very important. Clearly the dream is to sim ship and have all the versions on the same day, but it's okay and think again about agile development of fast release of weekly release or release every month. You can release in two languages and then a few days later to release more and more languages. But again, the delta should be very, very, very short. The delta should be very short. Experience the joy of faster innovation, happier customers, improve healthy relationship and increase success.

[Bjorn] Yeah. I mean that's obviously like motherhood and apple pie

[Daniel] Yeah.

[Bjorn] We just threw a lot of terms at you and you should worry a bit because we will test you on this or not. This concept overlap and **people across industry use terminology in slightly different ways**. Even we switched from **world ready**, and **international**, and international to world ready. And we will slip a few times throughout the course, but we are trying to stay consistent. To begin immersing yourself in the concept we will build on for the rest of the course, I recommend spending some time with the glossary that we have put together.

7.2 The Process of Creating World-ready Software

If you plan ahead, writing software that people all over the world can use (meaning, it will work the way they expect) can be relatively straight-forward. **Your end goal is to engineer software that's world-ready**. Ideally, you can **create a single binary executable** that you can ship for all language editions of your software, **with the translated elements in separate modules**. The more executables you have, the larger and more expensive your testing effort will be.

As you can imagine, there's a process for making software world-ready.

First, **let's talk about the wrong way** to approach this process.

1. Design software that's great for your market and works for the language you speak.
2. Write the code, test it, finalize it, and ship it.
3. Suffer over the scathing online reviews of your software from people outside your market who tried to use it and couldn't, because it doesn't support entering the characters their language uses, it doesn't display text according the rules of their language, or it formats dates and numbers incorrectly.
4. Say "oops," give the software you already shipped to a separate team (or, as we used to say, "Throw it over the wall"), and tell them to make it work for half a dozen other languages.
5. Get phone calls, emails, and visits with reports on how your software breaks in all kinds of fun and interesting ways when the international team tries to make it work for other languages.
6. Be forced to re-architect and clean up your code when you're busy trying to get your next rev to market.
7. Ship both your international editions and your next rev later than you planned, miss out on sales, and experience slow adoption because people who tried to use your software before have already written it off.
8. Triage complaints from international customers about features that are missing or still don't work correctly, because you focused on languages and not markets, which have differences that go beyond languages.

If this is your process, you will have lots of problems—we give plenty in this course—and you will spend more time and money than you should fixing them.

Here's the right way to do this.

1. Get a commitment up front from everyone on the team to do their part to globalize your software. That is, ensure you do not design the features or code that make up the core of your program around a single language or market.

2. Make sure your design and your code are locale-aware, that they account for differences in conventions based on language, country or region, and culture. Locale-based information includes sort order; keyboard layout; and formatting for dates, times, numbers, calendars, and currency.

3. Design your features and your code to be localizable, meaning you can translate the User Interface and customize features for specific locales without having to change your source code.

4. Enable your code to handle input, display, and processing of bidirectional languages, like Arabic, complex scripts like Hindi, and ideographic languages, like Japanese.

5. Localize as you go. For the languages and locales you plan to support, translate none, some, or all of the user interface; resize dialog boxes; customize features for the local market if necessary; and test thoroughly to make sure everything still works.

6. Build and maintain a single, worldwide binary – one program file that can be used "as is" for any language edition of your software.

7. Sim-ship (that is, "simultaneously ship") your software to multiple markets at the same time – on the same day or within a short release delta.

8. Experience the joys of faster innovation, happier customers, improved health and relationships, and increased financial and personal success.

We just threw a lot of terms at you. These concepts overlap and people across the industry use the terminology in slightly different ways. We'll be consistent throughout this course, however. To begin immersing yourself in the concepts we'll build on for the rest of the course, we recommend spending some time reviewing the terminology in Unit Five of this module.

8 Terminology

8.1 Video Transcript

[Bjorn] Hi, I'm Bjorn

[Daniel] Hi, I'm Daniel and we're here to discuss some common terminologies.

[Bjorn] Yes Daniel, I believe we have some disagreement here at times, when we talk about these very simple terms that are used in developing international software.

[Daniel] Such as?

[Bjorn] Internationalization.

[Daniel] What is internationalization, go on.

[Bjorn] So for me internationalization is the process of creating software for multiple markets around the world.

[Daniel] Yeah for me internationalization is actually, engineering of the product to enable efficient adaptation of the product for the local requirements.

[Bjorn] I mean it's almost the same, but we have slight distinctions there.

[Daniel] Yes, I agree because basically I'm looking more for the engineering process really. How we can take the product and make it more generic, then later we can customize it to different markets.

[Bjorn] When I was on the international team, I actually never really liked internationalization because what does it actually mean? Is international everything outside of the USA? Or is international just something that is not local? So I think this term is actually overused, but you need to understand what it means.

