Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

А.В. Терещенко

ENGLISH FOR PROGRAMMERS. PART II

Учебное пособие
для магистрантов ФСУ, ФВС

Томск
2025

УДК 811.111:004(075.8)
ББК 81.432.1я73
Т 350

Рецензент:

Шилина Елена Николаевна,

кандидат педагогических наук, доцент кафедры иностранных языков ФГАОУ ВО «Томский государственный университет систем управления и радиоэлектроники»

А.В. Терещенко

English for Programmers. Part II: учебное пособие для магистрантов ФСУ, ФВС / А.В. Терещенко. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2025. – 81 с.

Вторая часть учебного пособия “English for Programmers” («Английский язык для программистов») предназначена для магистрантов факультета систем управления (ФСУ) и факультета вычислительных систем (ФВС). В пособии представлено четыре раздела: “Web Development” (Веб-разработка), “Application Development. Mobile Application Development” (Разработка приложений. Разработка мобильных приложений), “Game Development” (Разработка игр), “Software Development” (Разработка программного обеспечения). Структура каждого раздела следующая: два текста для чтения (базового и повышенного уровня сложности); подраздел Language Development, который содержит комплекс лексико-грамматических упражнении; подраздел Improve your translation skills, включающий задания на совершенствование умений в переводе профессионально-ориентированных текстов с русского языка на английский. Кроме того, каждый раздел снабжен глоссарием, систематизированным в виде таблиц с пояснениями и вариантами перевода. Подраздел “Focus on Writing” содержит теоретический материал и практические упражнения, затрагивающие вопросы оформления внутритекстового цитирования, использования синонимов, свойственных научному стилю речи.

Одобрено на заседании кафедры иностранных языков (ИЯ), протокол № 7 от 14.10.2025 года

УДК 811.111:004(075.8)
ББК 81.432.1я73

CONTENTS

ВВЕДЕНИЕ

Вторая часть пособия предназначена для магистрантов ФСУ и ФВС, обучающихся по направлениям подготовки *«Программная инженерия», «Информатика и вычислительная техника», «Прикладная информатика».*

В современной IT-индустрии знание английского языка является неотъемлемой частью эффективной коммуникации и международного сотрудничества. Быстрый темп развития технологий бросает новые вызовы IT-специалистам.

Сфера IT развивается динамично, поэтому в условиях конкуренции и обновляющихся требований к работе преимущества будет иметь лишь тот, кто ранее всего изучит новую информацию, которая зачастую представлена в иноязычном сегменте.

Если в первой части пособия рассматривались более общие темы, такие как *"Computers"* («Компьютеры»), *"Programming Languages and paradigms"* («Языки программирования и парадигмы программирования»), то во второй части пособия внимание уделено профессионально-ориентированным темам, в частности, речь ведется о разработке:

- Unit 1 *"Web Development"* («Веб-разработка»);
- Unit 2 *"Application Development. Mobile Application Development"* («Разработка приложений. Разработка мобильных приложений"»;
- Unit 3 *"Game Development"* («Разработка игр»);
- Unit 4 *"Software Development"* («Разработка программного обеспечения»).

Каждый раздел пособия содержит два текста для чтения и глоссарий. Первый текст следует использовать для аудиторной работы; второй текст, в зависимости от объема учебного времени, которое отводится на изучение дисциплины, можно рекомендовать для самостоятельного изучения.

Глоссарий перед текстом и в конце каждого раздела (систематизированный в алфавитном порядке) содержит различные варианты перевода, комментарии, примечания. Лексика, представленная в глоссарии, имеет профессиональную направленность и полезна для изучения. Лексические единицы, представленные в глоссарии, можно интегрировать в элементы контроля, например, словарные диктанты. На основе изученной лексики магистранты могут совершенствовать умения в переводе с русского языка на английский, на что направлен отдельный подраздел в пособии *(Improve your translation skills).*

В пособие включены задания, направленные на поиск информации в Интернете, например, задание на заполнение таблицы на основе найденной информации с последующей презентацией доклада в группе.

В *«Приложении»* представлен дополнительный материал и упражнения по достаточно актуальной теме – *"Software Developer Soft Skills"* («Гибкие навыки, необходимые для разработчика программного обеспечения»). Материал можно интегрировать в процесс изучения четвертого раздела пособия (Unit 4 *"Software Development"*, «Разработка программного обеспечения»). Также в *«Приложении»* имеются задания по тематике "Emotional Intelligence" («Эмоциональный интеллект»).

Подраздел *“Focus on Writing”* освещает особенности использования синонимов в текстах академической направленности, нюансы употребления числовой информации, таблиц, процентов, графиков. Упражнения, содержащиеся в подразделе *“Focus on Writing”,* предполагают использование словарей, справочников, что способствует формированию важного навыка работы с большим объемом информации в англоязычном сегменте Интернета.

Высокий уровень владения английским языком значительно расширяет профессиональные горизонты программиста и способствует более эффективной работе в глобальной IT-среде. Автор выражает надежду, что разнообразные темы профессиональной направленности, изложенные в данном пособии, получат положительный отклик у обучающихся и будут способствовать их активному участию в учебном процессе.

Успехов в получении новых знаний!

UNIT 1

WEB DEVELOPMENT

LEAD-IN

1. What are the key skills required for a career in web development?
2. Which programming languages are commonly used for creating dynamic websites?
3. What is the purpose of responsive design in web development?
4. How does a website server communicate with a user's browser?

READING I

build website, *expr.*	создавать сайт
private network, *expr.*	частная сеть
concern, *v.*	иметь отношение к чему-либо, относиться к чему-либо
power, *v.*	приводить в действие, обслуживать *(context)*
website functionality, *expr.*	функциональные возможности веб-сайта
static web page, *expr.*	статическая веб-страница
e-commerce, *n.*	электронный бизнес, заключение сделок при помощи сети Интернет
Content Management System (CMS)	система управления контентом, система управления информационным наполнением
on a daily basis, *expr.*	на ежедневной основе, каждодневно, ежедневно
range, *v.*	варьироваться, изменяться в некоторых пределах *(context)*
refer, *v.*	обращаться, приписывать, ссылаться, подразумевать *(context)*
web engineering, *expr.*	веб-инжиниринг *(проф.)*
web design, *expr.*	дизайн сайтов, веб-дизайн *(проф.)*
web content development, *expr.*	разработка цифрового контента

client liaison, *expr.*	поддержание связей с клиентами
client-side scripting, *expr.*	разработка скриптов для выполнения на стороне клиента
server-side scripting, *expr.*	выполнение скриптов на стороне сервера
network security configuration, *expr.*	конфигурация сетевой безопасности
web developer, *expr.*	веб-разработчик, разработчик веб-страниц, разработчик веб-сайтов
agile methodology, *expr.*	гибкая методология программирования; методология программирования, рассчитанная на ситуацию, когда требования в системе постоянно меняются
collaborative, *adj.*	совместный, объединенный, коллективный *(context)*
designated department, *expr.*	определенный отдел, определенная отрасль *(context)*
execute, *v.*	реализовывать, исполнять, осуществлять, выполнять
layout, *n.*	формат, макет, диспозиция, раскладка *(context)*
font, *n.*	шрифт
run smoothly, *expr.*	работать безотказно
request, *n.*	запрос
database technology, *n.*	технология баз данных
retrieve, *v.*	исправить ошибку, вернуть, возвращать *(context)*
edit, *v.*	редактировать
relational database management system (RDBMS)	реляционная система управления базами данных, СУБД на основе реляционной модели данных
interchangeably, *adv.*	взамен, заменяя друг друга
gear, *n.*	механизм *(context)*
dashboard, *n.*	приборная панель
drop-down menu, *expr.*	выпадающее меню
scrollbar, *n.*	полоса прокрутки
responsive design, *expr.*	адаптивный дизайн, гибкий дизайн, отзывчивый дизайн

Web Development

Web development is the process of building websites and applications for the Internet, or for a private network known as an intranet. Web development is not concerned with the design of a website; rather, it's all about the coding and programming that powers the website functionality.

From the most simple, static web pages to social media platforms and apps, from e-commerce websites to content management systems (CMS) – all the tools we use via the Internet on a daily basis have been built by developers.

Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers, may include web engineering, web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.

For larger organizations and businesses, web development teams can consist of hundreds of people (web developers) and follow standard methods like Agile methodologies while developing web sites. Web development may be a collaborative effort between departments rather than the domain of a designated department.

There are three kinds of web developer specialization: front-end developer, back-end developer, and full-stack developer. *Front-end developers* are responsible for behavior and visuals that run in the user browser, while *back-end developers* deal with the servers.

Types of web development

Web development can be broken down into three layers: client-side coding (frontend), server-side coding (backend) and database technology.

Client-side

Client-side scripting, or frontend development, refers to everything that the end user experiences directly. Client-side code executes in a web browser and directly relates to what people see when they visit a website. Things like layout, fonts, colours, menus and contact forms are all driven by the frontend.

Server-side

Server-side scripting, or backend development, is all about what goes on behind the scenes. The backend is essentially the part of a website that the user doesn't actually see. It is responsible for storing and organizing data, and ensuring that everything on the client-side runs smoothly. It does this by communicating with the frontend. Whenever something happens on the client-side – say, a user fills out a form – the browser sends a request to the server-side. The server-side "responds" with relevant information in the form of frontend code that the browser can then interpret and display.

Database technology

Websites also rely on database technology. The database contains all the files and content that are necessary for a website to function, storing it in such a way that makes it easy to retrieve, organize, edit, and save. The database runs on a server, and most websites typically use some form of relational database management system (RDBMS).

The difference between web development and web design

Just like with software engineering, you might also hear the terms "web development" and "web design" used interchangeably, but these are two very different things.

Imagine a web designer and web developer working together to build a car: the developer would take care of all the functional components, like the engine, the wheels and the gears, while the designer would be responsible for both the visual aspects - how the car looks, the layout of the dashboard, the design of the seats - and for the user experience provided by the car, so whether or not it's a smooth drive.

Web designers design how the website looks and feels. They model the layout of the website, making sure it's logical, user-friendly and pleasant to use. They consider all the different visual elements: what color schemes and fonts will be used? What buttons, drop-down menus and scrollbars should be included, and where? Web design also considers the information architecture of the website, establishing what content will be included and where it should be placed.

Web design is an extremely broad field and will often be broken down into more specific roles such as user experience design, user interface design, and information architecture.

It is the web developer's job to take this design and develop it into a live, fully functional website. A frontend developer takes the visual design as provided by the web designer and builds it using coding languages such as HTML, CSS and JavaScript. A backend developer builds the more advanced functionality of the site, such as the checkout function on an e-commerce site. In short, a web designer is the architect, while the web developer is the builder or engineer.

Table 1. Job responsibilities of frontend developer, backend developer and full stack developer.

FRONTEND DEVELOPER	BACKEND DEVELOPER	FULL STACK DEVELOPER
1. Code the fronted of a website, i.e. the part that the user sees and interacts with. 2. Bring the web designer's designs to life using HTML, JavaScript and CSS. 3. Ensure responsive design.	1. Work behind-the-scenes, building and maintaining the technology needed to power the frontend. 2. Ensure that everything the frontend developer builds is fully functional. 3. Create and manage the database.	1. Expert in both frontend and backend development. 2. Guide on strategy and best practices. 3. Well-versed in both business logic and user experience.

LANGUAGE DEVELOPMENT

Exercise 1.1. Give English equivalents to the following words and word combinations.

Создавать вебсайт, статическая веб-страница, инструмент, сложные веб-приложения, перечень задач, исполнение скриптов на сервере, поддержание связей с клиентами, совместные усилия, ответственный отдел, разработчик пользовательских интерфейсов, разработчик полного цикла, шрифтовой комплект, посылать запрос, извлекать, взаимозаменяемо, раскрывать падающее меню, выполняться на сервере, полагаться на что-л., код клиентской стороны.

Exercise 1.2. Read the text and decide if the following sentences are true or false.

1. Web development is the process of building websites and applications for a private network.
2. Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services.
3. There are two kinds of web developer specialization: front-end developer and back-end developer.
4. The backend is not very actual because the user doesn't actually see it.
5. Web design is an extremely broad field.
6. Backend developers ensure that everything the frontend developer builds is fully functional.
7. A full-stack developer builds the more advanced functionality of the site, such as the checkout function on an e-commerce site.
8. Sometimes websites do not rely on database technology.

Exercise 1.3. Answer the following questions using information from the text.

1. What does a full-stack developer do?
2. What is a backend developer responsible for?
3. What does a frontend developer do?
4. Can you name any types of web development?
5. What is the difference between web development and web design?
6. How many people can web development team consist of?
7. What tasks does web development include?

Exercise 1.4. Match the word from the text with its synonym.

1. to save	a. of the sight or vision
2. complex	b. outline
3. to execute	c. to prepare, to supply
4. design	d. complicated, composed of several elements
5. foundation	e. constituent
6. to provide	f. basis
7. to designate	g. to appoint, to assign
8. component	h. to keep, to preserve
9. visual	i. project, scheme
10. configuration	j. to accomplish

Exercise 1.5. Match the word with its definition.

Word	*Definition*
1. backend	a. a collection of interlinked web pages on the World Wide Web
1. domain	b. all of the behind-the-scenes digital operations that it takes to keep the front end of a website running, such as the coding, style, and plugins
2. web designer	c. the address for a website as entered into the browser
3. website	d. the part of the website or app that the user sees. If the back end of your website is everything behind-the-scenes, this is what happens onstage
4. frontend	e. system software for creating and managing databases that makes it possible for end users to create, protect, read, update and delete data in a database
5. database management system (DBMS)	f. an IT professional who is responsible for designing the layout, visual appearance and the usability of a website

READING II

web publishing, *expr.*	веб-публикация
maintain, *v.*	содержать, поддерживать, сохранять, соблюдать *(context)*
trade-off, *n.*	взаимная уступка, компромисс
customization, *n.*	персонализация, изготовление изделия по техническим условиям заказчика
template, *n.*	шаблон, образец
editing tool, *expr.*	средство редактирования
incorporate, *v.*	объединять, включать, включать в состав
encompass, *v.*	включать в себя, охватывать
subset, *n.*	подгруппа

markup language, *expr.*	язык гипертекстовой разметки
Cascading Style Sheets (CSS)	каскадно расположенные стилевые листы
from scratch, *expr.*	с нуля
WYSIWYG	What You See Is What You Get
overlap, *v.*	перекрывать, частично совпадать, накладываться *(context)*
host, *n.*	хост-компьютер
reside, *v.*	храниться *(context)*
whip up a storyboard, *expr.*	создать раскадровку *(context)*
wireframe, *n.*	режим для просмотра рисунка, объекты которого представляются только их контурами
mock-up, *n.*	эскиз, прототип *(context)*
pitch an idea to the team, *expr.*	презентовать идею команде, подать идею команде
fine tune, *expr.*	настраивать, регулировать *(context)*
viable, *adj.*	жизнеспособный
grasp of skills, *expr.*	понимание навыков
feasible, *adj.*	выполнимый, осуществимый, возможный, вероятный *(context)*

Web Design and Web Development

Web development refers to building, creating, and maintaining websites. It includes aspects such as *web design,* web publishing, web programming, and database management. While the terms "web developer" and "web designer" are often used synonymously, they do not mean the same thing. Technically, a web designer only designs website interfaces using HTML and CSS. A web developer may be involved in designing a *website,* but may also write web scripts in languages such as PHP and ASP.