[Daniel] Yeah, I agree with you completely, but I'm looking at internationalization not just will be outside the USA. I'm thinking just how to make the product as generic as possible. Anything in the product should be generic.

[Bjorn] Yeah. Okay, then the next term, obviously, that gets used a lot is globalization. Globalization is a cool term, it obviously has a completely different meaning in finance terms or something. But for us it means, it's the process of making your software world ready. And world ready is obviously something we still need to define.

[Daniel] Yeah. [LAUGH]

[Bjorn] But, I love the term world ready. But, globalization your software means that you write the parts of your code that deal with the functionality, like text and input display, sorting, and so on. And then you make this generic. And then, that means that your code works, as is, for multiple languages and locations.

[Daniel] Yeah I agree with your definition. Clearly I have a slight different definition style. Certainly the globalization readaptation of the marketing strategy to regional requirement of all kinds. So I'm not looking on only the engineering side, as you discussed but more about how we should adapt the product really for marketing strategy?

[Bjorn] You see for me what you're saying there, with the marketing strategies, this is what I call market customization.

[Daniel] Yeah, differently we have some difference in the definition of those terms. But I think we agree that, what is internationalization and what is globalization.

[Bjorn] I think as long as we remember that globalization in software is not something bad that destroys local jobs, but it actually creates jobs. And it's not something like marketization where you go into a market and try to blast everybody with your requirements so that they can use your product. It's actually something that you're doing for the rest of the world to make software run, or making your marketing campaigns successful in other countries.

[Daniel] Yeah, I agree with you, I'm thinking about the user that's sits in New York, or in London, or in Paris, or in Delhi, or in Shanghai, or in Beijing.

[Bjorn] In some city we get it.

[Daniel] In some city. And basically we need to adapt the software for him. He doesn't need to be adapt to all software, we need to customize the software and the product for this user, which is somewhere in the world.

[Bjorn] Yeah, so for localization or localizing something what's really important for me is that people understand it's not translation. Translation is just taking something from one language to another. With localization we're actually talking about customizing software for a specific market. Customization obviously includes translation of the user interface and documentation, but we're also talking about offering features that are specific to a language or market. The more you localize, the more effort it takes, but the more you get in return.

[Daniel] Yeah, I agree with you Bjorn. I think that you know, localization is the process of adapting a software or a product and a company material to see the target, market local. This is the process of localization.

[Bjorn] Yeah and the other thing that's important to look at is, that often times when you look at the large software team there is a localization team. And that's a little bit of a misnomer because localization teams often cannot really adapt and customize features. They are mainly there to work on the translation of the UI and the documentation. So, this is something were, like all the localization teams, want to have the developers understand, that it's more than translation, but often times it is limited to translation. And then another aspect that we should mention here is, often, like when we look at Microsoft as localization, we always have no compile localization, which means we're taking all the resources, everything that's used in the UI, and move them into something like text files or into a database these days. And then we're localizing this database, and then we're putting the terms back into the UI or into the software, into the binary files without actually recompiling the source.

[Daniel] I have a question for you Bjorn. You mentioned earlier that you like the term world readiness. Can you tell me what is world ready?

[Bjorn] Sure, sure. Actually I love world readiness, world ready is something that I'm kind of proud of, because my team back in 2002, or something.

[Daniel] Whoa.

[Bjorn] Like we started to use this term, and I actually started a team at Microsoft which was called the world ready guides. We wanted to go away from all this terminology that can be so confusing. Internationalization, globalization, localization and we just wanted to talk to people about being world ready. I mean it's so simple, nobody can misunderstand what world ready means. It means ready for the world, right? You designed and coded to have, like that you can go into any market without redesigning or recoding. You can create additions of code

for a market, of your software for markets all over the world. And it will work the way people in those market expect.

[Daniel] Yeah. Well thank you Bjorn for the definition.

[Bjorn] Yeah Daniel, I believe definitions are really important to understand what we're doing here. And as I said world readiness is something close to my heart but as long as you do the work, you can call your software whatever you want. Thank you all.

[Daniel] Thank you.

8.2 Terminology

TERMINOLOGY

Wow, a glossary, everyone's favorite read! Actually, this unit serves two purposes. The first is as a reference. The second is as a conceptual aid. This is not a full glossary of terminology used in this course and the next two, but it contains enough definitions to get the point across that there are many details to consider when designing and building world-ready software.

CORE CONCEPTS

Globalize or Globalization The process of developing software whose features and code design support the input, display, and output of multiple languages and locale-specific data.

Internationalization The process of globalizing software and making it localizable. Often used interchangeably with "globalization."

Level of localization The amount of translation and customization completed for a specific language edition of software. Levels range from translating nothing to shipping a completely translated product with features customized for the local market.

Locale A set of conventions based on language, country/region, and cultural standards:

- Written script(s)
- Direction of text
- Currency symbol
- Long-date format
- Short-date format
- Time format
- Calendar
- Default paper size
- Decimal separator
- List separator
- Thousands separator
- Sort order
- Keyboard layout

Locale-aware or locale-sensitive Exhibiting different behavior or returning different data, based on locale.