Additionally, a web developer may help maintain and update a database used by a dynamic website. Web development includes many types of web content creation. Some examples include hand coding web pages in a text editor, building a website in a program like Dreamweaver, and updating a blog via a blogging website. In recent years, content management systems like WordPress, Drupal, and Joomla have also become popular means of web development. These tools make it easy for anyone to create and edit their own website using a web-based interface.

While there are several methods of creating websites, there is often a trade-off between simplicity and customization. Therefore, most large businesses do not use content management systems, but instead have a dedicated web development team that designs and maintains the company's website(s). Small organizations and individuals are more likely to choose a solution like WordPress that provides a basic website *template* and simplified editing tools. JavaScript programming is a type of web development that is generally not considered part of web design. However, a web designer may reference JavaScript libraries like jQuery to incorporate dynamic elements into a site design. Web design is the process of creating websites. It encompasses several different aspects, including *webpage layout,* content production, and graphic design. While the terms web design and web development are often used interchangeably, web design is technically a subset of the broader category of web development.

Websites are created using a markup language called HTML. Web designers build webpages using HTML tags that define the content and metadata of each page. The layout and appearance of the elements within a webpage are typically defined using CSS, or cascading style sheets. Therefore, most websites include a combination of HTML and CSS that defines how each page will appear in a browser. Some web designers prefer to hand code pages (typing HTML and CSS from scratch), while others use a "WYSIWYG" editor like Adobe Dreamweaver. This type of editor provides a visual interface for designing the webpage layout and the software automatically generates the corresponding HTML and CSS code. Another popular way to design websites is with a content management system like WordPress or Joomla.

These services provide different website templates that can be used as a starting point for a new website. Webmasters can then add content and customize the layout using a web-based interface. While HTML and CSS are used to design the look and feel of a website, images must be created separately. Therefore, graphic design may overlap with web design, since graphic designers often create images for use on the Web. Some graphics programs like Adobe Photoshop even include a "Save for Web" option that provides an easy way to export images in a format optimized for web publishing. Web publishing, or "online publishing," is the process of publishing content on the Internet. It includes creating and uploading websites, updating webpages, and posting blogs online. The published content may include text, images, videos, and other types of media. In order to publish content on the web, you need three things: 1) web development software, 2) an Internet connection, and 3) a *web server.* The software may be a professional web design program like Dreamweaver or a simple web-based interface like WordPress.

The Internet connection serves as the medium for uploading the content to the web server. Large sites may use a dedicated web *host,* but many smaller sites often reside on shared servers, which host multiple websites. Most blogs are published on public web servers through a free service like Blogger. Since web publishing doesn't require physical materials such as paper and ink, it costs almost nothing to publish content on the web.

Therefore, anyone with the three requirements above can be a web publisher. Additionally, the audience is limitless since content posted on the web can be viewed by anyone in the world with an Internet connection. These advantages of web publishing have led to a new era of personal publishing that was not possible before.

What is the difference between a web designer and a web developer?

In the early days of the web, the answer to that question was simple: designers design and developers code. Today that question requires a little more nuance – you'd be hard pressed to find a web designer who didn't know at least a little HTML and CSS, and you won't have to look far for a frontend web developer who can whip up a storyboard.

If you're strictly speaking about the general concepts of web design vs. web development, however, the distinction is a little more clear. Let's take a look at these two concepts and the roles they play in building the websites and apps we know and love. What is web design? Web design governs everything involved with the visual aesthetics and usability of a website – color scheme, layout, *information flow,* and everything else related to the visual aspects of the UI/UX (user interface and user experience).

Some common skills and tools that distinguish the web designer from the web developer are:

- Adobe Creative Suite (Photoshop, Illustrator) or other design software;
- Graphic design;
- Logo design;
- Layout/format;
- Placing call-to-action buttons;
- Branding;
- Wireframes, mock-ups, and storyboards;
- Color palettes;
- Typography.

Web design is concerned with what the user actually sees on their computer screen or mobile device, and less so about the mechanisms beneath the surface that make it all work. Through the use of color, images, typography and layout, they bring a digital experience to life. That said, many web designers are also familiar with HTML, CSS, and JavaScript – it helps to be able to create living mock-ups of a web app when trying to pitch an idea to the team or fine-tune the UI/UX of an app. Web designers also often work with templating services like WordPress or Joomla, which allow you to create websites using themes and widgets without writing a single line of code.

What is web development?

Web development governs all the code that makes a website tick. It can be split into two categories – frontend and backend. The frontend or client-side of an application is the code responsible for determining how the website will actually display the designs mocked up by a designer. The backend or server-side of an application is responsible for managing data within the database and serving that data to the frontend to be displayed.

As you may have guessed, it's the frontend developer's job that tends to share the most overlap with the web designer. Some common skills and tools traditionally viewed as unique to the frontend developer are listed below:

- HTML/CSS/JavaScript;
- CSS preprocessors (i.e., LESS or Sass);

- Frameworks (i.e., AngularJS, ReactJS, Ember);
- Libraries (i.e., jQuery);
- Git and GitHub.

Frontend web developers don't usually create mock-ups, select typography, or pick color palettes – these are usually provided by the designer.

It's the developer's job to bring those mock-ups to life. That said, understanding what the designer wants requires some knowledge of best practices in UI/UX design, so that the developer is able to choose the right technology to deliver the desired look and feel and experience in the final product.

Meet the "unicorn"

What started out as a joke in the industry – the designer/developer hybrid who can do it all – is now a viable endgame for both web designers and front-end developers, thanks to the increase in availability of educational resources across the web. Those developers/designers who have a good grasp of skills across both sides of the spectrum are highly sought after in the industry.

The "unicorn" can take your project from the conceptual stage of visual mock-ups and storyboards, and carry it through frontend development all by themselves. Not that you'd want them to; the real value of developers who design and designers who develop is their ability to speak each other's languages. This leads not only to better communication on the team and a smoother workflow, it means you'll land on the best solution possible. As a general rule, feel free to rely on the "unicorn" for small projects, where it's *feasible* for one or two people to handle both the back and frontends of an application. For larger projects, even if you do manage to hire a few "unicorns," more clearly defined roles are required.

LANGUAGE DEVELOPMENT

Exercise 1.6. Give the translation to the following words and words combinations presented in the table below. If necessary, consult with the dictionary.

English word or word combination	*Translation*
1. database managment	a.
2. blogging website	b.
3. web development team	c.
4. to dedicate	d.
5. digital experience	e.
6. typography and layout	f.
7. unique tools	g.

Exercise 1.7. Answer the questions using information from the text.

1. What are the main differences between frontend and backend web development?
2. Which programming languages are commonly used for creating websites?
3. How does responsive web design improve user experience on different devices?
4. What is the purpose of using CSS in web development?
5. Can you explain what a content management system (CMS) is and give some examples?
6. What are some popular tools or frameworks used for web development today?
7. Why is website accessibility important, and how can it be improved?

Exercise 1.8. Work with *the italicized* **words from the text. Give definitions to them using the dictionary. The first definition is presented as an example for you.**

The italicized word from the text	*Definition*
1. Web design	• the process of planning, creating, and arranging the visual and interactive elements of a website.
2. Website	•
3. Template	•
4. Webpage layout	•
5. Web server	•
6. Host	•
7. Information flow	•
8. Feasible	•

IMPROVE YOUR TRANSLATION SKILLS

Learn vocabulary from Unit 1 “WEB DEVELOPMENT” and translate the following sentences from Russian into English.

1. Веб-разработка включает в себя создание и поддержку сайтов и веб-приложений.
2. Веб-дизайн отвечает за внешний вид и удобство использования сайта.
3. Современные веб-сайты должны быть адаптивными и хорошо отображаться на всех устройствах.
4. HTML, CSS и JavaScript — основные технологии, используемые в веб-разработке.
5. Хороший веб-дизайн помогает привлечь и удержать посетителей сайта.
6. В процессе веб-разработки важно учитывать безопасность данных пользователей.
7. Использование фреймворков, таких как React или Angular, ускоряет создание сложных интерфейсов.
8. Веб-дизайнеры создают макеты и прототипы будущих сайтов.
9. Оптимизация скорости загрузки сайта важна для улучшения пользовательского опыта и поисковой оптимизации (SEO, Search Engine Optimization).
10. Постоянное обучение новым технологиям — залог успешной карьеры в области веб-разработки и дизайна.

FOCUS ON WRITING

Text features
Organization of the text

Before you read the information about text features and organization of the text, explain the following terms:

- Introduction, Main Body, Conclusion
- Abstract, Contents, Introduction, Main Body, Case Study, Discussion, Findings, Conclusion, Acknowledgements, References, Appendices
- Dedication, Foreword, Preface, Index

Where can you apply these terms?

Paragraph

A paragraph is a group of related sentences that discuss one main idea. A paragraph can be as short as one sentence or as long as ten sentences. The number of sentences is unimportant; however, the paragraph should be long enough to develop the main idea clearly.

A paragraph may stand by itself and may also be one part of a longer piece of writing such as an essay or a book. We mark a paragraph by indenting the first word about a half inch (five spaces on a typewriter or computer) from the left margin.

Writing Exercise 1. Read the following piece of writing that contains all the elements of a good paragraph.

We think of the Mona Lisa as a brilliant example of Renaissance art. First of all, the Mona Lisa is a mysterious image. Secondly, we think of it as a thoughtful study in composition, light, and shadow. Therefore, we don't think of it as a mathematical formula. However, in the computer world, all art, graphics, shapes, colors, and lines involve some type of mathematical algorithm. That statement isn't meant to belittle the works of Da Vinci and other great artists.

Mathematical algorithms cannot create art; that still takes a true artist, whether the artist's tools are brush, oils, and canvas or a computerized stylus. But math embedded in specific file formats can describe any piece of existing art. For example, a graphics-file image of the Mona Lisa that you can display on your PC is the result of mathematical calculations on the bytes of data saved in that file. In conclusion, all the capabilities of a darkroom and an artist's studio are available on a personal computer.

Writing Exercise 2. Answer the following questions. These questions will help you to analyze the structure of the paragraph.

1. What is the topic of the paragraph?
2. What two main points does the writer make about the topic?
3. In which two sentences does the writer say that there are two main points?

4. What examples does the writer use to support his idea?

All paragraphs have *a topic sentence* and *supporting sentences*, and some paragraphs also have a *concluding sentence. The topic sentence* states the main idea of the paragraph, limits the topic to one specific area that can be discussed completely in the space of a single paragraph. The part of the topic sentence that announces the specific area to be discussed is called the *controlling idea.* The topic sentence is often, but not always, the first.

Supporting sentences explain, prove or develop the topic sentence. That is, they explain or prove the topic sentence by giving more information about it.

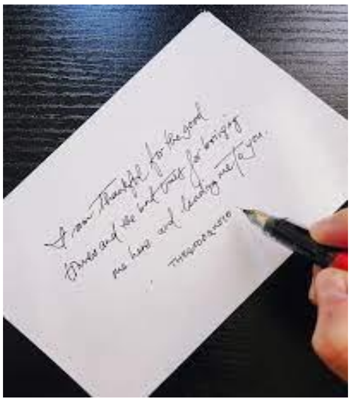
The concluding sentence signals the end of the paragraph and leaves the reader with important points to remember. You can do this by summarizing the main points of the paragraph or by repeating the topic sentence in different words. Concluding sentences are customary for stand-alone paragraphs. However, a concluding sentence is not needed for every paragraph in a multiparagraph essay.

Writing Exercise 3. Rewrite the sentences in the correct order so as to make a paragraph. Write your answers in the table below.

1. An algorithm is a fixed set of operations that change data in a way that makes the original document incomprehensible.
2. The key to decrypting it would be "shift one letter to left."
3. A person who wants to send another person a confidential document encrypts the file using the public key as a variable in the algorithm used by the software.
4. A simple example of an algorithm is "shift one letter to the right," so that HAL becomes IBM.

Element of the structure	*Number of sentence*
Topic	
Detail	
Example	
Reason	

Introduction



Introduction is an important part of your work. Unless you can introduce the subject clearly, the reader may not wish to continue. You must show *the importance of the topic.* This can be either in the academic word or as a contemporary issue of wider relevance. There is no such thing as a standard introduction, and much depends on *the nature of the research* and the length of the essay. However, for a relatively short essay the following are worth including, in this order:

a. Definitions of any terms in the title that are unclear;

b. Some background information;

c. Reference to other writers who have discussed this topic;

d. Your purpose in writing and the importance of the subject;

e. The points you are going to make/areas you are going to cover.

Writing Exercise 4. Write an introduction (it should be about 100 words) to an essay on a subject from your own discipline.

Main Body

The structure of the main body depends on the length of the essay and the subject of study. Longer essays may include the following sections:

- *Experimental set-up* – a technical description of the organization of an experiment;
- *Methods* – how the research was carried out;
- *Findings/results* – what was discovered by the research/experiment;
- *Case study* – a description of an example of the topic being researched;
- *Discussion* – an examination of the issues and the writer's verdict.

Inside the main body, ideas need to be presented in the most logical fashion, linked together to form a coherent argument. It is useful to mark the beginning of new paragraphs or the introduction of new subjects with special phrases.

Table 2. Useful phrases.

Useful phrases to introduce a new paragraph/topic	*Useful phrases to add information inside a paragraph*
The main/chief factor/issue is ... Turning to the subject of … Moving on to the question of … Another important area is ... … must also be examined Even if … It is wise to think …	a. Firstly, ... The first point ... In the first place ... b. Secondly, ... Next, ... Then, ... In addition … c. Moreover …, On the other hand,… Also, … d. Finally, ... Lastly, …

Conclusion

There is usually a link between the starting point, i.e. the title, and the conclusion. If the title is asking a question, the conclusion should contain the answer. The reader may look at the conclusion first to get a quick idea of the main arguments or points. In most cases it is helpful for the reader to have a section that (quite briefly) looks back at what has been said and makes some comments about the main part.

Conclusion paragraphs are about 5% of your essay word count (e.g. about 50 or so words per 1000 word essay). In clearly-written sentences, you restate the thesis from your introduction (but do not repeat the introduction too closely), make a brief summary of your evidence and finish with some sort of judgment about the topic.

It's a good idea to start your conclusion with *transitional words* (e.g. 'In summary', 'To conclude', 'In conclusion', 'Finally',) to help you to get the feel of wrapping up what you have said.

The conclusion is not the place to present new facts (should be in the body of your essay), so conclusions don't usually have references unless you come up with a 'punchy' quote from someone special as a final word.

Writing Exercise 5. Arrange in the correct order (1-5) the following ideas that may be written in conclusion section. Write your answer in the right column of the table.

Idea	*Number (1-5)*
Implications of the findings	
Proposals for further research	
Limitations of the research	
Reference to how these findings compare with other studies	
Summary of main findings	

VOCABULARY LIST OF UNIT 1 “WEB DEVELOPMENT”

agile methodology, *expr.*
build website, *expr.*
Cascading Style Sheets (CSS)
client liaison, *expr.*
client-side scripting, *expr.*
collaborative, *adj.*
concern, *v.*
Content Management System (CMS)
customization, *n.*
dashboard, *n.*
database technology, *n.*
designated department, *expr.*
drop-down menu, *expr.*
e-commerce, *n.*
edit, *v.*

editing tool, *expr.*
encompass, *v.*
execute, *v.*
feasible, *adj.*
fine tune, *expr.*
font, *n.*
from scratch, *expr.*
gear, *n.*
grasp of skills, *expr.*
host, *n.*
incorporate, *v.*
interchangeably, *adv.*
layout, *n.*
maintain, *v.*
markup language, *expr.*
mock-up, *n.*
network security configuration, *expr.*
on a daily basis, *expr.*
overlap, *v.*
pitch an idea to the team, *expr.*
power, *v.*
private network, *expr.*
range, *v.*
refer, *v.*
relational database management system (RDBMS)

request, *n.*
reside, *v.*
responsive design, *expr.*
retrieve, *v.*
run smoothly, *expr.*
scrollbar, *n.*
server-side scripting, *expr.*
static web page, *expr.*
subset, *n.*
template, *n.*
trade-off, *n.*
viable, *adj.*
web content development, *expr.*
web design, *expr.*
web developer, *expr.*
web engineering, *expr.*
web publishing, *expr.*
website functionality, *expr.*
whip up a storyboard, *expr.*
wireframe, *n.*
WYSIWYG (What You See Is What You Get)

UNIT 2

APPLICATION DEVELOPMENT. MOBILE APPLICATION DEVELOPMENT

LEAD-IN

1. What are the main stages involved in the application development process?
2. Which programming languages are commonly used for mobile application development?
3. How does user experience influence the success of an application?
4. What is the difference between native and cross-platform application development?
5. What tools and frameworks are popular for building web applications?

Stages of Application Development Life Cycle

1. The picture below shows 7 stages of application development life cycle (ADLC).
2. Speak on the topic using words and expressions from Table 3.

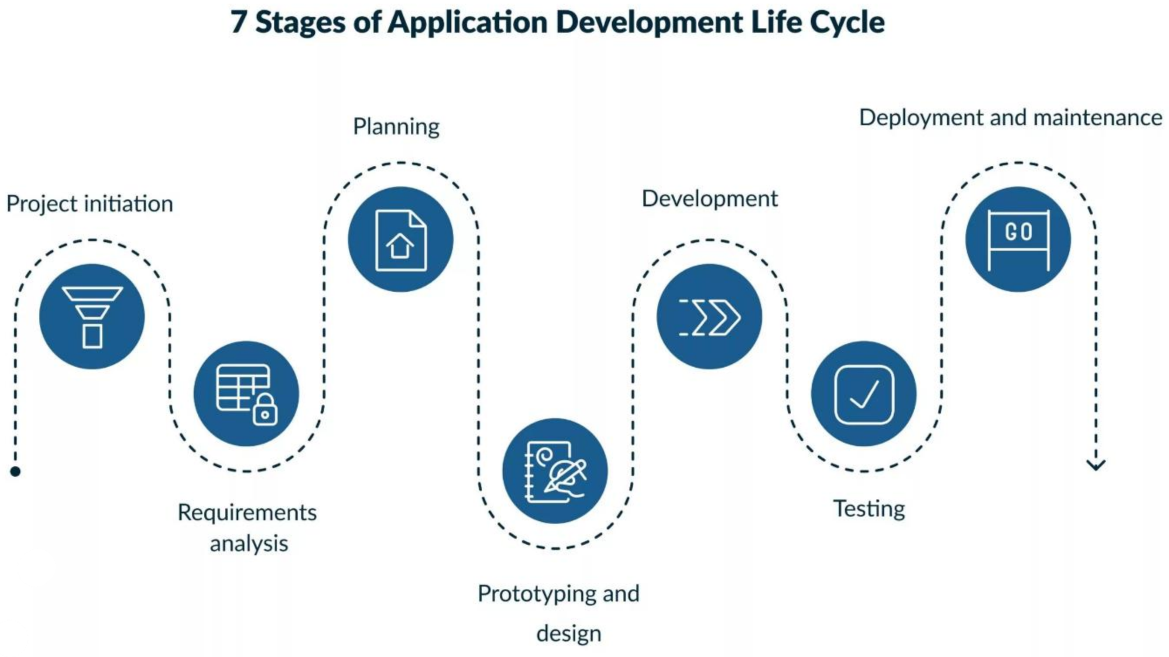


Table 3. Useful expressions.

Ideation and conceptualization
Requirements gathering
Discovery phase
To delve deeper into the target audience
Requirements specification and refinement
Alignment and adaptation
To gather user needs
To define functional requirements
To specify non-functional requirements
Prioritization and trade-offs
To address potential risks
To gather and analyze data
UI/UX design
Testing and quality assurance
Final testing and QA
Bug fixing
Enhancements

READING I

accomplish, *v.*	завершать, заканчивать, осуществлять, выполнять *(context)*
emerge, *v.*	появляться, возникать, выходить *(context)*
waterfall methodology, *expr.*	водопадный тип процесса разработки, каскадный процесс
sequence, *n.*	порядок, расположение, упорядоченность действий
map out, *ph.v.*	детально планировать
explicit, *adj.*	ясный, четкий, недвусмысленный, однозначно выраженный *(context)*
downward, *adv.*	вниз, в порядке убывания, в порядке уменьшения
line out, *ph.v.*	набросать общие очертания
specification, *n.*	техническая характеристика, спецификация, деталь, уточнение *(context)*
implement, *v.*	внедрить, осуществить, выполнить, привести в исполнение *(context)*
assume, *v.*	предполагать, допускать

unified vision, *expr.*	общее видение проекта *(context)*
clear vision, *expr.*	ясное, четкое видение проекта *(context)*
meticulous, *adj.*	тщательный, основательный
Rapid Application Development (RAD) Methodology, *expr.*	метод ускоренной разработки приложений
iterate, *v.*	повторять, возобновлять
divert, *v.*	отклоняться, перенаправлять *(context)*
cutover, *n.*	внедрение
complexity, *n.*	сложность
adherence to deadline, *expr.*	точное соблюдение дедлайнов
agile methodology, *expr.*	метод гибкой разработки
sign off, *ph.v.*	выходить (из системы)
sprint, *n.*	спринт (2-4 week increment of software development activities)
squad, *n.*	группа, команда
technical writer, *expr.*	технический писатель

Application Development Methodologies

Application development is the process of designing, building, and implementing software applications. It can be done by massive organizations with large teams working on projects, or by a single freelance developer. Application development defines the process of how the application is made and generally follows a standard methodology.

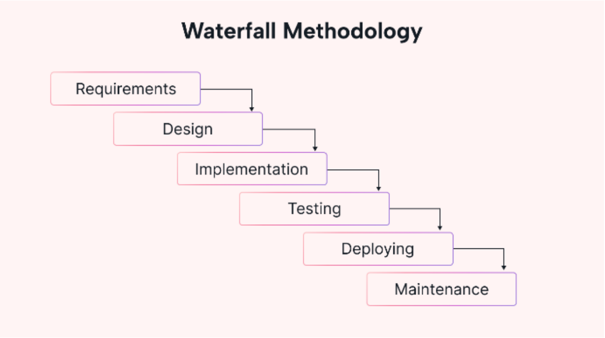
You must consider the size of the project, how specific the requirements are, how much the customer will want to change things, how large the development team is, how experienced the development team is, and the deadline for the project.

Application development is closely linked with the software development life cycle (SDLC). The basic stages of SDLC are *Planning, Analysis, Design, Construction, Testing, Implementation, Support.* The way that application development teams have accomplished these seven tasks has changed a lot in the last few decades, and numerous types of application development methods have emerged. Each methodology must provide a solution for the seven stages of the SDLC.

Most application development methodologies can be grouped into one of three categories: *Waterfall, RAD, Agile.*

Waterfall

The key words for the waterfall method of application development are planning and sequence. The entire project is mapped out in the planning and analysis stages. The customer comes with a very explicit list of features and functionalities for the application. Then, a project manager takes the whole process and maps it out amongst the team.



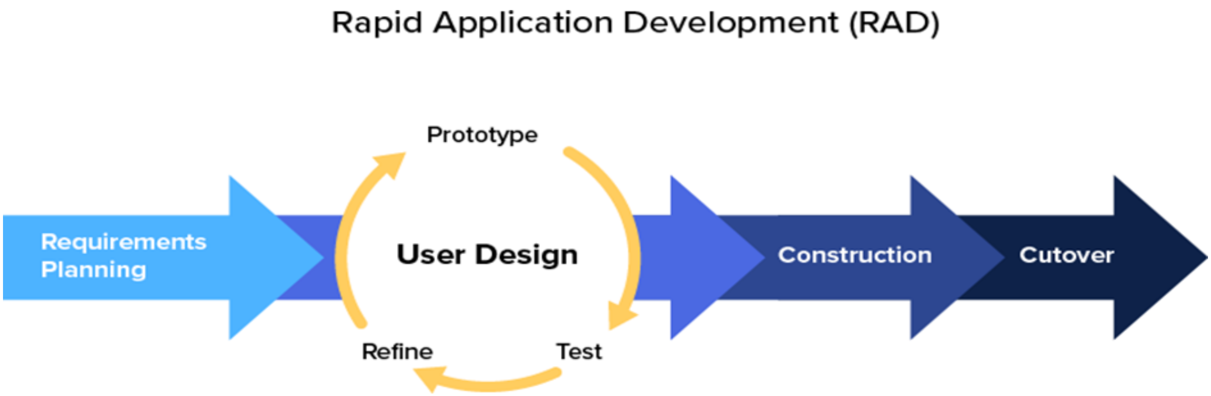
This application development method is called waterfall because once you go down, you can't go back up; everything flows downward. The development team works together over a set of time, building exactly what is lined out according to the specifications. After the architecture is designed, then only the construction can begin. The entire application is built, and then it is all tested to make sure that it is working properly. Then, it is shown to the customer and ready to be implemented.

The waterfall method assumes that the project requirements are clear, and the customer and project manager have a unified and clear vision about the end result. The advantage of the waterfall method is that it is very meticulous. It's also a good application development method to use for big projects that need to have one unifying vision. The waterfall method is also a good way to train junior programmers on parts of development without having to turn an entire project to them.

The disadvantages are that changes happen all the time. Even if the development team is able to build exactly what the customer originally wanted (which doesn't always happen), the market, technology, or the organization may have changed so much that it is effectively useless and a waste of time.

Rapid Application Development (RAD) Methodology

In many ways, RAD was the opposite of the waterfall method. RAD is based mostly on prototypes, meaning that the goal is to produce a working version of the application as quickly as possible, and then to continuously iterate after that. The application development team and the customer work very closely with each other throughout the process. RAD teams are usually small and only involve experienced developers who are skilled in many disciplines. If a project needs to divert from the original plan, RAD should be able to accommodate that easily.



In the RAD model, as each iteration is completed, the product gets more and more refined. The early prototypes are often very rough but give a picture of what can be. Each iteration then looks more like the finished product.

RAD advantages are a quick and highly flexible team and a very close relationship with the customer. If changes are expected, RAD will be able to accommodate these much faster than waterfall. RAD is also never too attached to a prototype and is always willing to change it to suit the needs of the customer.

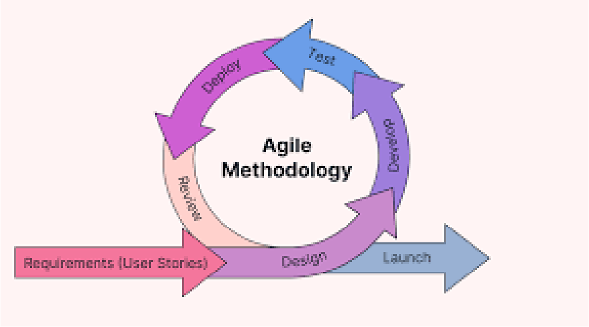
However, RAD isn't a perfect application development method. RAD requires highly skilled (and highly paid) programmers to work on a project that may change in complexity by the day. There's also less adherence to deadlines and more of a focus on adding features which can extend delivery dates. RAD requires a lot of input from customers who may not always be available or know what they need. Additionally, for some applications, having a prototype is not useful without seeing the entire product.

Agile Methodology

Agile application development is very similar to RAD but also includes some changes to make it more suitable to larger projects. Agile is iterative, like RAD, but focuses on building features one at a time. Each feature is built in a methodical way in the team, but the customer is involved to see the features and sign off on them before the next feature is developed.

Agile uses sprints or set of time when a certain feature should be built, tested, and presented. It tries to incorporate the entire SDLC for a feature into each sprint. This, ideally, helps to stick to a planned schedule, but also allows for frequent reviews.

Agile doesn't focus on prototypes but only presents completed work after the sprint is over. So, while the customer is informed more often than waterfall, the customer only ever sees finished work, unlike RAD.



Agile project management methodology is also more team or squad based. With RAD, you are working directly with a programmer. With Agile, the application development team will also include testers, UX designers, technical writers, and many others.

LANGUAGE DEVELOPMENT

Exercise 2.1. Give English equivalents to the following words and word combinations.

Внедрение программных приложений; внештатный разработчик; требования; стандартная методология; выполнить задачу; относить в одну из категорий; намечаться; подробный перечень; функции приложения; менеджер проекта; в соответствии со спецификациями; определенный период времени; пустая трата времени; быстрая разработка приложений; рабочая версия; заказчик; отклониться; привязываться к прототипу; в соответствии с потребностями; ориентирована на группу; запланированный график; метод гибкой разработки приложений.

Exercise 2.2. Answer the following questions using information from the text.

1. How many stages does software development life cycle include?
2. Why is one of the application development methods called waterfall?
3. What is the goal of Rapid Application Development?
4. Is RAD based mostly on prototypes?
5. Is RAD a perfect application development method? Why?
6. What does Agile application development use? Does it focus on prototypes?

Exercise 2.3. Match the word from the text with its synonym.

1. complexity	a. detail, very careful
2. entire	b. fitly, suitably
3. to suit	c. smart, active, quick
4. to incorporate	d. to correspond, to match
5. properlt	e. full, complete
6. managment	f. to unite, combine, mix, consolidate
7. agile	g. administration, guidance
8. to assume	h. complication, entanglement
9. meticulous	i. to suppose

Exercise 2.4. Match the word or word combination from the text with its definition.