Localize or Localization The process of adapting software for a specific local market, which may include translating the user interface, resizing dialog boxes, customizing or adding features (if necessary), and testing results to ensure that the software still works as expected.

Localizable Used to describe software that is straightforward to translate and customize for specific locales without requiring changes to source code.

Single binary or Worldwide binary A single program file that can be used "as is" for any language edition of your software.

World-ready Software that works the way people in different areas of the world, who speak different languages, expect it to. The design of its features, and the way its code is organized, makes localization straightforward.

CHARACTERS AND LANGUAGE (NON-TECHNICAL)

Accented character A character that has a diacritic attached to it, such as 'ö.'

Alphabet Elements of a writing system composed of a collection of letters that have a one-to-one relationship with a sound.

Alphanumeric Consisting of either letters or numbers, or both.

Bidirectional Text that includes characters that are read from left-to-right and characters that are read from right-to-left. Most Arabic and Hebrew text are read from right-to-left, but numbers and Western terms embedded within Arabic or Hebrew text are read from left-to-right.

Case Two distinct variations or forms of the same character within the same alphabet. These variants, which differ in shape and size, are called "uppercase" letters (also known as "capital" or "majuscule" letters) and "lowercase" letters (also known as "small" or "miniscule" letters).

Character (1) The smallest component of a writing system or script that has semantic value. A character refers to an abstract idea rather than to a specific glyph or shape that a character might have once rendered or displayed. (2) A code element.

Collation A set of rules that determine how textual data is sorted and compared.

Combined characters or ligatures Characters that join into one character when placed together. One example is the "ae" combination in English, which is sometimes represented by a single character "æ." The Arabic script has many combining characters.

Complex script A script that requires special handling when it comes to shaping and laying out characters based on linguistic requirements. Complex scripts can have any combination of the following attributes: bidirectional rendering, contextual shaping, combining characters, as well as specialized word-breaking and justification rules.

Cyrillic script The script traditionally used for writing various Slavic languages, including Russian.

Diacritic A character that is attached to or overlays a preceding base character. Most diacritics are non-spacing, meaning they don't increase the width of the base character.

Diaresis Two dots placed over a Latin vowel to indicate that the vowel is pronounced as a separate syllable, as in “naïve.” Typically used when two vowels are adjacent, but should be pronounced separately rather than as a diphthong.

Digraph Characters written separately but forming a single lexical unit (for purposes of sorting)—for example, the Danish “aa” and the Spanish “ch” and “ll.”

Glyph The actual shape (bit pattern, outline, and so forth) of a character image. For example, an italic “a” and a roman “a” are two different glyphs representing the same character.

Ideographic character A character of Chinese origin representing a word or a syllable that is generally used in more than one Asian language.

Kanji The Japanese name for ideographic characters of Chinese origin.

Latin script The set of 26 characters (A-Z) inherited from the Roman Empire that, together with later additions, is used to write languages throughout Africa, the Americas, parts of Asia, Europe and Oceania.

Letter (1) The basic element of an alphabet. (2) A higher level of abstraction than Character. For example, both the Spanish “ch” and the Danish “aa” are considered single letters for some purposes.

Ligature Two or more characters combined to represent a single typographical character. The modern Latin script uses only a few. Other scripts use many ligatures that depend on font and style. Some languages, such as Arabic, have mandatory ligatures; other languages have characters that were derived from ligatures, such as the German ligature of long and short “s” (ß) and the ampersand (&), which is the contracted form of the Latin word “et.”

Logograph or logographic From the Greek “logo,” meaning “word”: a letter, symbol, or sign used to represent an entire word.

Lower case Denotes letters that are not capitalized. The notion of lowercase does not apply to East Asian or Middle Eastern scripts.

Morpheme The smallest meaningful unit of a word. The word “dog” is one morpheme. The word “dogs” is two morphemes: “dog” + the plural marker “s.” Many ideographs are based on morphemes.

Phoneme A unique individual sound used in a language.

Reading order The overall direction of a sequence of text. Whereas words in a given script always flow in the direction associated with that script (for example, left-to-right (LTR) for Latin, right-to-left (RTL) for Arabic and Hebrew), the flow of the sentence itself depends on the reading order. For example, a mixture of Arabic and French text can be regarded as French embedded in an overall Arabic sentence, implying RTL reading order, or as Arabic embedded in French, implying LTR reading order.

Script A collection of characters for displaying written text, all of which have a common characteristic that justifies their consideration as a distinct set. One script can be used for several different languages (for example, the Latin script, which covers all of Western Europe). Some written languages require multiple scripts (for example, Japanese, which requires at least three).

Separators Symbols used to separate items in a list, mark the thousands place in numbers, or represent the decimal point. They vary based on locale.