Word or word combination	*Definition*
1. RAD	a. a linear, sequential approach to the software development life cycle (SDLC) that is popular
2. software testing	b. a repeatable fixed time-box during which a "Done" product of the highest possible value is created
3. sprint	c. a form of Agile software development methodology that prioritizes rapid prototype releases and iterations
4. implement	d. a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest
5. prototype	e. to recognize and use an element of code or a programming resource that is written into the program

6. waterfall method	f. an original model, form or an instance that serves as a basis for other processes. In software technology, this term is a working example through which a new model or a new version of an existing product can be derived

Exercise 2.5. Complete the table using information from the text. The first sentence is given as an example.

Methodology	*Advantages*	*Disadvantages*
Waterfall methodology	*Meticulous, good for big projects.*	*Changes happen all the time.*
RAD methodology		
Agile methodology		

READING II

native app, *expr.*	исходное приложение, платформозависимое приложение, платформенно-ориентированное приложение
native shell, *expr.*	платформозависимая оболочка приложения
performance, *n.*	производительность
engagement rate, *expr.*	процент вовлеченности, коэффициент вовлеченности *(context)*
maintainability, *n.*	возможность сопровождения, удобство эксплуатации *(context)*
modify, *v.*	варьировать, менять, трансформировать
ongoing maintenance, *expr.*	текущее обслуживание
decipher, *v.*	расшифровывать, распознавать
resolution, *n.*	выполнение *(context)*
foster, *v.*	способствовать, обеспечивать *(context)*
performant, *adj.*	высокопроизводительный, эффективный
redundancy, *n.*	избыток, излишек
sustainability, *n.*	устойчивость, способность к длительной эксплуатации *(context)*

elevate, *v.*	повышать
concise, *adj.*	лаконичный, сжатый, краткий, точный *(context)*
strive for, *expr.*	добиваться чего-либо, стремиться к чему-либо
overly, *adv.*	слишком, предельно, чрезмерно
convoluted algorithm, *expr.*	сверточный алгоритм
codebase, n.	кодовая база, база исходного кода
spacing, *ger.*	расстановка с интервалами
modularity, *n.*	модульный принцип организации *(context)*
troubleshooting, *ger.*	поиск и устранение неисправностей
descriptive variable, *expr.*	дескриптивная переменная
eliminate, *v.*	исключать, осуществлять устранение
adhere, *v.*	следовать, соблюдать, придерживаться
leverage, *v.*	эффективно использовать, улучшать, оптимизировать
assumption, *n.*	предположение, допущение
dependency, *n.*	зависимость
modularization, *n.*	модульная организация программы, модульность
reusability, *n.*	возможность многократного использования
encapsulate, *v.*	формировать, инкапсулировать, оформлять модуль *(context)*
recurring, *part.*	повторяющийся, цикличный
scalability, *n.*	универсальность, масштабируемость
force-fit, *v.*	подогнать, приспособить *(фраза не имеет прямого аналога перевода в русском языке)*
robust, *adj.*	устойчивый
unit test, *expr.*	модульный тест, автономный тест, блочный тест *(context)*

refactor, *v.*	выполнить реорганизацию кода, переписать исходный код с целью улучшения внутренней структуры программного кода при сохранении его интерфейсов
pinpoint, *v.*	проследить, точно определить
memory allocation, *expr.*	распределение памяти
OTA	Over the Air, технология, позволяющая обновлять программное обеспечение устройств по беспроводной сети (через Wi-Fi или мобильную сеть)

What is Mobile Application Development?

Mobile application development involves designing and creating software applications for mobile devices like smartphones and tablets. Such an application can be downloaded from an app store, come preinstalled on the device, or be accessed using a mobile web browser. Generally, this type of software development involves using Java, C#, Swift, and HTML as programming and markup languages.

Key Approaches to Building a Mobile App

You can build a mobile app in various ways. The following are some of the popular approaches followed by mobile app developers:

1. *Native App*

It is a platform-specific app that can run only on a particular operating system. This means that mobile application companies would need to develop an app differently for Android and iOS when following this approach.

2. *Cross-Platform App*

This app is based on a single, unified code that can run on multiple platforms. So, after coding once, mobile app companies can make it work for Android and iOS. Since it is a platform-independent app, it is cheaper to build.

3. *Hybrid App*

A hybrid app is a blend of two apps: a web application and a native mobile app. When building this app, a mobile application development agency transforms a web application into a native shell. This app is faster and easier to build but may suffer performance issues.

4. *Progressive Web App*

This web application extends a user experience similar to a native app. Since this app runs on the web, there's no need to install a separate app for it. The plus points of Progressive Web apps are that they perform better, consume less data, and enhance engagement rates.

The Significance of Clean and Efficient Code in Mobile Application Development

Clean and efficient code offers a multitude of benefits for mobile application development:

- Enhanced Maintainability: Well-written code is easier to understand and modify, allowing developers to introduce new features or fix bugs efficiently. This is particularly crucial as apps evolve and require ongoing maintenance.

- Reduced Development Time: Clean code promotes better code comprehension, minimizing time spent deciphering complex structures. This leads to a faster development process and quicker time-to-market for your app.

- Improved Debugging: Clean code is easier to debug, as the logic is clear and well-organized. This translates to faster issue identification and resolution, ensuring a smooth user experience.

- Enhanced Team Collaboration: Clean code fosters better communication and collaboration within development teams. When code is easy to understand, developers can work together more effectively.

- Optimized Performance: Clean and efficient code typically produces a more performant app. By avoiding unnecessary redundancies and optimizing resource usage, your app runs smoother and delivers faster user experience.

By prioritizing clean code, developers lay the foundation for a successful mobile app, ensuring long-term sustainability and user satisfaction.

7 Tips for Writing Clean and Efficient Code for Your Mobile App

Writing clean and efficient code is the foundation for a successful mobile app. It ensures your app development runs smoothly, is easy to maintain, and delivers a delightful user experience. Here are seven key tips to elevate the quality and efficiency of your mobile app code:

1. *Prioritize Readability and Maintainability*

Writing clear and concise code is essential while maintaining consistency in formatting and simplifying complex functions. You must strive for simplicity and avoid overly complex logic structures and convoluted algorithms. Instead, you can use clear variable names that reflect their purpose (e.g., ‘userName’ instead of ‘u’). When necessary, enhance readability with meaningful comments that explain non-obvious logic but avoid over-commenting in well-structured code. Maintain a consistent style throughout your codebase for indentation, spacing, and formatting. This improves readability and makes navigating the code much easier for you and future developers. Don’t hesitate to break large chunks of code into smaller, well-defined functions with clear purposes. This promotes modularity, improves readability, and helps with troubleshooting.

2. *Know the Art of Naming Conventions*

Utilize descriptive variable and function names that accurately reflect their purpose. This eliminates the need for excessive commenting and instantly clarifies what each piece of code does. For example, ‘calculateTotalCost’ is much more understandable than ‘calcTotal.’ Adhere to a consistent naming convention throughout your entire app. This ensures predictability and makes the code easier to navigate for you and other developers.

3. *Leverage Code Comments Strategically*

Documenting non-obvious logic and design choices is essential. Use comments to explain complex algorithms or logic that might not be immediately clear. However, avoid over-commenting well-structured code that is already self-documenting.
Always clarify assumptions and dependencies. Document any assumptions made within the code or external dependencies it relies upon. This aids future developers in understanding the code context and potential limitations.

4. *Apply Modularization and Reusability*

Organize your code into well-defined modules that encapsulate specific functionalities. This promotes code reuse, reduces redundancy, and simplifies maintenance. Think of your code as building blocks – the more reusable blocks you have, the easier it is to construct new features.
Modern mobile development platforms for iPhone and Android apps offer vast libraries and frameworks that provide pre-built, well-tested code components. Utilize these resources to avoid reinventing the wheel. Focus your efforts on your app unique functionalities and use libraries for everyday tasks like networking or image loading.

5. *Use Effective Design Patterns*

Make it a practice to identify common programming problems and solutions. Design patterns offer established and proven solutions to recurring programming challenges. Learn and implement relevant design patterns to enhance code structure, maintainability, and scalability.
Don't force-fit a design pattern into your code. Carefully analyze the problem you're trying to solve and select the pattern that best addresses the specific challenge.

6. *Implement Unit Testing*

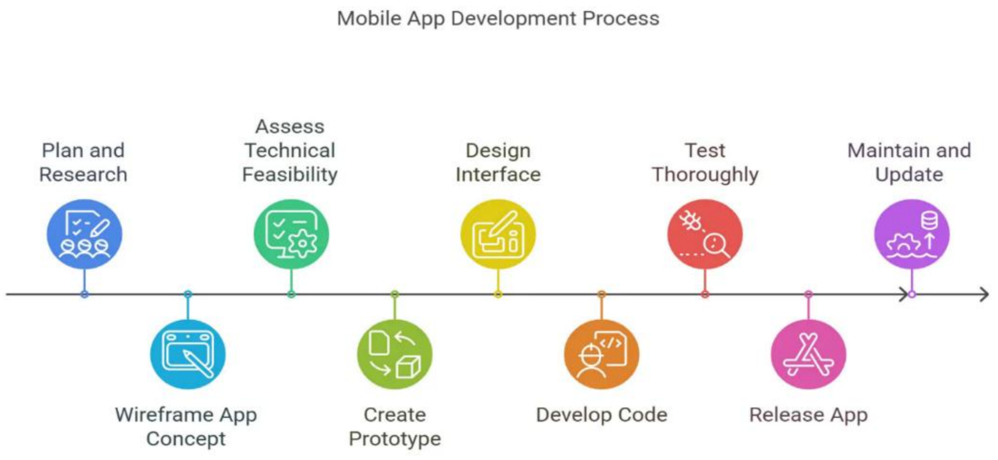
Unit testing involves creating automated tests for individual functions or modules within your code. It helps you isolate and verify each code unit functionality independently. Unit tests catch bugs early in the development cycle, leading to a more robust and reliable app.
It also helps identify bugs and potential issues early in the development cycle. This leads to a more robust and reliable app, minimizing crashes and unexpected behavior. With unit tests in place, you can refactor code with greater confidence, knowing that the underlying functionality remains intact. This promotes agility and simplifies future maintenance efforts. You can make changes to your codebase without worrying about breaking something else.

7. *Employ Performance Optimization Techniques*

Utilize profiling tools to identify areas of your code that consume excessive resources or slow down app performance. These tools pinpoint where your app might be struggling, allowing you to optimize those sections. Minimize unnecessary memory allocations and optimize resource handling to ensure your app runs smoothly, even on devices with limited resources. A well-optimized app uses resources efficiently, leading to better user experience. Implementing these performance optimization tips ensures your mobile app delivers a responsive and enjoyable user experience.

LANGUAGE DEVELOPMENT

Exercise 2.6. Learn information from the text and prepare 5-minute talk about mobile application development process. The picture below will help you to structure your speech.



Exercise 2.7. Work in pairs and in turns define the terms in your own words.

1. Mobile application is …
2. Software is …
3. Java is …
4. Markup language is …
5. Cross-platform application is …
6. Debugging is …
7. Maintainability is …
8. Prototype is …
9. Scalability is …
10. Codebase is …

Exercise 2.8. Answer the following questions using information from the text.

1. What are the main differences between native and cross-platform mobile app development?
2. Which programming languages are commonly used for developing iOS and Android apps?
3. What are the key factors to consider when designing a user-friendly mobile app interface?
4. How do you ensure the security of data in a mobile application?
5. What tools and frameworks are popular for mobile app development?
6. How can developers optimize the performance of a mobile app?
7. What is the importance of testing and debugging in the mobile app development process?
8. How do app stores review and approve new applications before they are published?
9. What are some common challenges faced during mobile app development, and how can they be addressed?
10. How does user feedback influence the ongoing development and improvement of a mobile app?

Exercise 2.9. a) Read the passage about cross-platform application development frameworks. If you find unknown words, consult with your dictionary.

Mobile engineers use cross-platform mobile development frameworks to build native-looking applications for *multiple platforms,* such as Android and iOS, using a single *codebase.* Shareable code is one of the *key advantages* this approach has over native app development. Having one single codebase means that *mobile engineers* can save time by avoiding the need to write code for each operating system, accelerating the *development process.*

b) Make up questions so as italicized words and expressions would be the answers.

Exercise 2.10. Search for information on the Internet and complete the table about cross-platform mobile application development tools. Prepare a short report (up to 5-7 minutes) and share information with your groupmates/colleagues.

Cross-platform mobile application development tools

Tool	*Features*	*Price*	*Demo*
React Native	• Code and iterate faster • Supports C++, Java, Swift, Objective C, and Python • Up to 80% of sharable codebase • OTA updates • Plug-n-play custom templates	Free	No
Flutter	• Offers in-depth coding assistance • Expressive and Flexible UI • Native Performance • Rich motion APIs	Free	No
Ionic			
Xamarin			
Unity			

IMPROVE YOUR TRANSLATION SKILLS

Learn vocabulary from Unit 2 “APPLICATION DEVELOPMENT. MOBILE APPLICATION DEVELOPMENT” and translate the following sentences from Russian into English.

1. Разработка мобильных приложений включает в себя проектирование, программирование и тестирование программного обеспечения для смартфонов и планшетов.
2. Для создания мобильных приложений используют языки программирования такие как Swift, Kotlin и Java.
3. Важной частью разработки является создание удобного и интуитивно понятного пользовательского интерфейса.
4. Мобильные приложения должны быть оптимизированы для быстрого запуска и работы без ошибок.
5. Разработчики часто используют специальные фреймворки и инструменты, такие как React Native или Flutter, для кроссплатформенной разработки.
6. Тестирование приложений помогает выявить и исправить ошибки перед их выпуском в магазин приложений.
7. В процессе разработки важно учитывать требования безопасности и защиты данных пользователей.
8. Обновление и поддержка мобильных приложений позволяют сохранять их актуальность и улучшать функциональность.
9. Создание мобильных приложений требует командной работы дизайнеров, разработчиков и тестировщиков.
10. С развитием технологий мобильная разработка становится все более востребованной и перспективной областью IT-индустрии.

FOCUS ON WRITING

Elements of writing: cohesion, definitions, examples, generalizations

Cohesion

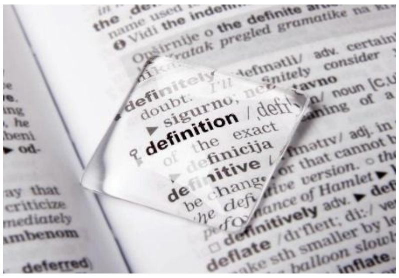
Cohesion means linking phrases together so that the whole text is clear and readable. It is achieved by several methods, such as the use of conjunctions and reference words and phrases, which refer back to something mentioned before.

Writing Exercise 1. Read the following text and complete the table below. What elements of cohesion do you notice in the text?

Ken Thompson (1943) was the principal inventor of UNIX. It is a multiuser, multitasking operating system. He received his Bachelor's and Master's degrees, both in electrical engineering, from the University of California at Berkeley (UCB). Soon thereafter, in 1966, he was hired by Bell Labs. The latter appears to be a research and development arm of AT&T. In 1973, Thompson made his first public presentation about UNIX. It was noticed by the right people at UCB, and this led to the first copy of the operating system being shipped to that university. The UNIX source code is distributed freely throughout the 70s, and it soon became popular at universities and research labs.

Reference	*Reference word or expression*
Ken Thompson	
UNIX	
The UNIX source code	
First public presentation	
The latter	
Noticed	

Definitions



Definitions are normally needed in two situations:
a) In introductions, to clarify a word or phrase in the title;
b) More generally, to explain a word or phrase that may be either very technical (and so not in normal dictionaries), or very recent, or with no widely agreed meaning.

More complete definitions may be written by adding examples or extra information:

Virtual reality (VR) is a term that applies to computer-simulated environments that can simulate physical presence in places in the real world, as well as in imaginary worlds.

Writing Exercise 2. Complete and extend the following definitions. Use your dictionary if necessary.

1. Descriptive geometry is … which allows … by using … .
2. Unreal Engine is …. which helps … .
3. Object-Orienting Programming is … that helps … .
4. Computational science (or scientific computing) is … concerned with … and using computers … .
5. A complex system is … composed of … that as a whole exhibit … .

Examples

When writing essays/articles it is often better to support statements by giving examples. Phrases for introducing examples include:

- for instance/for example
- such as
- e.g.
- particularly
- especially
- a case in point (for single examples).

Writing Exercise 3. Find sentences containing examples in articles related to your research topic. You should find at least 10 sentences. Answer the following questions:

- Are examples necessary in the articles?
- What phrases does the author use to introduce examples?

Generalizations

In written work, generalisations are very useful because they can be used to present complex ideas or data in a simple form. There are two ways of making a generalisation:

a) Using the plural: Lambert's model for body reflection is useful in computer graphics.

b) Using definite article and the singular: The model for body reflection is a useful model.

It is better to avoid absolute phrases, use more cautious phrases instead. When making generalisations it is easy to overgeneralize, using inadequate data.

Writing Exercise 4. Write generalizations on the following topics.

Concept/Notion	*Generalization*
1. Algorithm	
2. Visual Programming Environment (VPE)	
3. Computer Animation	
4. Virtual Reality	

VOCABULARY LIST OF UNIT 2 "APPLICATION DEVELOPMENT. MOBILE APPLICATION DEVELOPMENT"

accomplish, *v.*
adhere, *v.*
adherence to deadline, *expr.*
agile methodology, *expr.*
assume, *v.*
assumption, *n.*
clear vision, *expr.*
codebase, n.

complexity, *n.*
concise, *adj.*
convoluted algorithm, *expr.*
cutover, *n.*
decipher, *v.*
dependency, *n.*
descriptive variable, *expr.*
divert, *v.*
downward, *adv.*
elevate, *v.*
eliminate, *v.*
emerge, *v.*
encapsulate, *v.*
engagement rate, *expr.*
explicit, *adj.*
force-fit, *v.*
foster, *v.*
implement, *v.*
iterate, *v.*
leverage, *v.*
line out, *ph.v.*
maintainability, *n.*
map out, *ph.v.*
memory allocation, *expr.*
meticulous, *adj.*
modify, *v.*

modularity, *n.*
modularization, *n.*
native app, *expr.*
native shell, *expr.*
ongoing maintenance, *expr.*
OTA
overly, *adv.*
performance, *n.*
performant, *adj.*
pinpoint, *v.*
Rapid Application Development (RAD) Methodology, *expr.*
recurring, *part.*
redundancy, *n.*
refactor, *v.*
resolution, *n.*
reusability, *n.*
robust, *adj.*
scalability, *n.*
sequence, *n.*
sign off, *ph.v.*
spacing, *ger.*
specification, *n.*
sprint, *n.*
squad, *n.*
strive for, *expr.*
sustainability, *n.*

technical writer, *expr.*
troubleshooting, *ger.*
unified vision, *expr.*
unit test, *expr.*
waterfall methodology, *expr.*

UNIT 3 “GAME DEVELOPMENT’

LEAD-IN

1. What are the main stages involved in the game development process?
2. Which programming languages are commonly used for creating video games?
3. How does game design influence player engagement and experience?
4. What tools and engines, such as Unity or Unreal Engine, are popular for game development?
5. What skills are essential for a successful career in game development?

READING I

game engine, *expr.*	игровой движок
game console, *expr.*	игровая приставка
renderer, *n.*	рендерер *(проф.),* аппаратное устройство, выполняющее рендеринг изображений
physics engine, *expr.*	физический движок, подсистема в компьютерных играх, отвечающая за симуляцию физики тела
collision detection, *expr.*	обнаружение столкновений, детектирование столкновений
collision response, *expr.*	ответная реакция на столкновение
threading, *ger.*	многопоточность, организация поточной обработки
scene graph, *expr.*	граф сцены

integrated development environment (IDE)	интегрированная среда разработки
enable, *v.*	делать возможным, давать возможность, создавать условия *(context)*
data-driven manner, *expr.*	управляемый данными стиль
preempt, *v.*	вытеснять, приобретать преимущественное право
middleware, *n.*	промежуточное программное обеспечение, промежуточные программные средства
time-to-market, *expr.*	период от начала разработки компьютерной игры до выхода ее на рынок
game source-code, *expr.*	исходный код игры
comprise, *v.*	состоять из, включать
custom engine, *expr.*	клиентский движок
extensibility, *n.*	расширяемость
rendering capability, *expr.*	возможность визуализации
inconsistently, *adv.*	нестабильно, несовместимо, сбивчиво *(context)*
sprite, *n.*	спрайт *(проф.),* динамический графический элемент (small pictogram, used in games; graphical image that can move within a larger graphics)

Game Engines

A game engine is a software framework primarily designed for the development of video games and generally includes relevant libraries and support programs. The “engine” terminology is similar to the term “software engine” used in the software industry. Game engine can also refer to the development software utilizing this framework, typically offering a suite of tools and features for developing games.

Developers can use game engines to construct games for video game consoles and other types of computers. The core functionality typically provided by a game engine may include a rendering engine ("renderer") for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and video support for cinematics.

Game engine implementers often economize on the process of game development by reusing/adapting, in large part, the same game engine to produce different games or to aid in porting games to multiple platforms.

In many cases, game engines provide a suite of visual development tools in addition to reusable software components. These tools are generally provided in an integrated development environment to enable simplified, rapid development of games in a data-driven manner. Game-engine developers often attempt to preempt implementer needs by developing robust software suites which include many elements a game developer may need to build a game.

Most game-engine suites provide facilities that ease development, such as graphics, sound, physics and artificial-intelligence (AI) functions. These game engines are sometimes called "middleware" because, as with the business sense of the term, they provide a flexible and reusable software platform which provides all the core functionality needed, right out of the box, to develop a game application while reducing costs, complexities, and time-to-market – all critical factors in the highly competitive video-game industry.

Like other types of middleware, game engines usually provide platform abstraction, allowing the same game to run on various platforms (including game consoles and personal computers) with few, if any, changes made to the game source-code. Often, programmers design game engines with a component-based architecture that allows specific systems in the engine to be replaced or extended with more specialized (and often more expensive) game-middleware components. Some game engines comprise a series of loosely connected game middleware components that can be selectively combined to create a custom engine, instead of the more common approach of extending or customizing a flexible integrated product.

Extensibility remains a high priority for game engines due to the wide variety of uses for which they are applied. Despite the specificity of the name "game engine", end-users often re-purpose game engines for other kinds of interactive applications with real-time graphical requirements - such as marketing demos, architectural visualizations, training simulations, and modeling environments.

Some game engines only provide real-time 3D rendering capabilities instead of the wide range of functionality needed by games. These engines rely on the game developer to implement the rest of this functionality or to assemble it from other game-middleware components. These types of engines are generally referred to as a "graphics engine", "rendering engine", or "3D engine" instead of the more encompassing term "game engine". This terminology is inconsistently used, as many full-featured 3D game engines are referred to simply as "3D engines". Examples of graphics engines include Crystal Space, Genesis3D, Irrlicht, OGRE, RealmForge, Truevision3D, and Vision Engine.

Modern game or graphics engines generally provide a scene graph – an object-oriented representation of the 3D game-world which often simplifies game design and can be used for more efficient rendering of vast virtual worlds.

As technology ages, the components of an engine may become outdated or insufficient for the requirements of a given project. Since the complexity of programming an entirely new engine may result in unwanted delays (or necessitate that a project restarts from the beginning), an engine-development team may elect to update their existing engine with newer functionality or components.

LANGUAGE DEVELOPMENT

Exercise 3.1. Give English equivalents to the following words and word combinations.

Соответствующие библиотеки; игровая консоль; механизм визуализации; графический движок; многопоточность; поддержка локализации; специалист по внедрению, разработчик; способ управления данными; упреждать; надежное программное обеспечение (ПО); межплатформенное программное обеспечение (ПО); срок вывода на рынок; игровой исходный код; нестандартный движок; включать ряд компонентов; каждый раз по разному, непостоянно; тренировочное моделирование; гибкий интегрированный продукт; быть замененным; более эффективная передача.

Exercise 3.2. a) Define the part of speech of the following words. Write your answers in the table below.

Functionality, insufficient, interactive, unwanted, reusable, implementer, complexity, selectively, restart, discontinued, currently, informative, indefinitely.

	Noun	*Verb*	*Adjective*	*Adverb*	*Participle II*
Word					

b) Make up 5 sentences of your own using any words from part a of this exercise.

Exercise 3.3. Answer the following questions using information from the text.

1. What is a game engine?
2. What does a game engine include?
3. What types of engines are referred to as “graphics” engines? Can you give any examples of them?
4. What do game engines provide?
5. Is it necessary to know how to add realism to a character, how to add graphics effects or how to animate a sprite? Why?
6. What can you say about development of engines as technology ages?

Exercise 3.4. Read the text and decide if the following sentences are true or false.

1. The "engine" terminology is not entirely similar to the term "software engine" used in the software industry.
2. Developers can use game engines to construct games for video game consoles and other types of computers.
3. The core functionality typically provided by a game engine may include "3D engine" for 2D or 3D graphics.
4. In many cases, game engines provide a suite of visual development tools in addition to reusable software components.
5. Complexity of programming is the only requirement in today's demanding market of video-game industry.
6. Modern game or graphics engines generally provide a scene graph – an object-oriented representation of the 3D game world.
7. End-users often re-purpose game engines for other kinds of interactive applications with real-time graphical requirements.
8. Game engine implementers never economize on the process of game development because they want to produce different games.

Exercise 3.5. Match the word or word combination from the text with its definition.

Word or word combination	*Definition*
1. A physics engine	a. a core component of a complex software system.
2. Competitive	b. the art of creating three-dimensional (3D) images or animations showing the attributes of a proposed architectural design.
3. A scene graph	c. a physics model typically implemented in software for use in computer games
4. Visualization	d. strongly desiring to be more successful than others; as good as or better than others of a comparable nature
5. A game engine	e. any technique for creating images, diagrams, or animations to communicate a message
6. Rendering	f. a general data structures commonly used by vector based graphics editing applications and modern computer games

READING II

game development pipeline, *expr.*	процесс разработки игры, этапы разработки игры
bottleneck, *n.*	фактор, сдерживающий рост
AAA game	triple A, в индустрии компьютерных игр термин используется для обозначения наиболее высокобюджетных игр, которые рассчитаны на

	массовую аудиторию. Разработка подобных игр требует огромных затрат и связана с высокими экономическими рисками. Примеры игр ААА: *Grand Theft Auto V, Red Dead Redemption 2, Elden Ring, The Witcher 3 и Cyberpunk 2077*
indie game	инди-игра, компьютерная игра, созданная отдельным разработчиком или небольшим коллективом без финансовой поддержки издателя компьютерных игр. Примеры инди-игр: *Minecraft, Broforce, Celeste, Tacoma, Super Meat Boy, Alto's Adventures, Braid, World of Goo*
game design document, *expr.*	дизайн-документ игры
gameplay, *n.*	процесс игры, ход игры
restrain, *v.*	ограничивать, сдерживать
run over, *ph.v.*	превышать запланированные расходы *(context)*
pitch, *v.*	подать идею инвестору
asset, *n.*	цифровой объект в игре
conscious, *adj.*	осознанный, осмысленный
placeholder, *n.*	заполнитель (графический элемент, заменяемый реальным элементом)
vertical slice, *expr.*	вертикальный срез *(проф.)*
relegate, *v.*	направлять, переводить в низшую категорию
patch, *n.*	патч *(проф.),* временное исправление в программе
DLC	загружаемый контент (downloadable content)
sequel, *n.*	сиквел *(проф.),* продолжение игры

Video Game Development Pipeline

The game development pipeline is the process of building a video game from concept to completion. Much like a production line, the game development pipeline helps organize the flow of work so that everyone knows what they need to deliver and when.

The pipeline also helps manage the game development timeline and budget, reducing inefficiencies and bottlenecks. While pipelines vary between projects and studios, the process is fairly similar whether you're working on an AAA, indie, or mobile game.

A game is continuously evolving, and what sounds great in theory may not work so well in practice. Therefore, the pipeline is not necessarily a linear process. Work must be sent for creative approvals and can often be sent back for revisions. Pipelines must be flexible enough to factor in revisions and course changes.

3 stages of game development

Video game development is typically divided into 3 stages: pre-production, production, and post-production.

1. Pre-production

This is where every project begins. Essentially, pre-production defines the game topic, why it should be made, and what it will take to make it. You might have a great idea for a type of game, a story you want to bring to life, or you may want to build one that leverages a certain type of technology (e.g. VR, a new controller, or console).

In pre-production, you'll find answers to questions like:

- What is the game about?
- Who is the audience?
- Is there a market for it? What's the competition like?
- Which platform will it be published on?
- How will it be monetized? Will it be sold on a platform, or free to play with in-game purchases?
- How long will it take to develop?
- What staff and resources will it require?
- What is the estimated budget?

This stage can last anywhere between a week to a year, depending on the project type, resources, and finances available, and typically takes up to 20% of the total production time. At this point, the team is quite small. There may be a producer, programmer, or concept artist (or, if you're a one-person operation, you'll be doing most of it).

A video game producer handles the business aspect of the project, particularly the financials. They manage the budget and develop marketing strategies to sell the product. A concept artist sets the tone for the project early on by developing artwork and sketches.

These early visuals help form the language of the game, giving everyone working on the project a visual guide to the overall look and feel. The information gathered during this stage of pre-production forms the basis of the Game Design Document.

The Game Design Document (GDD)

A Game Design Document (GDD) is essentially the game north star. It's a living document that helps everyone understand and align with the project greater vision. The GDD includes things like:

- The idea or concept
- Genre
- Story and characters
- Core game mechanics
- Gameplay
- Level and world design
- Art and/or sketches
- Monetization strategy.

As a living document, the GDD is continuously updated and refined throughout production. This could be due to technical or financial restraints or simply realizing that things just don't look, play, or work as well as you had initially hoped.

Many people, especially smaller developers, like to use more agile development techniques, which are less about process and documentation and more about just building things. However, larger studios prefer a different approach.