Syllabary A set of written characters in which each character represents a syllable (for example, a consonant sound followed by a vowel sound). Examples of syllabaries include Japanese katakana and hiragana and the Indic scripts.

Writing system The collection of scripts and orthography required to represent a given human language in visual media.

CHARACTERS AND LANGUAGE (TECHNICAL)

ASCII Acronym for American Standard Code for Information Interchange, a 7-bit encoding. Although limited, ASCII's set of 128 characters is the common denominator among most standard character sets.

ANSI Acronym for the American National Standards Institute. The Windows code page 1252 was originally based on an ANSI draft that became International Organization for Standardization (ISO) Standard 8859-1.

Base character An encoding point that does not graphically combine with a preceding character and that is neither a control nor a format character. The Latin "a" is an example of a base character.

Charset or Character set A set of characters used in Windows, also known as a "Code Page"

Code page An ordered set of characters of a given script which associates a numeric index (code point value) with each character.

Code point, or code element (1) The minimum bit combination that can represent a unit of encoded text for processing or exchange. (2) An index into a code page or a Unicode standard.

Combining Character A character that combines graphically with a preceding base character. The combining character "applies" to the base character. The combining acute accent mark (‘) – U+0301 – is an example of a combining character.

Enable or Enabling Altering software code to handle input, display, and editing of bidirectional languages such as Arabic and Hebrew, or Asian languages such as Chinese and Japanese.

Encoding A method or system of assigning numeric values to characters (for example ASCII or Unicode).

Extended Characters (1) Characters above the ASCII range (32 – 137) in single-byte character sets. (2) Accented characters.

Floating Accent or Floating Diacritic A combining character that overlays the preceding base character; it can potentially change position or shape according to the shape of the base character. The combining right arrow above ò, character (U+20D7), is an example of a floating diacritic.

Font Any of numerous sets of graphical representations of characters that can be installed on a computer, printer, or another graphic output device.

Input Method Any method used to enter text or data, including keyboards and Input Method Editors, as well as voice-recognition engines or handwriting-recognition engines.

Input Method Editor A program that performs the conversion between keystrokes and ideographs or other characters, usually by user-guided dictionary lookup.

Keyboard layout A standard arrangement of characters on a keyboard that defines which keys produce particular characters.

Layout The order, positioning, and spacing of text or other user-interface elements.

Multilingual Supporting more than one language simultaneously. Often implies the ability to handle more than one script or character set.

Non-spacing character A character, such as a diacritic, that has no meaning by itself but overlaps an adjacent character to form a third character.

Shortcut key A keyboard combination that activates a software command directly, as an alternative to activating the command through menus.

Spacing character A character with a nonzero width.

Text element The smallest unit of text in a script that can be displayed or edited.

Unicode A worldwide character encoding that includes most of the world's scripts; it is developed, maintained, and promoted by the Unicode Consortium, a nonprofit computer industry organization. Unicode encompasses virtually all characters used widely in computers today. It is capable of addressing more than 1.1 million code points. The standard has provisions for 8-bit, 16-bit and 32-bit encoding forms.

9 Course Name Change

[Bjorn] Now that you know more about internationalization, localization, and world readiness, you will understand that developing international software is not really a good title for the course. What does international actually mean? Does this mean outside of the US or other countries? Well, so from now on we will talk about developing world-ready software. Let me do the course introduction again. Welcome to Developing World-Ready Software. My name is Bjorn. [LAUGH]

10 Basic UX Design Considerations

10.1 Video Transcript

[Bjorn] Excellent, so we learned a lot on how to design your team and lots of other things, but I believe what's really necessary is to talk about, what are the components of the user interface? Of the user experience, or UX? And the user experience consists of multiple items that need to be localized. One of that is menus and dialogues. Right? I mean menus and dialogues are kind of the interface that you have to your software, and we need to make sure that those are localized. So is there anything special that you need to consider when localizing the menus and dialogues over?

[Uwe] Right. So we touched on a few items already. You have to make sure that you expose all those strings to your translator and then often times what you have in menus and dialogues is hot keys. They're called hot keys or accelerator keys where you just can use your key board to get to a certain menu item right? And what you have to make sure is that you let localizers change those so it makes sense. And that can lead to a lot of problems because then you might have to use the same accelerator key a few times in dialog. And then the user doesn't get what they want. So then you have to think about, well if I allow that, I have to test it as well. So you can run into a few problems there if you use accelerator keys. That's a good design decision, how you do your dialogs.

[Bjorn] Yeah. And especially with the keyboard shortcut, one thing that came to my mind that I saw very early on in my localization career is that sometimes you're using a shortcut key which is actually not on the standard keyboard or not easy to get to on the standard keyboard of that country. So this is something that you need to

look at. There are resources available online where you can look at all the keyboard layouts and just checking out whether the tilde key for example is a standard key on the keyboard or if it's a combining character is super important to look at. I can send other graphics. We already talked about that we should try to avoid localizing images as much as possible. And we mentioned also that some of the icons are very culturally sensitive. What other things do we need to consider when we look at this?