EA, Microsoft, Sony, Ubisoft and other large game companies are highly process-driven and require heavy documentation. It's a big part of how they have achieved success over and over again. A GDD keeps you organized, helps identify potential risks, and lets you see ahead of time who you may need to hire/outsource to in order to bring your project to life. Your game idea may seem fairly straightforward, but once you lay it out in a GDD, you might soon realize just how big and resource-heavy your project is.

Projects without a plan are much more likely to run over time and budget. Another reason to have a GDD is to help pitch and finance your game. Potential investors will want to see a solid plan before investing. Finally, the GDD will help you market your product once it's ready to be released.

2. Production

Production is the longest stage of the pipeline, and it's all hands on deck. Ranging anywhere from 1 to 4 years, production is where the game really starts to take shape. The story is refined, assets (characters, creatures, props and environments) are created, the rules of play are set, levels and worlds are built, code is written, and so much more. Almost everything in a video game is a conscious decision. This includes every character, environment, object, as well as the look, colors, sounds, level of difficulty, rules and point-scoring system. However initial ideas don't always translate so well in reality, so as the work is being done, the game is being continuously tested and refined.

Production milestones

There are a number of milestones to hit throughout the game development process.

Prototype: This is the initial test of the game (which happens in pre-production and is described in detail above). Some games may never make it past this stage.

First playable: The first playable gives a much better idea of the look and gameplay. While it is still far from final, placeholders are replaced with higher-quality assets, and artwork is added.

Vertical slice: A vertical slice is a fully playable sample that can be used to pitch your game to studios or investors. Ranging from just a few minutes up to half an hour, a vertical slice provides a first-hand experience of your game.

Pre-alpha: The majority of the content is developed in the pre-alpha stage. At this point in game development, some big decisions will need to be made. Content may get cut, or new elements will need to be added to improve gameplay.

Alpha: The game is "feature complete" meaning the main features have all been added and the game is fully playable from start to finish. Some elements, such as art assets may still need to be added, but controls and functionality should be working properly. The QA testers will be making sure everything is running seamlessly and reporting errors back to the team.

Beta: At this point, all the content and assets are integrated, and the team should be focused on optimization rather than adding new functions or features.

Gold master: The game is final and ready to be sent to the publishing outlet and released to the public.

3. Post-production

Once production is complete and the game has shipped, the game development process continues. Some team members are relegated to maintenance (fixing bugs, creating patches) or creating bonus or downloadable content (DLC), while others may move on to the sequel or the next project.

LANGUAGE DEVELOPMENT

Exercise 3.6. Answer the following questions using information from the text.

1. What are the main stages of the game development pipeline?
2. How does concept art influence the overall game development process?
3. What role does prototyping play in game development?
4. How is asset creation integrated into the game development pipeline?
5. What are common tools used during the game design and development phases?
6. How does version control contribute to managing a game development project?
7. What is the importance of testing and quality assurance in the pipeline?
8. How do developers handle iteration and feedback during game production?
9. What are the key challenges faced during the integration phase of game development?
10. How does deployment and post-launch support fit into the overall pipeline?

Exercise 3.7. a) Read the following passage about testing phase in game development pipeline. If you find any unknown words, consult with your dictionary.

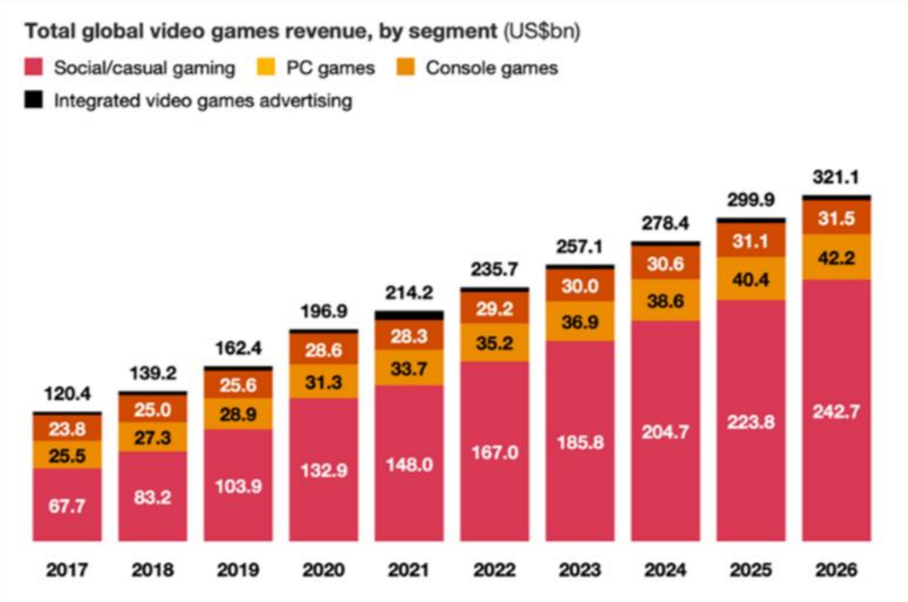
At testing phase the QA engineers, whose work previously went unnoticed, come into play. While developers and designers are busy creating concepts and animating 2D/3D models, game testers meticulously prepare test cases, checklists, and test plans. Once the developers release the environment for testing, the QA team takes over, meticulously examining the game and identifying its most vulnerable areas.

The testing phase typically occurs in multiple iterations - testers report issues to the developers and designers. After these issues are fixed, the QA team reevaluates the game thoroughly. This process is repeated until the developers resolve all the critical problems blocking the game release.

b) Discuss the importance of testing phase in game development with your colleagues. Use vocabulary from the passage above and vocabulary of Unit 3 "GAME DEVELOPMENT'.

c) Search for information on the Internet and prepare report about real cases of bugs fixing in AAA games, indie games or mobile games.

Exercise 3.8. The graph illustrates the increasing demand for video games from 2017 to 2026 approximately. Revise the vocabulary used for graph description (information is presented in "English for Programmers. Part I") and be ready to speak on the topic.



Exercise 3.9. Game development has become an-and-coming field. Prepare a monologue on the topic "What does it take to be a game developer?". Include in your monologue information from the table.

Technical Skills	*Soft Skills*
C++, C#, Java, Python	Effective communication
Game engine	Time management
Unreal Engine	Teamwork
Unity	Handling substantial data

Exercise 3.10. a) Search for information on the Internet and complete the table. There is an example for you.

Game engine	*Programming language*	*Scripting*	*Cross-platform*	*2D/3D oriented*	*Notable games*
GameMaker Studio					
Cocos2d, Cocos2d-x, Cocos2d-html5					
Roblox					
Unity					
Unreal Engine	C++	C++	Yes	3D	Gears of War, Fortnite, Valorant, Unreal series

b) Work in pairs. Take turns and share information you've found on the Internet with your groupmate/colleague.

IMPROVE YOUR TRANSLATION SKILLS

Learn vocabulary from Unit 3 “GAME DEVELOPMENT” and translate the following sentences from Russian into English.

1. Разработка видеоигр включает в себя создание сценария, дизайн уровней, программирование и тестирование.
2. Для разработки игр используют игровые движки, такие как Unity и Unreal Engine.
3. В процессе разработки важно создавать интересный сюжет и привлекательный визуальный стиль.
4. Программирование игровых механик требует знания языков, таких как C# или C++.
5. Геймдизайнеры разрабатывают концепцию игры и определяют ее основные особенности.
6. Тестирование игр помогает выявить баги и улучшить игровой процесс.
7. Создание качественной графики и анимации значительно повышает привлекательность игры.
8. Важной частью разработки является оптимизация производительности для разных устройств.
9. Командная работа дизайнеров, программистов и художников необходима для успешной реализации проекта.
10. Разработка видеоигр — это сложный и творческий процесс, требующий много времени и усилий.

FOCUS ON WRITING

Elements of writing: numbers, references and quotations, style

Numbers

Discussing statistical data is a necessary part of much academic writing. Figures and numbers are both used to talk about statistical data in a general sense. The figures in the report need to be read critically. Digits are individual numbers. Both fractions (1⁄2) and decimals (0.975) may be used.

No final -s on hundred/thousand/million:

In 2024, 77 303 573 people voted for Trump. but: Thousands of processors form a large cluster.

When presenting data, the writer must attempt to be accurate without confusing the reader with too much detail. In some cases, where the actual number is unimportant, words or phrases may replace numbers to simplify the text:

Deciding between <u>the 47 of programs</u> that transfer iPods to computers can be maddening.
Deciding between <u>the dozens of programs</u> that transfer iPods to computers can be maddening.

Table 4. Words and phrases that can be used to describe quality.

Few	Less than expected
Several	3-4
Various	3-6
Dozens of	30-60
Scores of	50-100

Percentages are commonly used for expressing rates of change:

Approximately 49,91% of US adults voted for Trump in 2024.

Table 5. Words and phrases that can be used to simplify statistics in writing.

A small proportion/a large proportion
The least/the most
To double/to halve
On average/the average number
A tenfold increase
Per cent, percentage
Twice/three times as many
The majority/the minority
One in three
A third/a quarter

References and Quotations

A reference is an acknowledgement that you are making use of another writer's ideas or data in your writing. There are four main reasons for giving references:

1. To avoid plagiarism.
2. To give more authority to your writing.

3. To show you are familiar with another research on the topic.
4. To help a reader to find the original source by using the reference section.

In order to give references accurately it is important to stick to the following procedure:

1. Keep a careful record of the details of your sources.
2. Study journals and departmental guidelines to find out which system of referencing is used in your subject area.
3. Use: summary of a writer's ideas, quotation of a writer's words, a mixture of summary and quotation.

Writing Exercise 1. Write a summary of the author's ideas, including references.

Usually, a limited number of such data types come built into a language. Without them, it becomes very difficult to maintain information within a computer program [Nick Bilton and K. Torrace, 2022]. Since the main principle behind computer programming is to take information, process it, and deliver the information in a different form to the user, data types obviously play a large part in determining how this is achieved.

Style

Modern academic writing has a semi-formal, impersonal and objective style. The focus is on presenting information as clearly and accurately as possible. Be as precise as possible when dealing with facts or figures. If it is necessary to estimate numbers use *approximately* rather than *about.* Following are some of the small but specific mistakes in style that are made in formal written work.

Avoid using:

- personal language;
- language that is emotional;
- words that express your opinion too strongly;
- unnecessary words;
- brackets and dashes to add information.

Writing Exercise 2. The following sentences contain examples of inappropriate style. Underline these examples. Rewrite sentences in more formal way.

1. Yes, we know this is a sadly long list of rules, but please read it.
2. You can't always trust those download links that you find.
3. Lots of studies looked at adults rather than kids.
4. I've heard mixed things about this game.
5. Why people insist computers can't think?

VOCABULARY LIST OF UNIT 3 “GAME DEVELOPMENT”

AAA game
asset, *n.*
bottleneck, *n.*
collision detection, *expr.*
collision response, *expr.*
comprise, *v.*
conscious, *adj.*
custom engine, *expr.*
data-driven manner, *expr.*
DLC (downloadable content)
enable, *v.*
extensibility, *n.*
game console, *expr.*
game design document, *expr.*
game development pipeline, *expr.*
game engine, *expr.*
game source-code, *expr.*
gameplay, *n.*
inconsistently, *adv.*
indie game
integrated development environment (IDE)
middleware, *n.*
patch, *n.*

physics engine, *expr.*
pitch, *v.*
placeholder, *n.*
preempt, *v.*
relegate, *v.*
renderer, *n.*
rendering capability, *expr.*
restrain, *v.*
run over, *ph.v.*
scene graph, *expr.*
sequel, *n.*
sprite, *n.*
threading, *ger.*
time-to-market, *expr.*
vertical slice, *expr.*

UNIT 4 “SOFTWARE DEVELOPMENT’

LEAD-IN

1. What are the main stages of the software development lifecycle?
2. How does Agile methodology differ from Waterfall in software development?
3. What are some common programming languages used in software development today?
4. What skills are essential for successful software development?
5. What are the key principles of writing clean and maintainable code?

READING I

address needs, *expr.*	обслуживать потребности *(context)*
intricate, *adj.*	трудный, громоздкий, запутанный
pivotal, *adj.*	основной, важнейший, основополагающий, ключевой
DevOps, *n.*	управление инфраструктурой и развертыванием программного обеспечения, непрерывная интеграция и развертывание
streamline, *n.*	направление, контур
foster, *v.*	способствовать, содействовать
enhance, *v.*	усиливать, повышать, увеличивать, акцентировать *(context)*
silo, *n.*	барьер, разрозненность *(context)*
scalable, *adj.*	масштабируемый
ongoing, *part.*	непрерывный, продолжающийся, постоянный
adherence, *n.*	приверженность, строгое соблюдение
meet user's needs, *expr.*	отвечать потребностям пользователя
manage constraints, *expr.*	управлять ограничениями
syntax, *n.*	синтаксис языка программирования
Git, *n.*	Гит, распределенная система управления версиями
code repository, *expr.*	репозиторий кода
unit test, *expr.*	модульный тест
integration test, *expr.*	комплексный тест, тестирование системы в целом
user acceptance test, *expr.*	пользовательский тест приемки
relational database, *expr.*	база данных с реляционной структурой
NoSQL database, *expr.*	система управления данными категории NoSQL, нереляционная база данных
in-memory database, *expr.*	база данных в оперативной памяти

encryption, *n.*	шифрование
vulnerability assignment, *expr.*	распределение уязвимостей
stakeholder, *n.*	крупный акционер, заинтересованная сторона; сторона, заинтересованная в результатах деятельности компании *(context)*
Integrated Development Environment (IDE)	интегрированная среда разработки
version control system, *expr.*	система управления версиями
hosting, *ger.*	внешнее размещение *(context)*
containearization, *n.*	контейнеризация *(проф.)*
orchestration, *n.*	механизм управления *(context)*
Database management system (DBMS)	система управления базами данных, СУБД
retrieval, *n.*	исправление, выборка данных *(context)*

What is Software Development?

Developing software is a systematic process of designing, coding, testing, and maintaining software applications and systems. It is the driving force behind the digital age, shaping the technological landscape we navigate daily.

The primary purpose of software development is to create efficient, functional, and user-friendly software solutions to address specific needs or problems. Whether it's a mobile app, a website, an operating system, or an intricate business application, software development plays a pivotal role in nearly every aspect of modern society.

When creating software, developers use DevOps, a set of practices and principles that aim to streamline and automate the software development and IT operations processes. It fosters collaboration between development (Dev) and operations (Ops) teams to enhance the efficiency and reliability of software delivery. DevOps encourages the use of automation, continuous integration (CI), continuous delivery (CD), and infrastructure as code to enable faster development cycles, quicker problem resolution, and a more agile approach to developing software. By breaking down silos, DevOps promotes a culture of continuous improvement and collaboration ultimately resulting in more reliable and responsive software systems. It's important to compare DevOps solutions to find what works best.

Key software development concepts

Software engineering is the systematic and disciplined approach to designing software, developing it, and maintaining its systems. It goes beyond coding and focuses on the entire software development lifecycle. Software engineers employ engineering principles and best practices to create reliable, efficient, and scalable software solutions. They analyze requirements, design architecture, write code, perform testing, and ensure ongoing maintenance of software. Collaboration, documentation, and adherence to industry standards help engineers deliver high quality software that meets user needs, while also managing constraints like time and resources.