[Uwe] Yeah, I have a good example for icons. When I first came to the United States, in a mail application it uses mail box where the flag goes up and down. And I had no idea what that was and that is not how our mail is sent and delivered in Europe. So when the flag was up and it's supposed to mean something I was confused and didn't understand at all.

[Bjorn] Right.

[Uwe] So it's a good example of if you use an icon to convey something that it might not be understood at all in other cultures. So if you want to use icons, you have to really make sure that they're globally accepted. And you also have to make sure that they're not, that they're not politically sensitive.

[Bjorn] Right, well or you create an icon that becomes an icon around the world and you're just setting a new standard, let's hope that happens. So, audio and video, very similar, very, very similar to icons and graphics. If you use audio, try not to use verts in audio, like don't try to record texts or messages because those are really hard to localize. Same is true with video. But obviously if your application needs that, then you need to bite the bullet and have it localized. It's impossible to extract resources out of it. You're basically asking for dubbing or a reshoot. Also when you consider audio, lots of people think it's funny when you make a mistake to say oops or something. Just remember not everybody around the world says oops or ouch or whatever. This might be not something that you realize but each language has their own oops and ouch sounds. And each language has their own animal sounds too. So it's, all this stuff does get localized. It's very, very specific to countries and you need to make sure that you're following the rules. We talked a lot about error messages. So in an earlier video we talked about concatenated strings and all of that. Other things that are important in error messages, what would you say for error messages? What's the one thing that people, who haven't worked for 20-some years in localization, often screw up?

[Uwe] Well, I think for our messages, often they are written in a way that they are not really helpful, they're very technical. Then the runtime concatenation using of variables to put certain characters or even words back into the string is always very hard for a localization company to really do a good job on.

[Bjorn] Right.

[Uwe] So it's just of developers try to be very efficient. Reuse and reuse the text they've already typed, and then they'll paste it together and think, oh, wow, I've written very efficient code and it makes the life for the localization company so much harder.

[Bjorn] Right.

[Uwe] So, that's really the biggest thing I've seen over the years.

[Bjorn] I just say the best error message is something like Percent S could not be Percent S. That's obviously super easy to localize, you know exactly what Percent S is and you know exactly what Percent S is and you can just be sure that this error message will just work just fine. So no it doesn't. Make sure when you do an error message, even if you have to write the same sentence ten times and you say, file could not be opened. File could not be printed and so on. It looks like you're wasting time and wasting resources, but you're actually saving money later on when you're going for a world ready design and when you're going for world wide market, because you don't know if the grammar changes. If stuff has to be re-sorted. In some languages the first percent s has to move to the second place and so on. And how are you doing this if you're just saying percent s has to be percent s or could not be percent s. So that's really I mean don't just think about like what we used before in an example that you attach an s to make it plural. It's also like the basic forms of grammar that needs to be followed like work order which is different. That actually leads me to another favorite of mine which is the formatted text, so anything that has date, time, currency and so on in there needs to be considered as a very, very special case, because you cannot assume. I mean it's very obvious to most people, that the currency in lots of countries is different. I mean, even the person who has never left their home country kind of understands that other countries are using different currencies. What's not obvious to everybody is the different time and date formats are used. You understand that there are different time zones, but that this is different formatted. Uwe, can you give me a few examples of that?

[Uwe] Yeah, we had the one example already in a previous video about Spain mixing up the day and the month order, right. And-

[Bjorn] Well actually, I wouldn't say they're mixing it up. They're just having a different format to the US.

[Uwe] No, yeah.

[Bjorn] You're becoming very US-centric, Uwe.

[Uwe] That's right. Yeah, that's It's what happens when you've lived here for a long time. Exactly. So but again the recommendation we made at that point is don't reinvent the wheel. Don't try to do your own thing. Use what is already available in the operating system to support you in terms of times and date and currency.

[Bjorn] Yeah, and just to mention this very, very briefly, you also need to consider when you're having a global application that runs worldwide, that you should standardize, like for your log files, on one time zone. Because otherwise you run into lots of problems of synchronization. This is just a side topic. Now lots of people like to make their UI or UX super special by using a really cool font. Comic San Serif comes to my mind there.

[Uwe] Right, yeah.

[Bjorn] Or something which is actually cool, but what could be a draw back of that?

[Uwe] Well, it sounds like a great idea, and you can make it really pretty, and it's not to expensive because, you know, you don't have that many characters if you start with English. But think about Asian languages, Arabic, Hebrew, you don't use any of these, right? So then you have to get a font that has all these characters that you need to support. And then it becomes really expensive really fast. So you're much better served with looking what font is available in the operating system of the platform that you're working on and use one of those.