Here's a breakdown of key concepts software engineers use every day:

It all starts with *coding,* which is the act of writing instructions in a language that a computer can understand. These instructions are called programs or code, and they're written in various programming languages. Each language has its syntax and purpose, making it suitable for distinct tasks. Some popular languages include Python, Java, C++, and JavaScript.

Algorithms and *data structures* are step-by-step procedures or sets of rules designed to solve specific problems or perform specific tasks. They serve as the core logic behind software applications.

Version control is the system that helps developers track changes to their codebase over time. It allows multiple developers to collaborate without overwriting each other's work. Git, a widely used version control system, enables teams to manage their code repositories efficiently.

Testing involves evaluating the software to identify and fix issues or bugs. Developers use various testing techniques, including unit testing, integration testing, and user acceptance testing. Debugging is the process of finding and resolving errors in the code to ensure the software functions correctly.

Databases manage and store data in software applications. They come in various forms, such as relational databases (e.g., PostgreSQL, MySQL), NoSQL databases (e.g., MongoDB), and in-memory databases. Selecting the right database type and designing efficient database structures are critical aspects of software development.

Security protects user data and prevents unauthorized access. Developers must follow best practices for securing applications, including encryption, access controls, and vulnerability assessments.

Software architecture defines the structure and organization of a software system. It involves deciding how components will interact, ensuring scalability, and planning for future development.

Documentation, including code comments, user manuals, and technical documentation, is crucial for understanding and maintaining software. Documentation also provides insights into the software functionality and usage and is essential for providing clarity, context, and guidance to developers and stakeholders.

Tools and technologies

In the world of software development, a multitude of tools and technologies are at your disposal. Here are some of the common ones:

Integrated development environment (IDE): IDEs like Visual Studio Code, IntelliJ IDEA, and Eclipse provide a unified environment for coding, debugging, and testing.

Programming languages: Popular programming languages include Python, Java, JavaScript, Ruby, and C++.

Version control systems: Git is the most widely used version control system, with platforms like GitHub providing hosting and collaboration features.

Containerization and orchestration: Docker and Kubernetes are key factors in containerizing applications and managing them at scale.

Testing frameworks: Testing frameworks like JUnit and Selenium help automate testing processes.

Database management systems (DBMS): MySQL, PostgreSQL, MongoDB, and Oracle are examples of DBMS used for data storage and retrieval.

LANGUAGE DEVELOPMENT

Exercise 4.1. Give English equivalents to the following words and word combinations.

База данных, архитектура программного обеспечения (ПО), процесс разработки программного обеспечения (ПО), система управления базами данных (СУБД), интегрированная среда разработки, тестирование и отладка кода, документирование технических требований, использование библиотек, развертывание приложения, модель жизненного цикла проекта, обеспечение безопасности данных, отвечать потребностям пользователя, модульный тест, механизм управления.

Exercise 4.2. Answer the following questions using information from the text.

1. How does requirements gathering impact the success of a software project?
2. What are some common challenges faced during software development?
3. What are some popular version control systems used in software development?
4. How do integrated development environments (IDEs) improve the coding process?

5. What is the role of continuous integration tools in modern software development?
6. What are the benefits of using containerization tools like Docker?
7. How do code review tools help maintain code quality?
8. What are some common challenges faced during software development?

Exercise 4.3. a) Make up six questions based on information from the text.

1. What ______________ ?

2. When ______________?

3. Who ______________ ?

4. Why ______________ ?

5. How _______________?

6. What kind of _________ ?

b) Work with your groupmate/colleague. Ask each other questions from part a of this exercise and answer them.

Exercise 4.3. Match type of software engineer's work with its definition.

Type of software engineer's work	*Definition*
1. Coding	a. process of running an application on a server or device
2. Testing	b. using various tools to ensure that an application is running at its peak efficiency after it's been deployed
3. Deployment	c. process of writing a computer program
4. Monitoring	d. reviewing the quality of software and risk of its failure before and after deployment
5. Pair programming	e. a software development technique in which two programmers work together at one workstation
6. Collaboration	f. putting the project plan into action
7. Implementation	g. working together as a team on a project that is too big for one person

Exercise 4.4. Match word or word combination from column A with the word or word combination from column B. Sometimes more than one variant of collocation is possible.

A	*B*
To work through	To solve problems faster
To maintain	A code
To figure out	A project
To enjoy	Coding
To do	Working with people
To deploy	Software
To implement	Your own thing
To tend	A solution
To find out	The answer
To be fascinated by	An issue

Exercise 4.5. Choose the correct answer (a, b, c or d). Sometimes more than one answer is correct.

1. What is true about the day-to-day of a software engineer?

   a. There are no meetings on the plan of implementation.
   b. There is a lot more to it than coding.
   c. Every day is the same.
   d. There can be work together with another coder.

2. What is true about pair programming?

   a. The navigator/passenger is problem solving.
   b. The navigator/passenger is typing.
   c. It allows you to learn.
   d. You do it only when you can’t solve a problem on your own.

3. What skills should a software engineer have?

   a. Non-verbal communication.
   b. Maintaining software.
   c. Coding.
   d. Organizational skills.

4. What is software engineering a mix of?

   a. Teamwork and collaboration.
   b. Collaboration and pair programming.
   c. Collaboration and working on your own.
   d. Teamwork and solo work.

READING II

streamline, *v.*	упрощать, систематизировать, приводить в соответствие *(context)*
rely on, *ph.v.*	рассчитывать на, положиться, учитывать *(context)*
wide array, *expr.*	обширный массив
tailor, *v.*	адаптировать, приспосабливать *(context)*
front-end framework, *expr.*	фреймворк на стороне клиента
back-end language, *expr.*	язык бэкенда *(проф.),* язык серверной разработки
no-code platform, *expr.*	платформа бескодовой разработки
low-code platform, *expr.*	платформа малокодовой разработки
democratize, *v.*	демократизировать, делать демократическим
cater to, *v.*	обслуживать, удовлетворять *(context)*
AI	Artificial Intelligence, искусственный интеллект (ИИ)
safeguard, *v.*	гарантировать, охранять
pivotal role, *expr.*	основная роль
backbone, *n.*	основа, опора *(context)*
ignite, *v.*	положить начало *(context)*
akin to, *expr.*	схожий с
terrain, *n.*	участок *(context)*
realm, *n.*	область, сфера *(context)*
forge into reality, *expr.*	воплощать в реальность *(context)*
comprehensive, *adj.*	обширный, комплексный, многогранный *(context)*
renowned, *part.*	известный, признанный *(context)*
syntax highlighting, *expr.*	выделение синтаксиса программы
cater, *v.*	принимать во внимание, пытаться угодить *(context)*

development workflow, *expr.*	процесс разработки
drastically, *adv.*	решительно, радикально
extension, *n.*	программное средство для обеспечения дополнительных возможностей
cornerstone, *n.*	основа, фундамент
toolkit, *n.*	инструментальный комплект, пакет инструментальных средств разработки
prowess, *n.*	совершенство, мастерство
refactoring, *ger.*	переработка структуры кода, реструктуризация кода
exemplify, *v.*	пояснять, приводить пример
bog down, *ph.v.*	препятствовать, задерживать
mundane task, *expr.*	обычная задача
version control system, *expr.*	система управления версиями
manageable, *adj.*	удобный в управлении
hub, *n.*	блок *(context)*
Jira, *n.*	система управления заданиями *(проф.)*
synergy, *n.*	совместная деятельность, согласование действий
code consistency, *expr.*	корректность кода
underscore, *v.*	делать акцент, акцентировать, подчеркивать
paramount, *adj.*	первостепенный, первоочередной, главнейший *(context)*
agile, *adj.*	адаптивный, оперативный
iterative, *adj.*	повторяющийся, итеративный
mere, *adv.*	только, лишь
fast-paced, *part.*	быстро развивающийся, стремительный, задающий быстрый темп *(context)*
within scope, *expr.*	в рамках
thrive, *v.*	процветать
forefront, *n.*	центр, важнейшее место *(context)*

build automation, *expr.*	автоматизация сборки

Software Development Tools

The right software development tools streamline the entire development lifecycle, improving efficiency and collaboration with IDEs, version control systems, and agile project management tools being central to this process.

Software development relies on a wide array of specialized tools tailored to specific layers of application development, including front-end frameworks, back-end languages, and mobile and cross-platform solutions.

The selection of software development tools must consider a balance between cost, complexity, and capability, with no-code and low-code platforms emerging as cost-effective solutions that democratize development and cater to various user expertise levels.

AI-powered tools like *GitHub Copilot* and design-to-code platforms like *Locofy* make coding more efficient and accessible.

Navigating the Landscape of Software Development Tools

The landscape of software development tools is as diverse as it is dynamic. From the code editors that serve as the canvas for our digital creations to the version control systems that safeguard our progress, each software development tool plays a pivotal role in the software development process.

They are the backbone that supports software development teams throughout the lifecycle of creating, maintaining, and enhancing software applications. They are the catalysts that ignite efficiency, collaboration, and innovation, ensuring that the development journey is as smooth as possible.

Selecting the right software development tools is akin to selecting the right crew for a voyage; with the proper expertise, the journey is not only successful but also enjoyable.The right tools can significantly reduce the time required to build custom software products, ensuring timely delivery and cost-efficiency.

Yet, navigating this terrain requires a thorough understanding of the various types of tools available and their capabilities. Let’s dive deeper into the world of Integrated Development Environments (IDEs), version control systems, agile project management tools to discover how they each contribute to making software development accessible and efficient for teams around the globe.

Understanding Integrated Development Environments (IDEs)

In the realm of software development, Integrated Development Environments (IDEs) are the sanctuaries where ideas are forged into reality. They offer a comprehensive suite of tools within a single environment, allowing for the swift creation, testing, and debugging of code. IDEs such as *Microsoft Visual Studio* and *IntelliJ IDEA* are renowned for their intelligent code completion and syntax highlighting features, which are instrumental in aiding developers to write efficient, error-free code. These environments cater not just to Java developers but to a wide array of programming languages, offering smart code editing that streamlines development workflows.

The intuitive interface of an IDE can drastically enhance a developer's productivity. By integrating debugging tools, built-in Git support, and the ability to add extensions for additional functionalities, IDEs serve as the cornerstone of a developer's toolkit. For example, the prowess of *IntelliJ IDEA* in handling multiple programming languages and frameworks, along with its refactoring capabilities, exemplifies how an IDE can elevate the quality of software projects. Mastery of these environments can significantly enhance the development workflow, enabling developers to focus on innovation rather than getting bogged down by mundane tasks.

The Power of Version Control Systems

Version control systems are guardians of the software development process, ensuring that the history of a project is preserved and manageable. They enable software development teams to track changes, manage multiple versions of software, and collaborate effectively, which is essential for maintaining code integrity. Platforms like *GitHub* have emerged as hubs for collaborative efforts, offering features that support code security and peer review, enhancing the quality of software applications. Similarly, *Bitbucket* integration with *Jira* exemplifies the synergy between version control and project management, streamlining development workflows and improving collaboration.

Adopting a distributed version control system like *GitLab* illustrates the industry push towards more collaborative and integrated development environments. With features that ensure code consistency and quality analysis, *GitLab* underscores its role in managing the software development lifecycle. Moreover, the importance of version control extends beyond coding tasks to managing user permissions and tracking contributions, which is paramount for copyright management in software projects.

Embracing Agile Project Management Tools

Agile project management tools are the compasses that guide software development teams through the iterative and dynamic nature of their projects. Tools such as *Jira* and *ClickUp* are not mere task managers; they are enablers of efficiency within the Agile framework, offering planning, tracking, and coordination capabilities that are essential for fast-paced development. The specialized features provided by these tools manage sprints, automate workflows, and set priorities, ensuring that software projects remain on track and within scope.

Nifty and *ClickUp* stand out with their customizable workflows and intuitive user interfaces, fostering an environment where complex workflows become manageable and team collaboration thrives. These tools are more than just software applications; they are the catalysts for rapid application development, enabling teams to adapt their procedures to the unique demands of their projects.

As agile project management continues to evolve, these tools remain at the forefront, empowering developers to deliver exceptional software applications with agility and precision.

LANGUAGE DEVELOPMENT

Exercise 4.6. Answer the following questions using information from the text.

1. What are some popular version control systems used in software development?
2. How does a continuous integration tool improve the development process?
3. What is the purpose of an integrated development environment (IDE)?
4. Can you name some common project management tools used by software teams?
5. How do debugging tools assist developers in writing better code?
6. What is the role of build automation tools in software development?
7. Which tools are typically used for code review and collaboration?
8. How do containerization tools like Docker benefit software deployment?
9. What are some testing frameworks commonly used in software development?
10. How do issue tracking tools help manage bugs and feature requests?

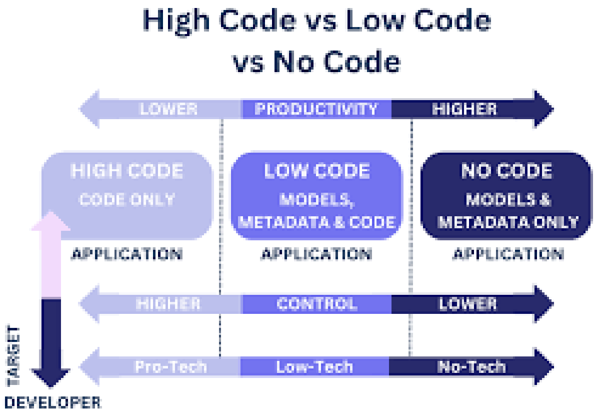
Exercise 4.7. Revise vocabulary of the text “Software Development Tools” and translate the following word combinations from Russian into English.

- Среда разработки
- Инструменты автоматизации сборки
- Средства для непрерывной интеграции
- Контейнерные технологии
- Средства тестирования программного обеспечения (ПО)
- Инструменты для отслеживания багов
- Платформы для совместной работы разработчиков

Exercise 4.8. Work in pairs and in turns define the terms in your own words.

1. Integraded Development Environment (IDE) is …
2. Version Control System is …
3. Low-code platgorm is …
4. HitHub is …
5. Development team is …
6. Code consistence is …
7. Build automation is …
8. Embedded software development is …

Exercise 4.9. The democratization of software development is driven by the emergence of low-code and no-code platforms. These tools enable people with minimal programming knowledge to create and deploy applications, significantly reducing development time. Speak on this topic using the picture on the right as a reference.



Exercise 4.10. a) Search for information on the Internet and complete the table. There is an example for you.

Software Development Tool	*Category*	*Developer*	*Latest Version*	*Platform*	*Price*
Visual Studio Code	IDE	Microsoft	1.97 (was released January 2025)	Windows, macOS, Linux, Web	Free, Open-source
Eclipse					
SQL Server Management Studio (SSMS)					
GitHub					

Table from ex. 4.10 continued

Bitbucket					
GitLab CI/CD					

b) Work in pairs. Take turns and share information you've found on the Internet with your groupmate/colleague.

IMPROVE YOUR TRANSLATION SKILLS

Learn vocabulary from Unit 4 "SOFTWARE DEVELOPMENT" and translate the following sentences from Russian into English.

1. Инструменты контроля версий помогают разработчикам отслеживать изменения в коде и работать в команде.
2. Среда разработки (IDE) обеспечивает удобное написание, редактирование и отладку программного кода.
3. Автоматизированные системы сборки ускоряют процесс компиляции и тестирования приложений.
4. Инструменты для тестирования позволяют выявлять ошибки и повышать качество программного обеспечения.
5. Средства для управления проектами помогают планировать задачи и отслеживать прогресс разработки.
6. Контейнеризационные технологии, такие как Docker, облегчают развертывание приложений в различных средах.
7. Инструменты для анализа кода помогают находить потенциальные уязвимости и улучшать структуру программы.
8. Системы для непрерывной интеграции автоматизируют процесс сборки и тестирования новых версий ПО.
9. Средства мониторинга позволяют отслеживать производительность и стабильность приложений в реальном времени.
10. Инструменты для совместной работы способствуют эффективному взаимодействию команд разработчиков и дизайнеров.

FOCUS ON WRITING

Prefixes and Suffixes

Adding affixes to existing words (the base) to form new words is common in academic English. If you know how prefixes and suffixes affect word meaning it is easy to understand the word. Prefixes are added to the front of the base (like – *dis*like), whereas suffixes are added to the end of the base (active – activ*ate*).

Prefixes usually do not change the class of the base word, but suffixes usually do change the class of the word. Prefixes change or give the meaning. Suffixes show the meaning or the word class. The most common prefixes used to form new verbs in academic English are: *re-, dis-, over-, un-, mis-, out-*. The most common suffixes are: *-ise, -en, -ate, -(i)fy.*

Table 6. Prefixes with their meaning.

Prefix	*Meaning*	*Examples of verbs*
Dis-	reverses the meaning of the verb	disappear, disallow, disarm
Over-	too much	overbook, oversleep, overwork
Un-	reverses the meaning of the verb	unbend, uncouple, unfasten
Mis-	badly or wrongly	mislead, misinform, misidentify
Out-	more or better than others	outperform, outbid
Be-	make or cause	befriend, belittle
Co-	together	co-exist, co-operate, co-own
De-	do the opposite of	devalue, deselect
Fore-	earlier, before	foreclose, foresee
Inter-	between	interact, intermix, interface
Pre-	before	pre-expose, prejudge, pretest

Table 7. Suffixes that make nouns, adjectives and verbs.

Suffix	*Meaning*	*Examples of nouns, adjectives and verbs*
-tion	Action	alteration, demonstration
-ity	state or quality of being	ability, similarity, responsibility
-er	person	advertiser, driver, astronomer, geographer
-ness	state or quality of being	darkness, preparedness, consciousness
-ism	doctrine	Thatcherism
-wards	mean in the direction	northwards
-ant/-ent	person who does sth.	assistant, consultant, student
-ship	state of being	friendship, leadership
-age	collection of sth.; result of action	baggage, breakage, package

Writing Exercise 1. a) Define part of speech of the following words.

b) Write definitions in column B. If necessary, use your dictionary.

A	*B*
1. Saleable	*Definition:*
2. Uncooperatively	*Definition:*
3. Interviewee	*Definition:*
4. Surrealism	*Definition:*
5. Symbolically	*Definition:*

Writing Exercise 2. Translate the sentences into Russian painting attention to the words in italics. Give possible meanings to them.

1. It is claimed that computers have created a *post-industrial* economy.
2. The *unavailability* of the product is due to the *exceptional* weather.
3. When we arrived, we were told that due to a virus in the computer there was a problem of *overbooking.*

4. Our team managed to *outperform* our competitors by delivering the project ahead of schedule and with higher quality.
5. Students learn best when they have the opportunity to *interact* actively with their teachers and classmates.

VOCABULARY LIST OF UNIT 4 “SOFTWARE DEVELOPMENT”

address needs, *expr.*
adherence, *n.*
agile, *adj.*
AI (Artificial Intelligence)
akin to, *expr.*
backbone, *n.*
back-end language, *expr.*
bog down, *ph.v.*
build automation, *expr.*
cater to, *v.*
cater, *v.*
code consistency, *expr.*
code repository, *expr.*
comprehensive, *adj.*
containearization, *n.*
cornerstone, *n.*
Database management system (DBMS)
democratize, *v.*
development workflow, *expr.*

DevOps, *n.*
drastically, *adv.*
encryption, *n.*
enhance, *v.*
exemplify, *v.*
extension, *n.*
fast-paced, *part.*
forefront
forge into reality, *expr.*
foster, *v.*
front-end framework, *expr.*
Git, *n.*
hosting, *ger.*
hub, *n.*
ignite, *v.*
in-memory database, *expr.*
Integrated Development Environment (IDE)
integration test, *expr.*
intricate, *adj.*
iterative, *adj.*
Jira, *n.*
low-code platform, *expr.*
manage constraints, *expr.*
manageable, *adj.*
meet user's needs, *expr.*

mere, *adv.*
mundane task, *expr.*
no-code platform, *expr.*
NoSQL database, *expr.*
ongoing, *part.*
orchestration, *n.*
paramount, *adj.*
pivotal role, *expr.*
pivotal, *adj.*
prowess, *n.*
realm, *n.*
refactoring, *ger.*
relational database, *expr.*
rely on, *ph.v.*
renowned, *part.*
retrieval, *n.*
safeguard, *v.*
scalable, *adj.*
silo, *n.*
stakeholder, *n.*
streamline, *n.*
streamline, *v.*
synergy, *n.*
syntax highlighting, *expr.*
syntax, *n.*
tailor, *v.*

terrain, *n.*
thrive, *v.*
toolkit, *n.*
underscore, *v.*
unit test, *expr.*
user acceptance test, *expr.*
version control system, *expr.*
version control system, *expr.*
vulnerability assignment, *expr.*
wide array, *expr.*
within scope, *expr.*

ЗАКЛЮЧЕНИЕ

Во второй части пособия “English for Programmers” внимание уделяется профессионально-направленным темам, которые затрагивают различные аспекты разработки, в частности веб-разработку (раздел “Web Development”), разработку приложений, со значительным уклоном в мобильную разработку (раздел “Application Development. Mobile Application Development”), разработку игр (раздел “Game Development”), разработку программного обеспечения (раздел “Software Development”).

Вторая часть пособия “English for Programmers” может быть использована как источник дополнительных профессионально-ориентированных текстов для изучения в классе и организации самостоятельной работы магистрантов ФСУ, ФВС.

Логичная структура пособия облегчает работу с ним и повышает эффективность его использования. Разнообразные тексты базового и повышенного уровня сложности, широкий спектр лексико-грамматических упражнений, задания по переводу способствуют активному вовлечению магистрантов в ученый процесс.

Автор пособия убежден, что высокий уровень владения иностранным языком – ключ к карьерному росту в сфере информационных технологий. Английский язык – необходимый навык для современного программиста, который хочет быть конкурентноспособным, участвовать в масштабных международных проектах и раскрывать свой личностный потенциал.

СПИСОК УСЛОВНЫХ СОКРАЩЕНИЙ

adj.	*adjective,* имя прилагательное
adv.	*adverb,* наречие
context	при переводе слова учитывается контекст предложения/фразы
expr.	*expression,* выражение
fixed expr.	*fixed expression,* фиксированное выражение
n.	*noun,* имя существительное
ph.v.	*phrasal verb,* фразовый глагол
prep.	*preposition,* предлог
ger.	*gerund,* герундий
part.	*participle,* причастие
transl.	*translation,* перевод
v.	*verb,* глагол
проф.	профессионально-ориентированная лексика

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алешугина, Е. А., Лошкарева, Д. А. Профессионально-ориентированный английский язык для специалистов в области информационных технологий / Е. А. Алеуштина, Д. А. Лошкарева. – Нижний Новгород : Издательство ННГАСУ, 2020. – 104 с.

2. Беседина, В.Г. English for IT students / В. Г. Беседина. – Барнаул : Издательство АлтГТУ, 2024. – 85 с.

3. Волченкова, К. Н., Колегова, И. А., Истомина, Е. М., Серяпина, Ю. С. English for IT specialists / К. Н. Волченкова, И. А. Колегова, Е. М. Истомина, Ю. С. Серяпина. – Челябинск : Издательский центр ЮУрГУ, 2021. – 145 с.

4. Кожанов, Д. А. Английский язык в сфере информационных технологий / Д. А. Кожанов. – Барнаул : Издательство АлтГПУ, 2017. – 112 с.

5. Малашенко, Е.А. English for IT students / Е. А. Малашенко. – Минск : МГЭУ им. А. Д. Сахарова, 2014. – 140 с.

6. Ситдикова, Ф. Б., Сабирова, Р. Н., Хакимзянова, Д. Ф. English for Masters of Computing / Ф. Б. Ситдикова, Р. Н. Сабирова, Д. Ф. Хакимзянова. – Казань : Издательство КФУ, 2013. – 126 с.

7. Шамсутдинова, Э. Х. English for programmers / Э. Х. Шамсутдинова. – Казань : Издательство Казанского университета, 2016. – 93 с.

8. Иллюстрации, представленные в настоящем пособии, заимствованы из ресурса: URL: hyandex.ru/images/ (дата обращения: 18.08.2025)

ПРИЛОЖЕНИЕ

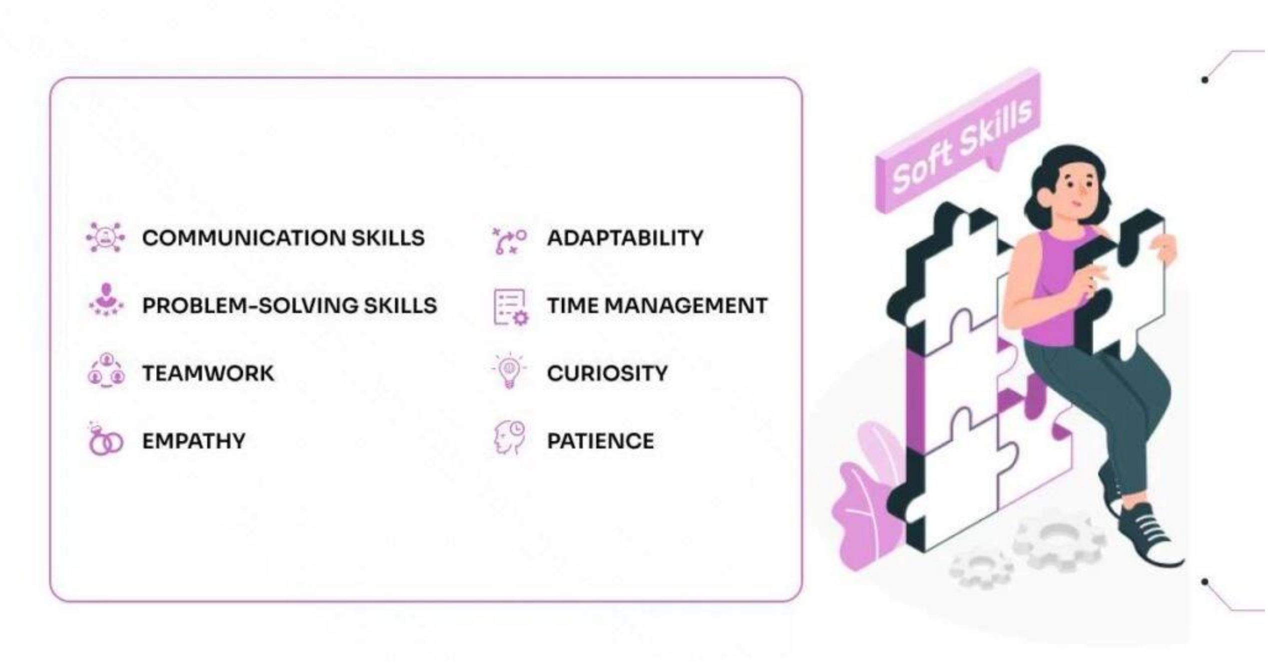
ADDITIONAL ECERCISES

Software Developer Soft Skills

Exercise 1. a) The picture below contains the list of soft skills necessary for IT specialists. Answer the following questions:

- What are the most important soft skills for a programmer/software developer/mobile app developer/game developer?
- Should soft skills be included in a CV?
- How can one develop soft skills?

b) Discuss the questions above with your groupmate/colleague.



Exercise 2. Match the word or words combination with its definition.

Word or word combination	*Definition*
1. Responsibility	a. being willing to consider another viewpoints and alternatives
2. Empathy	b. the combined action of a group, especially when effective and efficient
3. Communication	c. the capacity to be aware of, control, and express one's emotions

4. Emotional Intelligence	d. the state or fact of having a duty to deal with something or of having control over someone
5. Teamwork	e. a strong desire to know or learn something
6. Open-mindedness	f. exchanging information by speaking, writing, or using some other medium
7. Curiosity	g. the ability to understand and share the feelings of another

Exercise 3. a) Fill in prepositions instead of gaps in the text. Some prepositions could be used several times.

For In(x 4) With Of

Ability to Work $^{1.}$ ___ *a Team*

Most software development projects require a united team effort. So, collaboration skills are a must here. You need to find common ground $^{2.}$ ___ your teammates as well as other departments that might be involved. A united team always creates a positive atmosphere. It's important $^{3.}$ ___ everyone to feel support and freedom. To achieve that you can:

- Participate $^{4.}$ ___ peer reviews;
- Be empathetic and listen $^{5.}$ ___ concerns and opinions $^{6.}$ ___ your colleagues;
- Take part $^{7.}$ ___ different activities $^{8.}$ ___ your office.

b) Make up questions to the underlined word combinations. Here is an example for you.

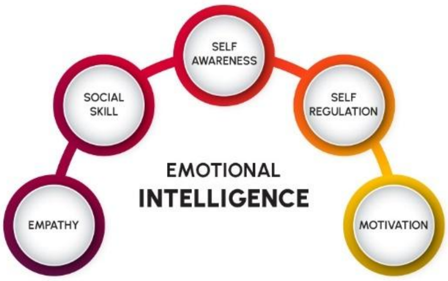
Example: What does require a unites team effort? – Most software development projects.

Exercise 4. Replace the italicized words in the text to their synonyms from the box. Remember to use the synonym in the appropriate grammatical form.
Note: the word **control** will be used two times.

Dwell on contact lessen the self-confidence ability

control (x 2) valuable forget permit

Emotional Intelligence



This *skill* is very important in the software development environment, even though many people *overlook* it. It *allows* you to *manage* your own emotions and *empathize* with others. The work environment can be pressuring, and high emotional intelligence (EQ) helps you *handle* stress calmly and maintain focus. Plus, this skill allows you *to communicate* properly with others and provide *constructive* feedback without *discouraging* colleagues. It's particularly important when you have to work in a team.