[Bjorn] Mm, Mm. Yeah and then, when you look at other text right? Lots of people forget that there is text that is supporting your application. And you have what we in general call content. So you have help files or documentation, perhaps you don't ship, obviously, local help files anymore. But you have a website that has all the user information to get them going with your app. And one thing I've seen often times is that we use metaphors or slang. And this stuff is hard to translate. Like a touch down pretty much only means something in the US. It means a lot in the US but not much outside of the US. Because most countries do not follow US football. And these are just generally hard to translate. And then we thought we were kind of going away from packaging a lot and we initially we didn't even want to talk about it, but IOT, Internet of Things, right. You're actually starting to have hardware again, and you need to package it somehow. And that's something that we need to consider. Is the packaging appealing? Do we have words on there that we need to have and stuff that we need to cover. Which, last but not least, comes to the name of your product. Name of your product is, A, you need to make sure that it's not offensive somewhere, that you're not using a word that doesn't work in another language. What other things with the naming of a product that should be considered?

[Uwe] Well, first of all you have to decide is my name a global name and does it make sense in all the countries of the world. Sometimes you have little prepositions in the name and usually you want to translate those, at least for some languages. So I would recommend if you invent a new name, make sure it doesn't mean a swear word in another language. We have lots of examples of that from the past, right? And make sure it's really, it doesn't have the way that it has to be translated. So be really smart when you, when you come across with your brand name.

[Bjorn] So one resource we have at Microsoft that we really need to point you to, and it's super helpful. And we will obviously put the link into the resource section of this course, is the Microsoft style guide. The Microsoft style guide gives you the general layout, what the file menu should look like and what other menus should look like, what the dialogue box would look like and so on. But it also gives you the standard translations of all of that. So all of this will help you to make your UX, your user experience as useful as possible. And as easy to make it world ready.

10.2 Basic UX Design Considerations

A great User Experience (UX) makes software easy and enjoyable to use. The next course in this series, which is about design, will cover this topic in detail. For now, we'll describe at a high level how globalizing your software can affect basic UX elements, which will be helpful for planning purposes.

Some people equate "User Experience" with the User Interface (UI), but the UI is a subset of the User Experience. UX design also includes how features work, for example, what steps a user has to follow to complete a task. Any feature you design should, as much as possible, work the same way across all language editions of your software. This simply means that when you make design choices, pick options that will work for most or all of your target markets. If any functionality needs to vary from market to market, isolate it in your design and code to make it easy to swap one market-specific module for another, as some programs do with grammar checkers.

The User Interface

The UI is, of course, the most visible part of your software. It includes elements with text, such as title bars, menus, dialog boxes, buttons, messages and tool tips, as well as keyboard shortcuts. The trick is to design your UI so you don't have to redesign it if the translated text ends up longer or taller than the original. If your design is too crowded, you may have to resize or even relocate UI controls, which can be time consuming and expensive.

In addition to text length, font size can also differ among language groups. For example, East Asian languages usually display text in a larger font size than other language groups, which means that edit boxes, static text, and button controls need larger vertical spacing. Bidirectional languages like Arabic and Hebrew require "mirroring" of the screen layout, including menus, text, dialog boxes, and edit boxes, to accommodate right-to-left (RTL) reading order.

The best way to ensure that the UI has a consistent look and feel across all editions of your software is to design it up-front to accommodate the variances that language and locale will introduce, such as longer text on dialog box buttons. At the same time, whoever is responsible for translation should only change the UI layout design (for example, rearrange elements within a dialog box) if absolutely necessary.

Terminology

Terminology that is clear, concise, and consistent is key to ease-of-translation as well as ease-of-use. Clear phrasing of the original text helps translators interpret the text's intent correctly and translate it accurately. This makes colloquialisms, idioms or metaphors risky, as they may have no equivalent in another language. They can be difficult to translate, and may potentially offend some cultures (even if they speak the same language). Conciseness is important for reducing the size of translated text—keep in mind that many translation services charge by the word. Consistent terminology is particularly important when automated translation tools come into play. Two words may have similar meaning in one language, but the meanings of their automated translations may differ markedly. Whenever possible, use standard glossaries for common software terms such as "File" "Edit" "Save" "Delete" and so forth.

Graphics and Multimedia

Graphics, such as photos or icons, have the advantage of communicating more universally than text, provided they are not culture-specific. Choose graphics that are not biased toward a particular culture or locale, and rather than embedding text within graphics, overlay it. Having to source new graphics for additional markets or languages can get very expensive. In addition, having different images can confuse people who use more than one language edition of your software (for example, people who work in multinational companies or carry multiple devices configured for different languages).

Also keep in mind that graphics used to represent abstract concepts may have different meanings in different contexts. Examples include icons that imply direction, such as "Undo" and "Repeat." An arrow pointing left could mean "undo" in languages that read left-to-right, but "redo" in languages that read right-to-left.

Formatting Data

Operating systems or programming language libraries offer functions or Application Programming Interfaces (APIs) that handle nuances in number formatting, sorting order, capitalization, and other locale-dependent functionality. Make use of them whenever possible. If the system does not provide the functionality you need to display a specific type of data differently based on language or market and you have to write it yourself, make sure you put the functionality into a separate component. That way it will be easier to modify when you add support for new languages and markets.

Help and Supports

If you cannot offer a fully localized product, prioritize localizing the product components that will help your customers have the best experience. If you have a great UI design and use standard terminology, then people who do not speak the language your UI is in (or do not speak it well), can usually get by as long as you have fully translated help files they can consult.

11 Basic Engineering Considerations

11.1 Video Transcript

11.2 Basic Engineering Considerations

When it comes to globalizing software, there are enough engineering considerations to fill an entire course (in fact, we are creating one). For the purposes of this course, we will explain at a high level what type of engineering work world-ready software requires.

Data Input

People input data via a keyboard, a mouse, inking (handwriting), or voice. Software, with help from the operating system it runs on, receives the keyboard strokes, mouse clicks, writing, or speech that someone enters, interprets it, and then takes the appropriate action. That's just a complicated way of saying that your users need a way to get words and commands into your software. Different countries use different standard keyboard layouts, arranged to best facilitate entering text in the local language or languages. Characters may swap positions from one keyboard layout to the next. Since standard keyboards cannot possibly accommodate the thousands of characters that Asian languages use, an additional piece of software called an input method editor, or IME, is necessary to convert multiple keystrokes into a single ideographic character.

For properly managing input based on locale, the operating system is your friend. It contains drivers for multiple keyboard layouts, and ships with standard Input Method Editors so you don't have to write your own. Recognition engines, which are cloud-based these days, convert speech and handwriting into text or commands. Given how much computer science and big data goes into properly converting even one language, we suggest using engines that someone else built rather than trying to construct your own, unless you are a linguist or a researcher, in which case go for it.

Data Manipulation

"Data manipulation" may sound daunting, but it simply means changing the data in some way, for example sorting a list, capitalizing a letter, formatting a date, or comparing one string of text to another. Operating systems and programming languages offer functions for these types of operations, which are dependent on language or cultural conventions or both.

Converting data from one system to another before transmitting it or after receiving it to preserve data integrity is what most people mean when they say "system interoperability." E-mail programs, for example, convert timestamps to the message recipient's time zone. Properly globalized software will convert data to the form each market expects, for example dates, times, and numbers with the correct formats. It will also convert between character encodings so that the right characters appear on screen instead of question marks, boxes, or the wrong script (see the unit on terminology for an explanation of encodings).

Data Display

There are two steps to displaying data, specifically text: laying it out and rendering it. Operating systems contain functions to help with text layout, which isn't always a straightforward proposition.

For example, Arabic scripts (used for languages such as Arabic, Farsi, Pashtu, and Urdu) and Hebrew scripts (used for Hebrew and Yiddish) are read from right-to-left (RTL). For these scripts, software needs to convert the logical order (how the user enters text) and the visual order (how the characters appear on screen). It gets more complex when text mixes left-to-right (LTR) and RTL characters. For example, positioning characters and moving the caret in a bidirectional context can be particularly challenging.

Another aspect of text layout and display is contextual shaping of characters. For the Arabic and Indic family of languages, a character's glyph (that is, the actual shape of a character image) can change significantly depending on the glyph's position within a word and the characters that precede or follow it. Some characters in Arabic have several different shapes depending on their context. Here's where it gets complicated. Standards like Unicode assign each distinct character a distinct number, or code point, so that computers can identify them. Different shapes of the same character don't have distinct code points, they just have different glyphs. An example of three

characters with the same code point but different glyphs is “a” vs. italic “a” vs. bold “a.” Fonts contain multiple glyphs for the same character, and they have their own tables. The third course will cover this topic in far more detail. All you need to know for now is that when it comes to laying out and displaying bidirectional text, “It’s complicated.”

Word and line breaking, hyphenation, and word wrapping can also be complicated, particularly for text containing multiple languages. Latin script follows straightforward rules, such as breaking a line at a space, tab, or hyphen. For languages like Thai, Khmer, Chinese, Japanese, and Korean, words run together. There are no spaces between the end of one word and the beginning of the next, which makes word breaking in such languages more complex, requiring grammatical analysis and word matching in dictionaries.

Take Japanese, for example. Japanese line breaking is based on the kinsoku rules—you can break lines between any two characters, with several exceptions:

- A line of text cannot end with any leading characters, such as opening quotation marks, opening parentheses, and currency signs, that shouldn’t be separated from succeeding characters.
- A line of text cannot begin with any following characters, such as closing quotation marks, closing parentheses, and punctuation marks, that shouldn’t be separated from preceding characters.
- Certain overflow characters (such as punctuation characters) are allowed to extend beyond the right margin for horizontal text or below the bottom margin for vertical text.

You see, it’s complicated.

Have Compassion For Your Users

If you have read both this unit and the unit on basic User Experience design considerations, you now have a better idea of why creating international software can be challenging, time consuming, and expensive. If you ship your software without globalizing the design or the code, imagine how frustrating life could be for some of your international users. Dialog box text looks squished together, the “undo” icon looks backwards to you, characters sort in the wrong order, short date formats are reversed, you can’t type kanji characters into your documents, the software displays the wrong glyph for an Arabic character in the middle of a word, and line breaks split words in half in nonsensical ways. Watch the next video for some real-life examples of frustrating user experiences.

12 UX and Engineering Errors

13 World-Ready Marketing

World Ready Marketing

A Microsoft subsidiary once ran into trouble when they photoshopped an ad used in the United States before publishing it in their country. They replaced the head of the African American model in the ad with the head of a white person. It might have gone unnoticed, but they neglected to change the color of the model’s hands. Whereas they may have felt they were trying to make the ad visuals more representative of their country, which has a small black population, their actions came across as racist. Microsoft took down the ad and issued an apology. But the press still pounced. One headline labeled the incident “one of the biggest racist Photoshop disasters ever.” And the story lived on when the tech news blog TechCrunch sponsored a contest encouraging readers to submit parodies of the ad.

Marketing campaigns are no different than software when it comes to cultural sensitivities. If you’ve ever watched a compilation of “the best television ads of the year” you probably said to yourself, at least once, “How on earth did they get away with that?” What’s considered catchy or edgy in one culture may be too much for another.

Unless you work for a company with deep pockets that is willing to give you funding, you won’t have a big marketing budget anyway. But you can still ensure that the marketing you do disseminate has a better chance of resonating in multiple countries.

Consider branding:

1. **Your company name**: Startups these days like to choose cool-sounding names. One technique is to take a word relevant to the business (which ideally can be used as a verb) and give it a clever spelling, for example, deleting a vowel or changing "ph" to "f" or "k" to "c." People who do not speak your language, however, may not grasp the reference behind the original word, much less the respelled one.

2. **Your product name**: Choose wisely. Just remember the lesson of the Chevy Nova, a car whose name means "bright, yet dying star" in English but "doesn't go" in Spanish.

3. **Your tagline**: You may come up with a pithy, poetic tagline for your product that makes perfect use of metaphor, alliteration, and clever homonyms. And all the pithiness, poetry, and alliteration will disappear the moment it's translated into another language. The homonym effect will be lost, and the metaphor will no longer make sense.

4. **Your color palette**: See Module One for a discussion on colors and cultures.

5. **Your logo**: If your product logo is a representation of a "thing" make sure the representation is accurate, meaningful, and acceptable across cultures. A US mailbox is not international, a dragon fruit is unfamiliar to many non-Asians, and some people eat animals others view as sacred.

As branding is a combination of the elements described above, you need to think not only about the individual pieces, but also what they signify when put together. Choose a color scheme for your icon whose symbolism has a wide range of interpretation globally, use an obscure literary reference for a product name, and a tagline that makes brilliant use of local slang, and without realizing it you could be marketing death for kittens on the other side of the world.

Sending Your Message

Messaging forms the core of marketing campaigns and your tagline is just one expression of your messaging. Visuals also send a message, as does placement—advertising a product on a luxury website sends a different message from advertising on a discount website. One of your instructors once worked with an Asian hardware company that made a very upscale, fashion-conscious device and chose to sell it in the United States exclusively through a store that Americans consider a discount retailer, something they did not realize until after they had closed the deal.

Sometimes it works to use the same messaging in international marketing campaigns, translated into multiple languages. Then again, sometimes it doesn't. Different features may resonate in different markets. Perhaps what you consider a minor feature becomes the major selling point in one or two markets, something you would never have expected. Even the perfect translation of the perfect marketing copy could seem awkward because it doesn't translate culturally. To give an overly simplistic example, saying that your product will protect people from the cold of December doesn't make to people who live in tropical climates, or in the Southern Hemisphere, where it's summer in December.

Visuals and graphics can be particularly problematic, as our opening story demonstrates. Some cultures think nothing of using sexy women to advertise software products. Others find it inappropriate. In Microsoft's earlier days, Redmond employees might put up a subsidiary's product poster that brought Human Resources running down the hallway. Use of sports analogies can also be problematic. What Americans think of as "football" isn't how the rest of the world thinks of it.

Any marketing messaging or imagery has the potential for offending someone somewhere in the world. You will lower the odds, however, if you check with people who are authorities on the markets you plan to enter. If you don't have the resources to hire an international advertising agency, you can at least do some searches on the Internet. We submitted a question about how to market software to Koreans to a blog called "Ask a Korean." If you can find other good Internet resources, please post them in the course Wiki. If you have good advice to give, please post that too.